Spring 2006

An Implicit Level Set Model for Firespread

Pallop Huabsomboon
Old Dominion University

AN IMPLICIT LEVEL SET MODEL FOR FIRESPREAD

by

Pallop Huabsomboon
B.S. April 1995, Mahidol University
M.S. December 2000, Oregon State University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTATIONAL AND APPLIED MATHEMATICS

OLD DOMINION UNIVERSITY
May 2006

Approved by:

David E. Keyes (Director)

Hideaki Kaneko (Member)

Fang Q. Hu (Member)

Glenn Williams (Member)

Alexander N. Godunov (Member)

ABSTRACT

AN IMPLICIT LEVEL SET MODEL FOR FIRESPREAD

Pallop Huabsomboon
Old Dominion University, 2006
Director: Dr. David E. Keyes

The level set method is a mathematical and computational technique for tracking a moving interface over time. It can naturally handle topological changes such as merging or breaking interfaces. Intrinsic geometric properties of the interface, such as curvature and normal direction, are easily determined from the level set function ϕ. There are many applications of the level set method, including kinetic crystal growth, epitaxial growth of thin films, image restoration, vortex dominated flows, and so forth. Most applications described in the growing literature on the applications of level sets advance the level set equation with explicit time integration. Hence, small CFL-respecting time steps are needed to maintain stability. In this thesis, an implicit level set method is introduced and applied to wildland firespread models, removing vulnerability to instability.

This thesis is dedicated to my wife, *Sansanee Huabsomboon*,
and my lovely daughter,
Natnisha (Jeannie) Huabsomboon.

ACKNOWLEDGMENTS

I am indebted to my advisor, Dr. David E. Keyes, for his guidance and support during the period of my doctoral study at Old Dominion University, much of it remotely. I was privileged and enjoyed working under his supervision.

I would also like to thank my dissertation committee, Dr. Hideaki Kaneko, Dr. Fang Q. Hu, Dr. Glenn Williams, and Dr. Alexader L. Godunov, for their time and effort in reading my dissertation and providing useful feedback.

I am grateful to the Royal Thai Government for sponsoring me to pursue my masters and doctoral degrees during my years in the U.S. I also acknowledge support provided by the U.S. government. This research was supported by the National Science Foundation's Information Technology Research Program through grant ACI-0121207 under the project named *Caliente* (Control, AssimiLation, and InvErsioN for TErascale Simulations). I am also thankful for Dr. John M. Dorrepaal, the chairman of the Department of Mathematics and Statistics, for providing me financial support during the last two semesters.

My thanks also go to all officers of the Office of Educational Affairs both in Washington D.C. and in Thailand, and to the president, the dean of faculty of science and the chairman of Mathematics and Statistics Department of Mahidol University for helping me to get through many difficult situations during my study in the U.S.

Finally, my particular and very special thanks go to my beautiful wife, Sansanee, and my two-year old daughter, Jeannie, for providing me their love, caring, continued support and inspiration to get through years of hard work. I would also like to extend my special thank to my father, Sangwarn, my mother, Ream, my brother and my sister, and my wife's family in Thailand for their support and concern in many ways.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The level set method, introduced by Osher and Sethian in [31], is a numerical technique designed for tracking the evolution of moving interfaces. Level sets can be found in many applications such as crystal growth, image processing, vortex dominated flow and shape recognition. Osher and Fedkiw provide an overview and some recent results on level set methods in [29]. Eulerian level set methods are among many mathematical formalisms used to track interfaces.

Lagrangian marker methods are also popular for tracking moving interfaces. Marker methods involve planting marker particles along the propagating interfaces and follow their movement (see Figure 1). This technique is very economical in terms of storage. They are algorithmically complex, especially for surfaces of co-dimension one in three-dimensional settings, where for large and complex deformations marker points must be removed as they get too close or added as they get too far apart, and where complex data structures are required to accommodate topological change, such as pinch-off or merge, and to prevent tangling of the interface in regions of high curvature. Nevertheless algorithms and software for front tracking have been perfected in recent years [17].

Volume-of-fluid methods, introduced by Noh and Woodward [28], embed the interface in an Eulerian grid in which it is tracked implicitly. A fixed grid is laid over the computational domain (see Figure 2), and a value assigned to each grid cell based on the fraction of the interior currently located in that cell. The cell value is zero if the cell is completely outside the interface. The cell value is one if the cell is completely inside the interface. A fraction between 0 and 1 is assigned to cells that straddle the interface. An advantage of the volume-of-fluid technique is that topological complications boundaries are accommodated without effort. However, calculation of intrinsic geometric properties of the interface, such as curvature and normal direction, is difficult and inaccurate.

Eulerian level set methods are well known for their gracefulness in handling topological changes such as merging or breaking interfaces. The position of the interface at time t is defined by the zero level set of a scalar function of space and time, which

---

This dissertation follows the style of *SIAM Journal on Scientific computing.*

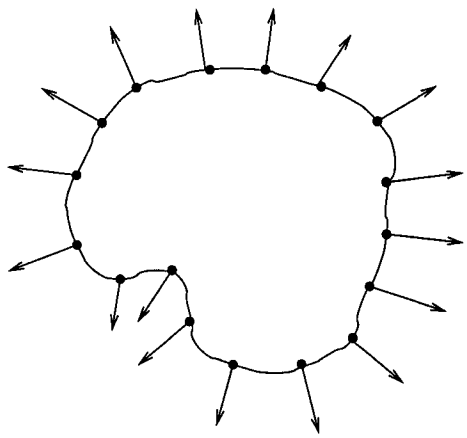

FIG. 1. *Marker particles are planted along the interface and their movement tracked.*

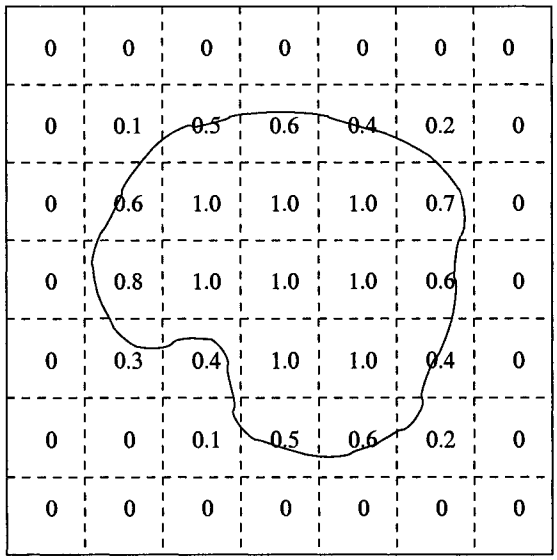

FIG. 2. *A grid is fixed in the computational domain and values assigned to each grid cell based on the fraction of the cell that is interior to the boundary.*

can represent the signed distance to the front (see Figure 3); i.e., $\phi(\vec{x}, t) = 0$. Intrinsic geometric properties of the interface are easily calculated from the level set function ϕ; for example, a normal direction to the interface is given by $\frac{\nabla\phi}{|\nabla\phi|}$, and the curvature of the interface is given by $\nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|}\right)$. Level set methods are relatively easy to implement. They build on the established theory of Hamilton-Jacobi equations see, e.g. [14]. Thus, we can exploit techniques borrowed from the numerical solution of Hamilton-Jacobi equations.

It is well known that solutions of Hamilton-Jacobi equations can develop kinks (or discontinuities in the first derivative) even when the initial condition is smooth. However, viscosity solutions of Hamilton-Jacobi equations can provide a path to existence and uniqueness results. The study of viscosity solutions of Hamilton-Jacobi equations was pioneered by Crandall and Lions in [9, 10]. They introduced a class of monotone first-order accurate numerical schemes for Hamilton-Jacobi equations and proved that these monotone schemes converge to the viscosity solution. Later, Osher and Shu exploited the relationship between conservation laws and Hamilton-Jacobi equations to construct higher-order numerical schemes in [32].

This thesis focuses on the application of level set methods to wildland firespread. Understanding and control of wildfire is a high priority in contemporary forest management. A century of active fire suppression has led to ecosystems out of balance with respect to fire. There are ten of thousands of wildfires each year in the United States [26]. Billions of dollars in property damage occur annually and up to 15 million dollars per day are spent fighting wildfires during peak season. Computational simulation of firespread is employed to predict the speed of wildland fires in order to help containment and manage resources during a fire incident, as well as to plan controlled burns.

In 1982, Anderson et. al. [1] proposed a commonly used firespread model in which the fire appears as an elliptical front with the long axis stretched out in the wind direction. However, such models are too crude for firespread in practice. The USDA Forest Service code FARSITE [15] employs a Huygens mechanism and advances the local firefront by a large number of semi-empirical rules, taking into account local wind, topography, and landcover. Other first-principles approaches, e.g. [7], use conservation of mass, momentum, and energy for modeling and predicting the speed of firefront. However, it is difficult to measure and calculate the required constituent

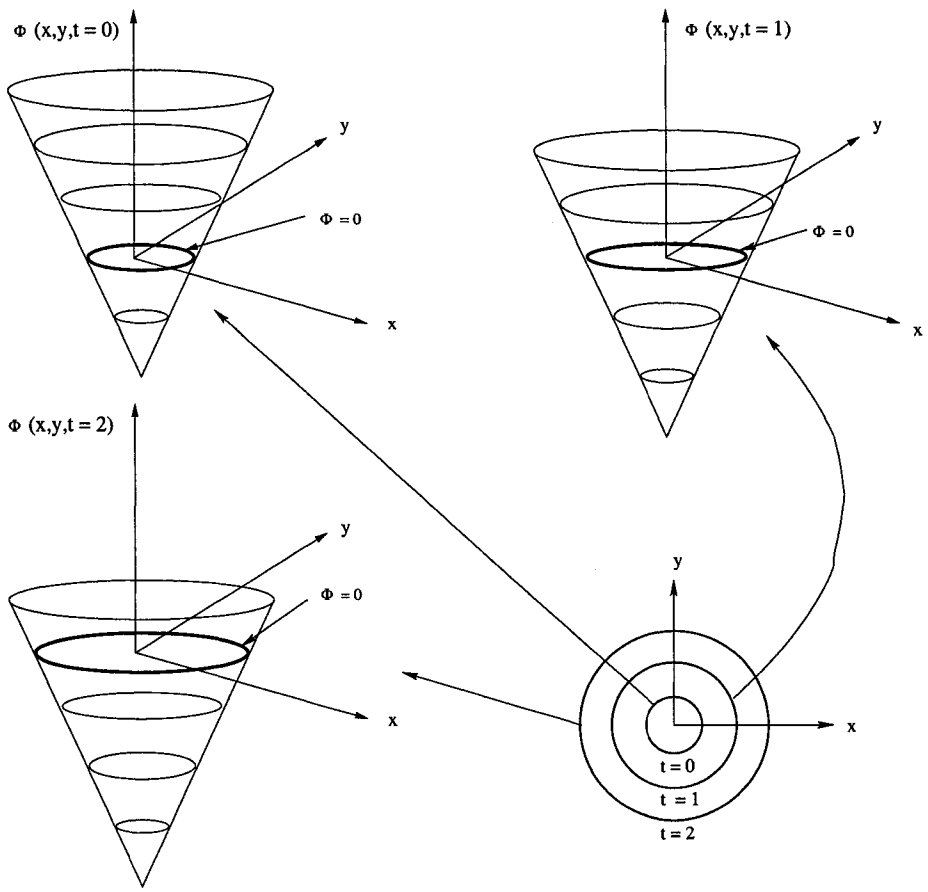

FIG. 3. *At any time t, the interface is defined by the zero level set ($\phi = 0$).*

properties and to accurately model three-dimensional geometry of forests in large-scale simulation of wildfire. This is a classic multiscale problem for which routine simulation, or simulation that can interact with real-time fire fighting, is not yet possible. In our application, we use semi-empirical firespread models derived from one originally proposed by Fendell and Wolff [16] that requires a minimal amount of information about properties of the environment around the firefront, corresponding to the rudiments that are often available. This model specifies the speeds of a wind-driven firefront at each point along its perimeter. Later, Mallet [25] proposed a simplified version of the Fendell-Wolff firespread model. These firespread models are not based on first-principles theory but contain parameters to be tuned by comparison with historical fire data. We refer to the simplified model as the Fendell-Mallet model.

Experience with explicit level set implementations of these firespread models shows numerical instability unless the timestep is carefully chosen and sufficiently small. Therefore, in this thesis, we develop an implicit solver for the level set equation, which is nonlinear. Newton's method is a leading candidate for solving a nonlinear system of equations of the discrete form $F(x) = 0$, where F is a vector-valued nonlinear function of state variables x. It can be applied on each time step of an implicit time discretization of a partial differential equation. It is attractive because of the asymptotically quadratic rate of convergence. However, computing a Newton step can be expensive when the size of the problem is large. Instead of exactly computing a Newton step, Dembo et. al. [11] introduce an inexact Newton method in which an iterative method is used to compute a Newton step approximately. Both Newton's method and inexact Newton methods require explicitly formation of a complete Jacobian matrix. For very large problems, it can be expensive to form and store a Jacobian matrix. However, we can use Jacobian-free Newton-Krylov methods [23], such as Newton-GMRES, to overcome the Jacobian complexity. The Newton-GMRES method requires only products between a Jacobian matrix and a vector. These can be approximated by using finite differences. Hence, the Newton-GMRES can be implemented without explicitly forming the Jacobian matrix.

This thesis is organized as follows. A background discussion of level set methods is given in Chapter 2. Chapter 3 presents some theory and numerical approximation schemes for the Hamilton-Jacobi equations. Chapter 4 discusses methods of solving large systems of nonlinear equations, focusing on the case of a sparse Jacobian commonly arising in partial differential equations. Firespread models are presented in

Chapter 5. Chapter 6 contains idealized numerical experiments including a circular expansion that allows error verification. Finally, conclusions and future plans are given in Chapter 7.

CHAPTER 2

LEVEL SET METHOD

In this chapter, we provide some background of the level set method. Most material covered in this chapter comes from [30] and [36].

To derive a formula for a moving interface, we start with a curve in two-dimensional space. This curve moves in a direction normal to itself with a speed F (see Figure 4). This speed function F may depend on local or global properties of the interface, or on features independent of the interface, itself.

Local properties are properties determined by local information such as curvature. Global properties are those that depend on the shape and position of the interface. Examples of independent properties are underlying flow or other features embedded in the medium.

2.1 BOUNDARY VALUE FORMULATION

We first consider that the speed F is nonnegative. Therefore, the front will always expand outward. If a function $T(x, y)$ is the arrival time of the front as it crosses a point (x, y), then we can use the relation $distance = rate \times time$ to get

$$1 = F\frac{dT}{dx}. \tag{1}$$

In higher dimensions, we have

$$|\nabla T|F = 1, T = 0 \text{ on } \Gamma, \tag{2}$$

where Γ is the initial location of the front. Equation (2) is the boundary value formulation of the level set method. We can see in equation (2) that we require no time step in the boundary value formulation. We can use a method called *Fast Marching* [36] to solve the equation (2).

2.2 INITIAL VALUE FORMULATION

We suppose now that the speed function F is neither strictly positive nor negative, so that the front can move forward or backward as it evolves. As the result, the front can pass a point $\vec{x}$ multiple times. Hence, the crossing time function $T(\vec{x})$ is not a

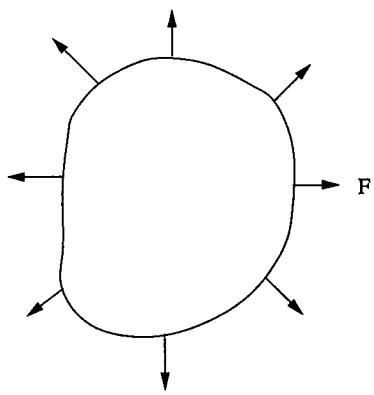

FIG. 4. *A curve moves in a direction normal to itself with a speed* $F > 0$.

single-valued function. To take care of multi-valued $T(\vec{x})$, we embed the initial position of the front as the zero level set of a function ϕ. In other words, at any time t, the position of the interface is defined as the locus of points $\vec{x}$ such that $\phi(\vec{x}, t) = 0$. Writing

$$\phi(\vec{x}(t), t) = 0, \tag{3}$$

the chain rule yields a partial differential equation for ϕ

$$\phi_t + \nabla\phi(\vec{x}(t), t) \cdot \vec{x}(t)' = 0. \tag{4}$$

Let $\vec{n}$ be a normal vector directed outward from the curve. Then $\vec{n}$ can be written in terms of $\phi(\vec{x}(t), t)$ as follows:

$$\vec{n} = \frac{\nabla\phi}{|\nabla\phi|}. \tag{5}$$

The speed function F, defined in the outward normal direction, is

$$F = \vec{x}(t)' \cdot \vec{n}. \tag{6}$$

Substituting equations (5) and (6) into equation (4), we obtain an evolution equation for ϕ,

$$\phi_t + F\,|\nabla\phi| = 0, \quad \text{given } \phi(x, t = 0). \tag{7}$$

This is the initial value formulation of the level set method.

2.3 ADVANTAGES OF THE LEVEL SET METHOD

There are several advantages of the level set method, as mentioned in the previous chapter.

1. Although the level set function remains well defined, the level set surface may change topology. The interface may merge or break. At any time t, the position of the interface is given by $\phi(x, y, t) = 0$ (in the initial value formulation) or by $T(x, y) = t$ (in the boundary value formulation).

2. Both initial value formulation and boundary value formulation generalize in their coordinate-invariant form to propagation in higher dimensions.

3. Intrinsic geometric properties of the interface are easy to evaluate. For example, the normal direction to the interface is given by

$$\vec{n} = \frac{\nabla\phi}{|\nabla\phi|} \text{ or } \vec{n} = \frac{\nabla T}{|\nabla T|}, \tag{8}$$

and, the curvature of the interface is given by

$$\kappa = \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \text{ or } \kappa = \nabla \cdot \frac{\nabla T}{|\nabla T|}. \tag{9}$$

4. Level set methods are easy to implement. They can be linked to hyperbolic conservation laws. Hence, we can use techniques borrowed from numerical solution of hyperbolic conservation laws.

2.4 NARROW BAND LEVEL SET METHOD

The initial value formulation of the level set method (7) can be solved in the entire computational domain. This approach is called a *full matrix approach* because all values of ϕ are updated. A more efficient approach to solving the initial value formulation (7) is to use only computational domain close to the zero level set. This approach was introduced by Chopp [6] and is called the *Narrow Band* level set method.

In the narrow band level set method, we place a narrow band around the initial front. A square array is used to store the entire two-dimensional grid of data. We use a one-dimensional array to keep track of the points in this band. Then, we update only the values of ϕ within this band. The calculation halts temporarily when the front gets too close to the edge of the band. Then, we rebuild a new band around the front and repeat the process.

CHAPTER 3

THEORY AND NUMERICAL APPROXIMATIONS

In this chapter, we provide some theoretical background and numerical approximation techniques for the Hamilton-Jacobi equation. Most material covered in this chapter comes from [14] and from [36, 38].

3.1 HAMILTON-JACOBI EQUATION

An initial-value problem for the Hamilton-Jacobi equation is given by

$$\begin{cases} u_t + H(Du) = 0 \text{ in } \mathbb{R}^n \times (0,\infty) \\ u = u_0 \text{ on } \mathbb{R}^n \times \{t = 0\} \end{cases} \tag{10}$$

where $u : \mathbb{R}^n \times [0,\infty) \to \mathbb{R}$ is unknown, $u = u(x,t)$ and $Du = D_x u = D(u_{x_1}, \cdots, u_{x_n})$. Here $H : \mathbb{R}^n \to \mathbb{R}$ is called the *Hamiltonian.* Both the Hamiltonian H and $u_0 : \mathbb{R}^n \to \mathbb{R}$ are given. If we wish to study the Hamilton-Jacobi equations, and recover corners or shocks, then non-smooth solutions u of the Hamilton-Jacobi equation, known as *weak solutions*, must be allowed. One approach for capturing these non-smooth solutions is to add an artificial viscosity term to the right-hand side of equation (10), that is,

$$\begin{cases} u_t^\epsilon + H^\epsilon(Du^\epsilon) = \epsilon \Delta u^\epsilon \text{ in } \mathbb{R}^n \times (0,\infty) \\ u^\epsilon = u_0^\epsilon \text{ on } \mathbb{R}^n \times \{t = 0\} \end{cases} \tag{11}$$

where ϵ is a small positive number. The limit of the solution u^ϵ of (11) as $\epsilon \to 0$ gives the physically correct weak solution when no classical solution exists.

Instead of defining the weak solution as a limit of smooth solution, Crandall and Lions define a weak solution in [9] as follows:

Definition 1 (viscosity solution) *A bounded, uniformly continuous function u is said to be a* viscosity solution *of the initial-value problem for the Hamilton-Jacobi equation (10) if for all smooth test functions v,*

(i) if $u - v$ has a local maximum at a point $(x_0, t_0) \in \mathbb{R}^n \times (0,\infty)$, then

$$v_t(x_0,t_0) + H(Dv(x_0,t_0), x_0) \leq 0, \tag{12}$$

(ii) if $u - v$ has a local minimum at a point $(x_0, t_0) \in \mathbb{R}^n \times (0,\infty)$, then

$$v_t(x_0,t_0) + H(Dv(x_0,t_0), x_0) \geq 0. \tag{13}$$

Now, we want to show that any smooth classical solution of the Hamilton-Jacobi equation is also a viscosity solution. The proof is straightforward. Let u be a solution of equation (10). Suppose u is bounded and uniformly continuous. If $v \in \mathbb{C}^\infty(\mathbb{R}^n \times (0,\infty))$ and $u - v$ has a local maximum at a point (x_0, t_0), then

$$\begin{cases} Du(x_0,t_0) = Dv(x_0,t_0), \\ u_t(x_0,t_0) = v_t(x_0,t_0). \end{cases} \tag{14}$$

Since u is the solution of (10), then

$$u_t(x_0,t_0) + H(Du(x_0,t_0),x_0) = 0. \tag{15}$$

Using equation (14) in (15), we have

$$v_t(x_0,t_0) + H(Dv(x_0,t_0),x_0) = 0. \tag{16}$$

Hence, (12) holds for a smooth function v. Similarly, if $u - v$ has a local maximum, then (13) also holds.

To show that viscosity solution provides an appropriate weak solutions for the initial value problem for the Hamilton-Jacobi equation, we only need to verify consistency and uniqueness of the viscosity solution.

Theorem 1 (Consistency of viscosity solutions) *(cf. [14]) Suppose a viscosity solution u of (10) is differentiable at some point $(x_0,t_0) \in \mathbb{R}^n \times (0,\infty)$. Then u satisfies the Hamilton-Jacobi equation at (x_0,t_0), that is,*

$$u_t(x_0,t_0) + H(Du(x_0,t_0),x_0) = 0. \tag{17}$$

Next we need to establish the uniqueness of a viscosity solution to the initial-value problem (10).

Theorem 2 (Uniqueness of viscosity solutions) *(cf. [14]) Suppose the Hamiltonian H is Lipschitz continuous, that is, $\forall x,y,p,q \in \mathbb{R}^n$, and some constant $C \geq 0$, we have*

$$\begin{cases} |H(p,x) - H(q,x)| \leq C\,|p-q| \\ |H(p,x) - H(p,y)| \leq C\,|x-y|(1+|p|). \end{cases} \tag{18}$$

Then there exists at most one viscosity solution of the initial-value problem for Hamilton-Jacobi (10).

3.2 HAMILTON-JACOBI EQUATION AND CONSERVATION LAWS

Consider the one-dimensional Hamilton-Jacobi equation

$$\phi_t + H(\phi_x) = 0. \tag{19}$$

By applying a spatial derivative to equation (19), we have

$$(\phi_x)_t + H(\phi_x)_x = 0. \tag{20}$$

Setting $u = \phi_x$, we obtain an equation for a scalar hyperbolic conservation law

$$u_t + [H(u)]_x = 0. \tag{21}$$

Hence, in one-dimensional space, we can connect Hamilton-Jacobi equations to hyperbolic conservation laws. In other words, the solution ϕ of a Hamilton-Jacobi equation can be obtained by integrating a solution u of a conservation law. Conversely, a solution u of a conservation law can be obtained differentiating a solution ϕ of a Hamilton-Jacobi equation.

Since conservation laws may have nonunique solutions, we use an *entropy condition* to pick out the physically correct solution. This is the vanishing viscosity solution discussed in the previous section.

The relationship between Hamilton-Jacobi equations and conservation laws provides some useful information. For example, solutions to Hamilton-Jacobi equations can develop kinks even from the smooth initial data. We need an entropy condition for Hamilton-Jacobi equations because conservation laws can have nonunique solutions. Finally, successful numerical methods for solving hyperbolic conservation laws can be inherited for Hamilton-Jacobi equations.

Although the relationship between Hamilton-Jacobi equations and conservation laws fails to hold in two or higher spatial dimensions, we can still perform numerical discretizations to a Hamilton-Jacobi equation in a dimension by dimension fashion. In [32], Osher and Shu use successful methods from the theory of conservation laws to propose a general framework for the numerical solution of Hamilton-Jacobi equation.

3.3 NUMERICAL APPROXIMATION

For simplicity, we consider numerical schemes for a two-dimensional Hamilton-Jacobi equation

$$\phi_t + H(\phi_x, \phi_y) = 0. \tag{22}$$

A forward Euler time discretization of (22) can be written as

$$\frac{\phi^{n+1}-\phi^n}{\Delta t}+\hat{H}(\phi_x^-,\phi_x^+;\phi_y^-,\phi_y^+)=0, \tag{23}$$

where $\hat{H}$ is a numerical approximation of $H(\phi_x,\phi_y)$. Here, the spatial derivative $\phi_x^{\pm}$ and $\phi_y^{\pm}$ can be either forward or backward differencing or the high-order accurate ENO (Essentially Non-Oscillatory) or WENO (Weighted Essentially Non-Oscillatory) scheme (see [19, 37]).

Definition 2 (Conservation form) *A numerical scheme to approximate the solution of (21) is said to be in* conservation form *if it can be written in the form*

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}=-\frac{\hat{H}_{i+1/2}-\hat{H}_{i-1/2}}{\Delta x}, \tag{24}$$

where $\hat{H}_{i+1/2}=\hat{H}(u_{i-p},\cdots,u_{i+q})$.

Here, the numerical flux functions $\hat{H}$ must be Lipschitz continuous and satisfy the consistency requirement $\hat{H}(u,\cdots,u)=H(u)$.

Definition 3 (Monotone) *A numerical scheme is called* monotone *if $\hat{H}$ is a non-decreasing function of all its arguments.*

It is known that numerical solutions of a conservative, monotone scheme satisfy the entropy condition (see [39] for a proof). Hence, if we want to construct a numerical scheme to approximate a solution of a Hamilton-Jacobi equation, we only need to check that it is in conservation form, and that it satisfies monotonicity.

3.3.1 Lax-Friedrichs Scheme

The Lax-Friedrichs (LF) scheme for the approximation of $\hat{H}$ from [10] is given by

$$\hat{H}^{LF}=H\Big(\frac{\phi_x^-+\phi_x^+}{2},\frac{\phi_y^-+\phi_y^+}{2}\Big)-\alpha^x\Big(\frac{\phi_x^+-\phi_x^-}{2}\Big)-\alpha^y\Big(\frac{\phi_y^+-\phi_y^-}{2}\Big), \tag{25}$$

where

$$\alpha^x=\max_{\substack{A\le\phi_x\le B\\ C\le\phi_y\le D}}|H_{\phi_x}(\phi_x,\phi_y)|,\quad \alpha^y=\max_{\substack{A\le\phi_x\le B\\ C\le\phi_y\le D}}|H_{\phi_y}(\phi_x,\phi_y)|. \tag{26}$$

Here H_{ϕ_x} and H_{ϕ_y} are partial derivatives with respect to ϕ_x and ϕ_y, respectively, and the computational domain is $[A,B]\times[C,D]$.

In the above scheme, the dissipation coefficients α^x and α^y are used to control the amount of numerical viscosity. The dissipation coefficients are set to be the maximum values of $|H_{\phi_x}|$ and $|H_{\phi_y}|$, respectively. However, it is sometimes difficult to evaluate the maximum values of $|H_{\phi_x}|$ and $|H_{\phi_y}|$. Osher and Fedkiw suggest in [30] estimates for the coefficients α^x and α^y. Since $|\phi_x|/|\nabla\phi| \leq 1$ for all ϕ_x and ϕ_y and $H_{\phi_x} = F\ |\phi_x|/|\nabla\phi|$ where F is the speed function, we have the bound $|H_{\phi_x}| \leq |F|$. Similarly, we have the bound $|H_{\phi_y}| \leq |F|$. Hence, both dissipation coefficients α^x and α^y can be set to the maximum value of $|F|$ on the Cartesian mesh.

The artificial viscosity coefficients should be chosen as small as possible because large values will increase the amount of artificial dissipation, degrading the quality of the solution. Hence, it makes little sense to choose the dissipation coefficient globally. Some regions of the grid may need a small value of the artificial coefficients while other regions may need a large value of α. These considerations lead to a *Stencil Lax-Friedrichs (SLF) scheme*, where we use only grid points close to the point $x_{i,j}$ to determine artificial dissipation [30].

3.3.2 Local Lax-Friedrichs Scheme

The Local Lax-Friedrichs (LLF) scheme [38] is introduced by Shu and Osher. Instead of looking at neighboring grid points, they use only the value of ϕ_x^- and ϕ_x^+ to determine an interval $I^x(\phi_x^-, \phi_x^+) = [\min(\phi_x^-, \phi_x^+),\ \max(\phi_x^-, \phi_x^+)]$ at each grid point. Then the dissipation coefficient α^x is chosen from the interval $I^x(\phi_x^-, \phi_x^+)$, while the dissipation coefficient α^y is still determined in the LF fashion. Similarly, α^y uses an interval $I^y(\phi_y^-, \phi_y^+)$, defined using only value of ϕ_y^- and ϕ_y^+, while α^x is determined in the LF fashion. Hence, the LLF scheme is given by

$$\begin{aligned} \hat{H}^{LLF} &= H\Big(\frac{\phi_x^- + \phi_x^+}{2}, \frac{\phi_y^- + \phi_y^+}{2}\Big) - \alpha^x(\phi_x^+, \phi_x^-)\Big(\frac{\phi_x^+ - \phi_x^-}{2}\Big) \\ &\quad -\alpha^y(\phi_y^+, \phi_y^-)\Big(\frac{\phi_y^+ - \phi_y^-}{2}\Big), \end{aligned} \tag{27}$$

where

$$\alpha^x(\phi_x^+, \phi_x^-) = \max_{\substack{\phi_x \in I^x(\phi_x^-, \phi_x^+) \\ C \leq \phi_y \leq D}} |H_{\phi_x}(\phi_x, \phi_y)|,\ \ \alpha^y(\phi_y^+, \phi_y^-) = \max_{\substack{A \leq \phi_x \leq B \\ \phi_y \in I^y(\phi_y^-, \phi_y^+)}} |H_{\phi_y}(\phi_x, \phi_y)|. \tag{28}$$

The Local Local Lax-Friedrichs (LLLF) scheme, proposed by Osher and Shu [32], has even smaller numerical dissipation than the LLF scheme. At any grid point, we

determine the interval $I^x(\phi_x^-, \phi_x^+)$ using the value of ϕ_x^- and ϕ_x^+ at that grid point. Similarly, we determine the interval $I^y(\phi_y^-, \phi_y^+)$ by using the value of ϕ_y^- and ϕ_y^+ at that grid point. Then we use the intervals $I^x(\phi_x^-, \phi_x^+)$ and $I^y(\phi_y^-, \phi_y^+)$ to determine both α^x and α^y. Hence, the LLLF scheme is given by

$$\begin{aligned} \hat{H}^{LLLF} &= H\Big(\frac{\phi_x^- + \phi_x^+}{2}, \frac{\phi_y^- + \phi_y^+}{2}\Big) - \alpha^x(\phi_x^\pm, \phi_y^\pm)\Big(\frac{\phi_x^+ - \phi_x^-}{2}\Big) \\ &\quad -\alpha^y(\phi_x^\pm, \phi_y^\pm)\Big(\frac{\phi_y^+ - \phi_y^-}{2}\Big), \end{aligned} \tag{29}$$

where

$$\alpha^x(\phi_x^\pm, \phi_y^\pm) = \max_{\substack{\phi_x \in I^x(\phi_x^-, \phi_x^+) \\ \phi_y \in I^y(\phi_y^-, \phi_y^+)}} |H_{\phi_x}(\phi_x, \phi_y)|, \quad \alpha^y(\phi_x^\pm, \phi_y^\pm) = \max_{\substack{\phi_x \in I^x(\phi_x^-, \phi_x^+) \\ \phi_y \in I^y(\phi_y^-, \phi_y^+)}} |H_{\phi_y}(\phi_x, \phi_y)|. \tag{30}$$

3.3.3 Godunov's Scheme

The Godunov scheme can be written as

$$\hat{H}^G = \operatorname*{ext}_{\phi_x \in I^x(\phi_x^-, \phi_x^+)} \operatorname*{ext}_{\phi_y \in I^y(\phi_y^-, \phi_y^+)} H(\phi_x, \phi_y), \tag{31}$$

where

$$\operatorname*{ext}_{\phi_x \in I^x(\phi_x^-, \phi_x^+)} = \begin{cases} \min\limits_{\phi_x^- \le \phi_x \le \phi_x^+} & \text{if } \phi_x^- \le \phi_x^+ \\ \max\limits_{\phi_x^+ \le \phi_x \le \phi_x^-} & \text{if } \phi_x^- > \phi_x^+ \end{cases}, \tag{32}$$

$$\operatorname*{ext}_{\phi_y \in I^y(\phi_y^-, \phi_y^+)} = \begin{cases} \min\limits_{\phi_y^- \le \phi_y \le \phi_y^+} & \text{if } \phi_y^- \le \phi_y^+ \\ \max\limits_{\phi_y^+ \le \phi_y \le \phi_y^-} & \text{if } \phi_y^- > \phi_y^+ \end{cases}. \tag{33}$$

Here, intervals $I^x(\phi_x^-, \phi_x^+)$ and $I^y(\phi_y^-, \phi_y^+)$ are defined as in the LLLF manner. Since $\operatorname*{ext}_{\phi_x \in I^x(\phi_x^-, \phi_x^+)} \operatorname*{ext}_{\phi_y \in I^y(\phi_y^-, \phi_y^+)} H(\phi_x, \phi_y) \neq \operatorname*{ext}_{\phi_y \in I^y(\phi_y^-, \phi_y^+)} \operatorname*{ext}_{\phi_x \in I^x(\phi_x^-, \phi_x^+)} H(\phi_x, \phi_y)$, we obtain different versions of Godunov's scheme by changing the order of min and max [30].

3.3.4 Roe-Fix Scheme

A Roe-Fix (RF) scheme, proposed by Shu and Osher [38], uses a Roe upwind method along with an LLF entropy correction. In this scheme, intervals I^x and I^y are determined as in an LLF scheme. We use the partial derivatives $H_{\phi_x}(\phi_x, \phi_y)$ and $H_{\phi_y}(\phi_x, \phi_y)$ to estimate the potential for upwinding. If $H_{\phi_x}(\phi_x, \phi_y)$ does not change

a sign in $\phi_x \in I^x$ and $C \leq \phi_y \leq D$, then we can apply upwinding. Hence, we do not need a dissipation coefficient α^x, and set $\alpha^x = 0$. Similarly, If $H_{\phi_y}(\phi_x, \phi_y)$ does not change sign in $A \leq \phi_x \leq B$ and all $\phi_y \in I^y$, then the dissipation coefficient α^y can be set to zero. When either H_{ϕ_x} or H_{ϕ_y} changes sign, we need an artificial dissipation to pick out the physically correct vanishing viscosity solution.

If $H_{\phi_x}(\phi_x, \phi_y) \geq 0$, then information travels along characteristics from left to right, and we use a backward difference for approximating ϕ_x. Otherwise, if $H_{\phi_x}(\phi_x, \phi_y) \leq 0$, we use a forward difference for approximating ϕ_x. Similarly, $H_{\phi_y}(\phi_x, \phi_y) \geq 0$ indicates using a backward difference for ϕ_y, and $H_{\phi_y}(\phi_x, \phi_y) \leq 0$ indicates using a forward difference for approximating ϕ_y. Hence, the RF scheme can be summarized as follows:

$$\hat{H}^{RF} = \begin{cases} H(\phi_x^*, \phi_y^*) \\ \quad \text{if both } H_{\phi_x}(\phi_x, \phi_y) \text{ and } H_{\phi_y}(\phi_x, \phi_y) \text{ have the same sign in} \\ \quad\quad \phi_x \in I^x,\ \phi_y \in I^y, \\ H\left(\frac{\phi_x^+ + \phi_x^-}{2}, \phi_y^*\right) - \alpha^x\left(\frac{\phi_x^+ - \phi_x^-}{2}\right) \\ \quad \text{if } H_{\phi_x}(\phi_x, \phi_y) \text{ changes sign but } H_{\phi_y}(\phi_x, \phi_y) \text{ has the same sign in} \\ \quad\quad A \leq \phi_x \leq B,\ \phi_y \in I^y, \\ H\left(\phi_x^*, \frac{\phi_y^+ + \phi_y^-}{2}\right) - \alpha^y\left(\frac{\phi_y^+ - \phi_y^-}{2}\right) \\ \quad \text{if } H_{\phi_x}(\phi_x, \phi_y) \text{ has the same sign but } H_{\phi_y}(\phi_x, \phi_y) \text{ changes sign in} \\ \quad\quad \phi_x \in I^x,\ C \leq \phi_y \leq D, \\ \hat{H}^{LLF} \quad \text{otherwise} \end{cases} \tag{34}$$

where

$$\phi_x^* = \begin{cases} \phi_x^+ & \text{if } H_{\phi_x} \leq 0 \\ \phi_x^- & \text{if } H_{\phi_x} \geq 0 \end{cases}, \quad \phi_y^* = \begin{cases} \phi_y^+ & \text{if } H_{\phi_y} \leq 0 \\ \phi_y^- & \text{if } H_{\phi_y} \geq 0 \end{cases}. \tag{35}$$

3.3.5 Upwind Scheme

It has been pointed out in [30] that truncation errors in a upwind scheme serve the same purpose as the artificial term $\epsilon\Delta\phi$. In [36], Sethian examines a one-dimensional wave equation with nonconstant speed and demonstrates the importance of upwind scheme in propagating interface. We give an outline of the study as follows.

A one-dimensional wave equation with a velocity $a(x)$ is given by

$$u_t(x,t) + a(x)u_x(x,t) = 0. \tag{36}$$

A forward Euler time discretization of the equation (36) can be written as

$$\frac{u^{n+1}-u^n}{\Delta t}+\Big[a(x)u_x\Big]^n=0, \tag{37}$$

where $u^n = u(t^n)$ represents the current values of u at time t^n. Next, we have to find an appropriate finite difference operator (forward, backward or central) for the spatial derivative u_x. Recall that these difference operators are given by

$$\begin{aligned} D^{+x}u_i^n &= \frac{u_{i+1}^n-u_i^n}{\Delta x} \quad \text{(forward)},\\ D^{-x}u_i^n &= \frac{u_i^n-u_{i-1}^n}{\Delta x} \quad \text{(backward)},\\ D^{0x}u_i^n &= \frac{u_{i+1}^n-u_{i-1}^n}{2\Delta x} \quad \text{(central)}, \end{aligned} \tag{38}$$

respectively. The method of characteristics applied to the equation (36) helps decide what numerical operator to use. Let the parameter s be a characteristic curve with the relations

$$\frac{dt}{ds}=1 \text{ and } \frac{dx}{ds}=a. \tag{39}$$

Then the chain rule and equation (36) yield

$$\frac{du}{ds}=\frac{du}{dx}\frac{dx}{ds}+\frac{du}{dt}\frac{dt}{ds}=\frac{dt}{ds}\Big(u_t+au_x\Big)=0. \tag{40}$$

Thus we can express equation (36) as the ordinary differential equation

$$\frac{du}{ds}=0. \tag{41}$$

Hence, given the value of u at some arbitrary point (x_0, y_0), we can determine the coordinates of the characteristic curve passing through (x_0, y_0) by integrating equations (39). A solution along this characteristic can be obtained by integrating equation (41). Using equations (39), we obtain $\frac{dx}{dt}=a$ which the solution is $x=at+c$. Therefore, the direction of the propagating wave depends on the sign of the propagation speed a. If the speed a is positive, then information travels from left to right. Hence, the backward operator is desired that uses information at node i and $i-1$ to compute a new value of u at node i. By physical analogy, the backward operator is often referred as an upwind operator. Similarly, if the propagation speed a is negative, then information travels from right to left. Hence, we prefer the forward operator in

this case. As the result, Osher and Sethian [31] construct a numerical scheme to suit the sign of the propagating speed a as follows:

$$u_i^{n+1} = u_i^n - \Delta t\Big[\max(0, a_i)D^{-x}u_i^n + \min(0, a_i)D^{+x}u_i^n\Big]. \tag{42}$$

We can see from the above equation that the backward operator is selected when the propagation speed a is positive, and the forward operator is selected if the propagating speed a is negative.

In [31], Osher and Sethian use the idea above to propose a numerical approximation of $|\nabla\phi|$ where

$$|\nabla\phi| = \sqrt{\phi_x^2 + \phi_y^2}. \tag{43}$$

If the speed function $F > 0$ then the approximation of $|\nabla\phi|$ is given by

$$\begin{aligned}\nabla^+ &= \Big(\max(D^{-x}\phi_{ij}, 0)^2 + \min(D^{+x}\phi_{ij}, 0)^2 \\ &\quad + \max(D^{-y}\phi_{ij}, 0)^2 + \min(D^{+y}\phi_{ij}, 0)^2\Big)^{\frac{1}{2}}.\end{aligned} \tag{44}$$

If the speed function $F < 0$ then the approximation of $|\nabla\phi|$ is given by

$$\begin{aligned}\nabla^- &= \Big(\min(D^{-x}\phi_{ij}, 0)^2 + \max(D^{+x}\phi_{ij}, 0)^2 \\ &\quad + \min(D^{-y}\phi_{ij}, 0)^2 + \max(D^{+y}\phi_{ij}, 0)^2\Big)^{\frac{1}{2}}.\end{aligned} \tag{45}$$

However, if we do not know the sign of the speed function F ahead of time then the product between the speed function F and $|\nabla\phi|$ is approximated by

$$F|\nabla\phi| \approx \left(\max(F_{ij}, 0)\nabla^+ + \min(F_{ij}, 0)\nabla^-\right). \tag{46}$$

Finally, the implicit numerical scheme of a level set equation used in this thesis is given by

$$\phi_{ij}^{n+1} - \phi_{ij}^n + \Delta t\left(\max\left(F_{ij}^{n+1}, 0\right)\nabla^{(n+1)+} + \min\left(F_{ij}^{n+1}, 0\right)\nabla^{(n+1)-}\right) = 0, \tag{47}$$

where ϕ_{ij}^n is the value of ϕ at point (x_i, y_j) at time n.

If an explicit numerical scheme is used, then we need to enforce numerical stability by using the Courant-Friedrichs-Lewy (CFL) condition first discussed in [8]. Since the propagation speed of numerical waves should be at least as fast as the speed of the physical waves, we have $\Delta x/\Delta t > |F|$. Hence, the CFL condition is

$$\Delta t < \frac{\Delta x}{\max\{|F|\}}, \tag{48}$$

where $\max\{|F|\}$ is chosen to be the largest value of $|F|$ over the entire computational domain. In practice, we can enforce equation (48) by choosing a CFL number ν with

$$\nu = \Delta t\left(\frac{\max\{|F|\}}{\Delta x}\right) \tag{49}$$

and $0 < \nu < 1$. A common choice of ν is 0.5. In two dimensions, a CFL condition can be written as

$$\nu = \Delta t\left(\frac{\max\{|F|\}}{\min\{\Delta x, \Delta y\}}\right). \tag{50}$$

CHAPTER 4

SOLVING THE NONLINEAR EQUATION

In this chapter, we discuss methods of solving nonlinear equations. Most material covered in this chapter comes from [20], [21], [22] and [27]. Since we need to solve a nonlinear system on each implicitly differenced time step, and have a good initial guess from the previous time step, we employ a version of Newton's method, which offers up to quadratic convergence. In practice, we adopt an economical inexact Newton's method. Newton's method requires at each iteration the solution of a linear system involving a Jacobian matrix. For this purpose, we employ a Krylov method tailored to the properties of the Jacobian and the inexact requirements for the solution of the linear system. In introducing these methods, we quote some well-known convergence results that motivate our choices.

4.1 RATE OF CONVERGENCE

The performance of an algorithm can be measured, in part, by its rate of convergence. Let $\{x_k\}$ be a sequence of real numbers that converges to x^*. We say that the rate of convergence is *linear* if there is a constant $c \in (0,1)$ such that

$$||x_{k+1} - x^*|| \leq c\,||x_k - x^*||, \quad \text{for all } k \text{ sufficiently large.} \tag{51}$$

We say that the rate of convergence is *superlinear* if

$$\lim_{k\to\infty} \frac{||x_{k+1} - x^*||}{||x_k - x^*||} = 0. \tag{52}$$

We say that the rate of convergence is *quadratic* if there is a positive constant M (not necessarily less than 1) such that

$$||x_{k+1} - x^*|| \leq M\,||x_k - x^*||^2, \quad \text{for all } k \text{ sufficiently large.} \tag{53}$$

In general, we say that the rate of convergence is at least *order* α if there is a positive constant M such that

$$||x_{k+1} - x^*|| \leq M\,||x_k - x^*||^\alpha, \quad \text{for all } k \text{ sufficiently large.} \tag{54}$$

4.2 NEWTON'S METHOD

Newton's method is an iterative scheme for solving a nonlinear system. It is attractive because the method converges quadratically from any initial guess sufficiently close to the root. We first discuss the Newton's method in one dimension because diagrams readily illustrate its strengths and weaknesses. Then, we discuss the Newton's method in higher dimensions.

4.2.1 One-Dimensional Version of Newton's Method

We want to find solutions of

$$f(x) = 0. \tag{55}$$

Let x^* be a zero of f and let x be an approximation to x^*. If f'' exists and is continuous, then by Taylor's theorem

$$0 = f(x^*) = f(x+h) = f(x) + h\,f'(x) + \mathcal{O}(h^2), \tag{56}$$

where $h = x^* - x$. If x^* is close to x, then h will be small. Thus, we can ignore the $\mathcal{O}(h^2)$ term and then solve for h. We have $h = -f(x)/f'(x)$. If x is an approximation to x^* then $x - f(x)/f'(x)$ should be a better approximation to x^*. Newton's method starts with an initial estimate x_0 of x^* and then defines inductively

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad (k \geq 0). \tag{57}$$

A geometric interpretation of a Newton's method in one dimension is simple. At each step, we approximate a function $f(x)$ by its tangent at point $(x_k, f(x_k))$, and find where the tangent line intersects the x-axis (see Figure 5).

The Newton's method works well when the initial guess x_0 is close to the solution x^*. However, the method has difficulties when the initial guess x_0 is far from the solution x^*, or when $f'(x)$ is close to zero (see Figure 6). For problems in which a good initial guess is lacking, including many practical problems that are not time-stepped, Newton's method requires globalization strategies, whether based on heuristics from the application or general-purpose mathematical approaches.

4.2.2 Higher Dimensional Version of Newton's Method

The system of n equation in n unknowns

$$f_i(x_1, x_2, \ldots, x_n) = 0, \quad 1 \leq i \leq n, \tag{58}$$

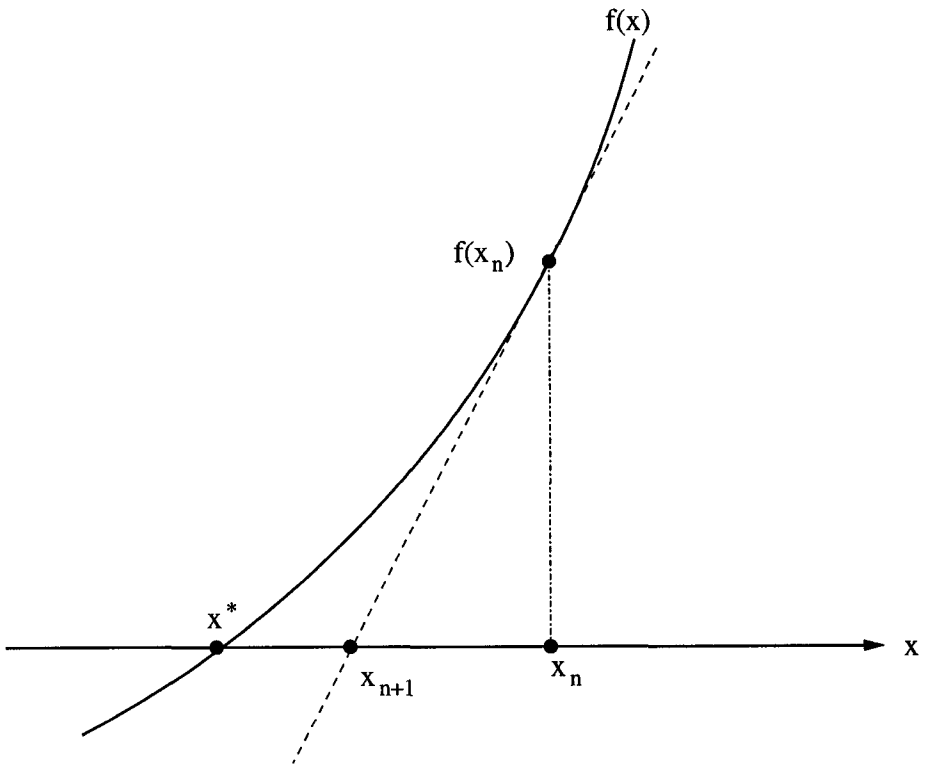

FIG. 5. *Newton's method in one dimension.*

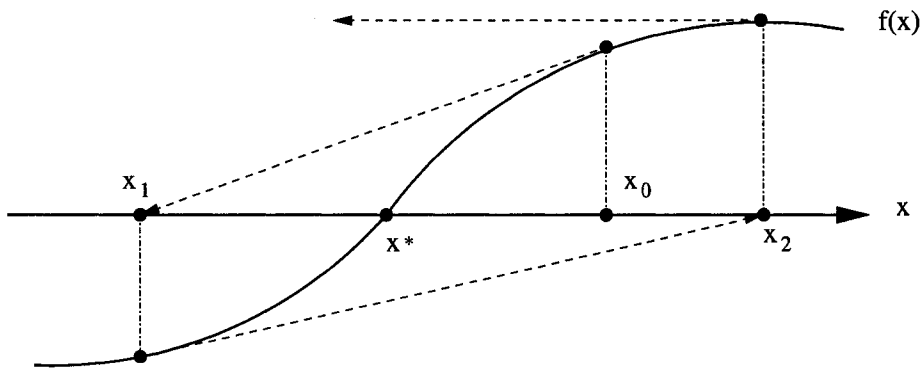

FIG. 6. *Example of divergence of Newton's method.*

can be expressed as

$$F(x) = 0 \tag{59}$$

by letting $x = (x_1, x_2, \ldots, x_n)^T$ and $F = (f_1, f_2, \ldots, f_n)^T$. Similar to the one-dimensional case, Taylor's theorem gives

$$0 = F(x + s) \approx F(x) + F'(x)\, s, \tag{60}$$

where $s = (h_1, h_2, \ldots, h_n)^T$ and $F'(x)$ is the Jacobian matrix $J(x)$ with elements $\partial f_i / \partial x_j$; namely,

$$J(x) = \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \cdots & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 & \cdots & \partial f_2/\partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1 & \partial f_n/\partial x_2 & \cdots & \partial f_n/\partial x_n \end{pmatrix}. \tag{61}$$

We obtain the correction vector s by solving the system of equations (60). Thus,

$$s = -J(x)^{-1} F(x). \tag{62}$$

Hence, Newton's method for n nonlinear equations in n unknowns variable is given by

$$x_{k+1} = x_k + s_k, \tag{63}$$

where the Jacobian system is

$$J(x_k) s_k = -F(x_k). \tag{64}$$

An algorithm for computing a Newton iteration is shown in Figure 7.

Solving for a Newton step in the equation (64) may require evaluation and factorization of the Jacobian matrix, which can be very expensive. However, we can avoid explicit computation of the Jacobian matrix by using finite difference approximations of Jacobian-vector products in conjunction with a solver that requires only such products, and not the construction and storage of a full Jacobian, such as a Krylov method.

We note that each element of a Jacobian matrix J can be approximated by

$$J_{ij} \approx \frac{F_i(x + \epsilon e_j) - F_i(x)}{\epsilon}, \tag{65}$$

where $\epsilon > 0$ is some small number and e_j is a vector that has all zeros and the value 1 at the jth location.

Algorithm

1 *Choose an initial guess* x_0
2 *For* $k = 0, 1, \ldots$
3 *Find a step* s_k *from the Newton equations* $J(x_k)\, s_k = -F(x_k)$
4 *Update* $x_{k+1} = x_k + s_k$
5 *End*

FIG. 7. *Newton's method.*

4.2.3 Local Convergence Theory

We recall the fundamental theorem of calculus:

Theorem 3 *(cf.[20]) Suppose F is differentiable in an open set $\Omega \subset \mathbb{R}^n$ and let $x^* \in \Omega$. Then for all $x \in \Omega$ sufficiently near x^*, we have*

$$F(x) - F(x^*) = \int_0^1 J(x^* + t(x - x^*))(x - x^*)dt, \tag{66}$$

where $J(x) = F'(x)$.

Throughout this chapter, we let x^* be a solution of F, and we make assumptions on F as follows [20]:

Assumption 1 (The Standard Assumptions)

1. Equation (59) has a solution x^*.

2. $J(x)$ is Lipschitz continuous near x^*; that is, there exists a positive number γ such that

$$||J(x) - J(y)|| \leq \gamma ||x - y||, \tag{67}$$

for all x and y sufficiently close to x^*.

3. $J(x^*)$ is nonsingular.

We denote the ball of radius δ about x^* as

$$\mathcal{B}(\delta) = \{x \,|\, ||x - x^*|| < \delta\}. \tag{68}$$

We can use the standard assumptions to prove the following Lemma:

Lemma 1 *(cf.[20]) Suppose the standard assumptions hold. Then there is a $\delta > 0$ such that for all $x \in \mathcal{B}(\delta)$*

(i). $||J(x)|| \leq 2||J(x^*)||$,

(ii). $||J(x)^{-1}|| \leq 2||J(x^*)^{-1}||$,

(iii). $||x - x^*||/(2||J(x^*)^{-1}||) \leq ||F(x)|| \leq 2||J(x^*)||\,||x - x^*||$.

Then, the convergence result for Newton's method is given in the following theorem.

Theorem 4 *Suppose the standard assumptions hold and the initial guess x_0 is sufficiently close to x^*. Then the Newton iteration $x_{k+1} = x_k - J(x_k)^{-1}F(x_k)$ converges quadratically to x^*.*

Proof. (cf. [20]) Let $\delta > 0$ and $x_k \in \mathcal{B}(\delta)$, $k \geq 0$, be the Newton iterate. Since x^* is a solution of F and by Theorem 3,

$$F(x_k) = \int_0^1 J(x^* + t(x_k - x^*))(x_k - x^*)\, dt. \tag{69}$$

Let $e_k = x_k - x^*$. Then by Newton iteration and the equation (69), we have

$$\begin{aligned} e_{k+1} &= e_k - J(x_k)^{-1}F(x_k) \\ &= J(x_k)^{-1}J(x_k)e_k - J(x_k)^{-1}\int_0^1 J(x^* + te_k)e_k\, dt \\ &= J(x_k)^{-1}\int_0^1 \big(J(x_k) - J(x^* + te_k)\big)e_k\, dt. \end{aligned} \tag{70}$$

Since J is Lipschitz continuous and by Lemma 1,

$$\begin{aligned} ||e_{k+1}|| &\leq ||J(x_k)^{-1}||\, ||\int_0^1 \big(J(x_k) - J(x^* + te_k)\big)e_k\, dt|| \\ &\leq ||J(x_k)^{-1}|| \int_0^1 ||J(x_k) - J(x^* + te_k)||\, ||e_k||\, dt \\ &\leq \Big(2||J(x^*)^{-1}||\Big)\Big(\gamma||e_k||^2/2\Big) \\ &= M||e_k||^2, \end{aligned} \tag{71}$$

where $M = \gamma||J(x^*)^{-1}||$. We reduce δ so that $M\delta = \zeta < 1$. Since $x_k \in \mathcal{B}(\delta)$, we have

$$||e_{k+1}|| \leq M||e_k||^2 < \zeta||e_k|| < ||e_k||. \tag{72}$$

Hence, $x_{k+1} \in \mathcal{B}(\zeta\delta) \subset \mathcal{B}(\delta)$. By assumption, we have $x_0 \in \mathcal{B}(\delta)$. Therefore, the entire sequence $\{x_k\} \subset \mathcal{B}(\delta)$. Thus, the inequality (72) implies that x_k converges quadratically to x^*. □

4.3 INEXACT NEWTON METHOD

Since computing the Newton step can be expensive, we can solve the equation (64) approximately. Dembo et.al. [11] introduce an inexact Newton method in which the Newton step satisfies the following condition

$$||J(x_k)s_k + F(x_k)|| \leq \eta_k||F(x_k)||, \tag{73}$$

where η_k is a forcing term and s_k is a Newton step. If we choose a small value of η_k, then the iteration will more closely approximate Newton's method. However, a

small value of η_k may make the computation of a Newton step that satisfies (73) very expensive.

In [12], Eisenstat and Walker propose two choices of the forcing term in an inexact Newton method. The first choice is given by

$$\eta_k = \frac{\left| ||F(x_k)|| - ||F(x_{k-1}) + J(x_{k-1})s_{k-1}|| \right|}{||F(x_{k-1})||}, \quad k = 1, 2, \ldots, \tag{74}$$

for any given $\eta_0 \in [0, 1)$. Another choice of the forcing term is given by

$$\eta_k = \gamma \left(\frac{||F(x_k)||}{||F(x_{k-1})||} \right)^{\alpha}, \quad k = 1, 2, \ldots, \tag{75}$$

where $\gamma \in [0, 1], \alpha \in (1, 2]$ and $\eta_0 \in [0, 1)$ are given.

When we use iterative methods for solving the equation for the Newton step, we typically use (73) as a stopping criterion. Therefore, we usually call the overall nonlinear solver a *Newton iterative method.* The inexact Newton algorithm is shown in Figure 8.

In line 5 of the inexact Newton algorithm (see Figure 8), we can use a linear iterative method to solve for a Newton step s_k. This linear iteration is then referred as an *inner iteration* while the overall (nonlinear) iteration is called an *outer iteration.*

Theorem 5 (Inexact Newton method) *Suppose the standard assumptions hold. Then, there exists $\delta > 0$ and $\bar{\eta} > 0$ such that if $||x_0 - x^*|| \leq \delta$, $\{\eta_k\} \subset [0, \bar{\eta}]$, then the sequence of inexact Newton iterates $x_{k+1} = x_k + s_k$ where $||J(x_k)s_k + F(x_k)|| \leq \eta_k ||F(x_k)||$ converges linearly to x^*. Moreover, if $\eta_k \to 0$ then x_k converges superlinearly to x^*.*

Proof. (cf. [20]) Let $\delta > 0$ and $x_k \in \mathcal{B}(\delta)$, $k \geq 0$ be the Newton iterate. If

$$r = -J(x_k)s_k - F(x_k) \tag{76}$$

is the linear residual, then

$$\begin{aligned} J(x_k)^{-1}r &= -J(x_k)^{-1}J(x_k)s_k - J(x_k)^{-1}F(x_k) \\ &= -s_k - J(x_k)^{-1}F(x_k). \end{aligned} \tag{77}$$

Hence, we have

$$s_k = -J(x_k)^{-1}r - J(x_k)^{-1}F(x_k). \tag{78}$$

Algorithm

1 *Given* η
2 *Choose an initial guess* x_0
3 *For* $k = 0, 1, \ldots$
4 *Choose forcing term* $\eta_k \in [0, \eta]$
5 *Find* s_k *that satisfies* $||J(x_k)s_k + F(x_k)|| \leq \eta_k ||F(x_k)||$
6 *Update* $x_{k+1} = x_k + s_k$
7 *End*

FIG. 8. *Inexact Newton algorithm.*

Since $e_{k+1} = e_k + s_k$ and by equation (78), we have

$$e_{k+1} = e_k - J(x_k)^{-1}r - J(x_k)^{-1}F(x_k). \tag{79}$$

Using equation (78), the condition (73) and Lemma 1, we have

$$\begin{aligned} ||s_k + J(x_k)^{-1}F(x_k)|| &\leq ||J(x_k)^{-1}||\,||r|| = ||J(x_k)^{-1}||\,||J(x_k)s_k + F(x_k)|| \\ &\leq \left(2\,||J(x^*)^{-1}||\right)\left(\eta_k||F(x_k)||\right) \\ &\leq 2\,||J(x^*)^{-1}|| \cdot 2\,\eta_k\,||J(x^*)||\,||x_k - x^*|| \\ &= 4\,\eta_k\,\kappa(J(x^*))\,||e_k||, \end{aligned} \tag{80}$$

where $\kappa(J(x^*)) = ||J(x^*)||\,||J(x^*)^{-1}||$ is a condition number. Using inequality (71) and equation (79), we have

$$\begin{aligned} ||e_{k+1}|| &= ||e_k - J(x_k)^{-1}r - J(x_k)^{-1}F(x_k)|| \\ &\leq ||e_k - J(x_k)^{-1}F(x_k)|| + 4\,\eta_k\,\kappa(J(x^*))\,||e_k|| \\ &\leq M||e_k||^2 + 4\,\eta_k\,\kappa(J(x^*))\,||e_k|| \\ &\leq M_I\left(||e_k|| + \eta_k\right)||e_k||, \end{aligned} \tag{81}$$

where $M_I = M + 4\,\kappa(J(x^*))$.

Now, we reduce δ and $\bar{\eta}$ if needed so that

$$M_I(\delta + \bar{\eta}) < 1. \tag{82}$$

Since $x_k \in \mathcal{B}(\delta)$ and with inequality (81), we have

$$||e_{k+1}|| \leq M_I\left(||e_k|| + \eta_k\right)||e_k|| \leq M_I(\delta + \bar{\eta})||e_k|| < ||e_k||. \tag{83}$$

This prove that x_k converges linearly to x^*.

If $\eta_k \to 0$ then superlinear convergence is obtained by the definition. □

4.4 NEWTON-GMRES

Newton-GMRES uses the GMRES (Generalized Minimal RESidual) method as a linear solver. GMRES was proposed by Saad and Schultz in [35] as an iterative method for approximating the solution of a nonsymmetric linear system. In our problem, the nonlinear system is $J(x)s = b$ where $J(x)$ is a Jacobian matrix, s is a Newton step, and $b = -F(x)$. We refer the interested reader to [3, 18, 20, 33, 40] for more details and background concerning GMRES, and to [41] for some detail about iterative Krylov methods.

4.4.1 GMRES

At every step k of GMRES, the norm of the residual vector $||r_k|| = ||b - Js_k||$ is minimized over a Krylov subspace

$$\mathcal{K}_k \equiv span\{r_0, Jr_0, \ldots, J^{k-1}r_0\}. \tag{84}$$

Hence, we have the least squares problem

$$\min_{s \in s_0 + \mathcal{K}_k} ||b - Js||. \tag{85}$$

Let V_k be an orthogonal projector onto $\mathcal{K}_k$. Then any $z \in \mathcal{K}_k$ can be written in the form $z = V_k y$. Hence, the least squares problem (85) in $\mathcal{K}_k$ can be converted to a least squares problem in $\mathbb{R}^k$ for the coefficient vector y of $z = s - s_0$. Since $z = s - s_0$ and $z = V_k y$, we have $s - s_0 = V_k y$ for some $y \in \mathbb{R}^k$. Hence, we must have $s_k = s_0 + V_k y$ where y minimizes

$$||b - Js_k|| = ||b - J(s_0 + V_k y)|| = ||b - Js_0 - JV_k y|| = ||r_0 - JV_k y||.$$

Hence, the least squares problem in the reduced space $\mathbb{R}^k$ is

$$\min_{y \in \mathbb{R}^k} ||r_0 - JV_k y||. \tag{86}$$

In the GMRES method, Arnoldi's method is used to form an orthonormal basis for the Krylov subspace $\mathcal{K}_k$, which is stored in the columns of a matrix V_k. The Arnoldi algorithm (cf. [33]) is shown in Figure 9.

The Arnoldi algorithm uses Gram-Schmidt orthogonalization to form an orthonormal basis for $\mathcal{K}_k$. After k steps, it produces a $(k+1) \times k$ upper Hessenberg matrix $\tilde{H}_k$, whose entries h_{ij} satisfy $h_{ij} = 0$ if $i > j + 1$, and an $n \times k$ matrix V_k whose columns form an orthonormal basis $\{v_1, \ldots, v_k\}$. It follows from lines 5, 6, and 8 of Algorithm 4.4.1 (Figure 9) that (see also [33, 34, 35])

$$JV_k = V_k H_k + h_{k+1,k}\, v_{k+1} e_k^T, \tag{87}$$

where H_k is the $k \times k$ Hessenberg matrix formed by dropping the bottom row of $\tilde{H}_k$, which, in turn, shows up as the last term in (87), where $e_k = (0, \ldots, 0, 1)^T \in \mathbb{R}^k$. This relation can also be written as

$$JV_k = V_{k+1} \tilde{H}_k. \tag{88}$$

Algorithm

1 *Given an arbitary vector* r_0
2 *Define* $v_1 = r_0/||r_0||$
3 *For* $j = 1, 2, \ldots, k$ *Do*
4 *Compute* $h_{ij} = (Jv_j, v_i)$ *for* $i = 1, 2, \ldots, j$
5 *Compute* $w_{j+1} = Jv_j - \sum_{i=1}^{j} h_{ij} v_i$
6 $h_{j+1,j} = ||w_{j+1}||$
7 *If* $h_{j+1,j} = 0$ *then Stop*
8 $v_{j+1} = w_{j+1}/||w_{j+1}||$
9 *EndDo*

FIG. 9. *Arnoldi algorithm.*

For notational convenience, we set $\beta = ||r_0||$ and let $e_1 = (1, 0, \ldots, 0)^T \in \mathbb{R}^k$. Using $r_0 = \beta v_1$, the relation (88), and the orthonormality of V_{k+1}, we have, for some $y_k \in \mathbb{R}^k$

$$\begin{aligned} ||r_k|| &= ||r_0 - J(s_k - s_0)|| = ||\beta v_1 - JV_k y_k|| \\ &= ||\beta V_{k+1} e_1 - V_{k+1} \tilde{H}_k y_k|| = ||V_{k+1}(\beta e_1 - \tilde{H}_k y_k)|| \\ &= ||\beta e_1 - \tilde{H}_k y_k||. \end{aligned} \tag{89}$$

Hence, the solution of the least squares problem (85) is given by

$$s_k = s_0 + V_k y_k, \tag{90}$$

where y_k minimizes (89) over $y_k \in \mathbb{R}^k$.

Introducing a relative tolerance $\epsilon > 0$ for the solution of (85), we want to find a vector s such that

$$||b - Js|| \leq \epsilon ||b||. \tag{91}$$

Equation (91) is the stopping criterion for the method. The GMRES algorithm is shown in Figure 10.

The GMRES method uses orthogonality of the Krylov basis to estimate the residual. However, we may loose orthogonality because of floating point rounding errors. As a result, the iteration could terminate prematurely and the approximate solution could be inaccurate. A standard remedy to preserve orthogonality of the basis in floating point arithmetic is to replace the classical Gram-Schmidt orthogonalization with modified Gram-Schmidt orthogonalization (Figure 11). However, as a practical matter in large-scale parallel computation, modified Gram-Schmidt has the disadvantage of requiring frequent inner products, which impose synchronizations that can reduce parallel performance, so standard Gram-Schmidt is often used despite its poorer numerical pedigree, and loss of orthogonality carefully monitored.

In the GMRES method, the basis for the Krylov subspace must be stored. For large problems, this may become burdensome as the iteration progress. In practice, the size of the Krylov subspace must often be limited. This can be imposed by restarting the iteration when a storage space for a basis vector is exhausted. The restarted version of GMRES algorithm is called *GMRES(m)* (see Figure 12). However, strong convergence results for GMRES(m) are lacking and restarting will slow down the convergence.

Algorithm

1 *Compute* $r_0 = b - Js_0$, $\beta = ||r_0||$ *and* $v_1 = r_0/||r_0||$
2 *For* $j = 1, 2, \ldots, k$ *Do*
3 $h_{ij} = (Jv_j, v_i)$ *for* $i = 1, 2, \ldots, j$
4 $w_{j+1} = Jv_j - \sum_{i=1}^{j} h_{ij} v_i$
5 $h_{j+1,j} = ||w_{j+1}||$
6 *If* $h_{j+1,j} = 0$ *set* $k = j$ *and go to 9*
7 $v_{j+1} = w_{j+1}/||w_{j+1}||$
8 *EndDo*
9 *Compute* y_k *the minimizer of* $||\beta e_1 - \tilde{H}_k y_k||$ *and* $s_k = s_0 + V_k y_k$
10 *End*

FIG. 10. *GMRES algorithm.*

Algorithm

1 *Compute* $r_0 = b - Js_0$, $\beta = ||r_0||$ *and* $v_1 = r_0/||r_0||$
2 *For* $j = 1, 2, \ldots, k$ *Do*
3 *Compute* $w_{j+1} = Jv_j$
4 *For* $i = 1, \ldots, j$ *Do*
5 $h_{ij} = (w_{j+1}, v_i)$
6 $w_{j+1} = w_{j+1} - h_{ij}v_i$
7 *EndDo*
8 $h_{j+1,j} = ||w_{j+1}||$
9 *If* $h_{j+1,j} = 0$ *set* $k = j$ *and go to 12*
10 $v_{j+1} = w_{j+1}/||w_{j+1}||$
11 *EndDo*
12 *Compute* y_k, *the minimizer of* $||\beta e_1 - \tilde{H}_k y_k||$, *and set* $s_k = s_0 + V_k y_k$
13 *End*

FIG. 11. *GMRES algorithm using modified Gram-Schmidt process.*

Algorithm

1 *Choose* ϵ
2 *Compute* $r_0 = b - Js_0$, $\beta = ||r_0||$ *and* $v_1 = r_0/||r_0||$
3 *For* $j = 1, 2, \ldots, m$ *Do*
4 *Compute* $w_{j+1} = Jv_j$
5 *For* $i = 1, \ldots, j$ *Do*
6 $h_{ij} = (w_{j+1}, v_i)$
7 $w_{j+1} = w_{j+1} - h_{ij}v_i$
8 *EndDo*
9 $h_{j+1,j} = ||w_{j+1}||$
10 *If* $h_{j+1,j} = 0$ *set* $k = j$ *and go to 13*
11 $v_{j+1} = w_{j+1}/||w_{j+1}||$
12 *EndDo*
13 *Form* $s_m = s_0 + V_m y_m$, *where* y_m *minimizes* $||\beta e_1 - \tilde{H}_m y||$
14 *Compute* $r_m = b - Js_m$
15 *If* $||r_m|| \leq \epsilon||b||$ *then*
16 *stop*
17 *else*
18 *set* $s_0 = s_m$, $v_1 = r_m/||r_m||$ *and go to line 2*

FIG. 12. *GMRES(m) algorithm.*

4.4.2 Convergence of GMRES

The kth GMRES residual in $\mathcal{K}_k$ can be written as a polynomial in J multiplied by the residual b as follows:

We define the set of kth degree monic polynomial $\mathcal{P}_k$ as

$$\mathcal{P}_k = \{p \,|\, p \text{ is a polynomial of } k \text{ degree with } p(0) = 1\}. \tag{92}$$

If $x \in \mathcal{K}_k$, then x can be written as a linear combination of powers of J times b, i.e.,

$$x = c_0 b + c_1 Jb + c_2 J^2 b + \ldots + c_{k-1} J^{k-1} b. \tag{93}$$

Defining $q \in \mathcal{P}_{k-1}$ as the polynomial

$$q(z) = c_0 + c_1 z + c_2 z^2 + \ldots + c_{k-1} z^{k-1}, \tag{94}$$

equation (93) can be rewritten as

$$x = q(J)b. \tag{95}$$

Then

$$r_k = b - Js_k = (I - Jq_k(J))b. \tag{96}$$

Define a polynomial $p_k \in \mathcal{P}_k$ by

$$p_k(z) = 1 - zq(z). \tag{97}$$

The equations (96) and (97) give

$$r_k = p_k(J)b. \tag{98}$$

Then

$$||r_k|| = ||p_k(J)b|| \leq ||p_k(J)||\,||b||. \tag{99}$$

Hence, we have

$$\frac{||r_k||}{||b||} \leq \min_{p_k \in \mathcal{P}_k} ||p_k(J)||. \tag{100}$$

Inequality (100) can be used to prove finite termination of the GMRES algorithm [20].

Theorem 6 *Suppose J is a nonsingular matrix. Then the GMRES algorithm will find the solution of $Js = b$ within N iterations where N is the dimension of J.*

Proof. Let $p(z)$ be a characteristic polynomial of J, i.e.,

$$p(z) = det(J - zI), \tag{101}$$

where I is an $N \times N$ identity matrix. Then p is a polynomial of degree N. Since J is nonsingular, $p(0) = \det(J) \neq 0$. We define

$$p_N(z) = p(z)/p(0). \tag{102}$$

Hence, $p_N(x) \in \mathcal{P}_N$ is a residual polynomial. Then by the Cayley-Hamilton theorem (see [24] for example), we have

$$p_N(J) = p(J)/p(0) = 0. \tag{103}$$

The inequality (100) gives $r_N = b - Jx_N = 0$. Hence, x_N is the solution. □

Now, we recall that J is *diagonalizable* if there is a nonsingular matrix B and a diagonal matrix of eigenvalues $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_N)$ such that $J = B\Lambda B^{-1}$. And, if a matrix J is diagonalizable, then we have

$$p(J) = Bp(\Lambda)B^{-1}, \tag{104}$$

where p is a polynomial.

The structure of a diagonalizable matrix can be used to prove the following basic theorem on convergence of GMRES (see [18, 20, 40]).

Theorem 7 *Suppose J is a nonsingular diagonalizable matrix. At the kth step of the GMRES iteration, the residual r_k satisfies*

$$\frac{||r_k||}{||b||} \leq \kappa(B) \min_{p_k \in \mathcal{P}_k} \max_{i=1\ldots N} |p_k(\lambda_i)|, \tag{105}$$

where $\kappa(B) = ||B||\,||B^{-1}||$ is the condition number of the eigenvector matrix B.

Proof. Let $p_k \in \mathcal{P}_k$. Then

$$||r_k|| \leq \min_{p_k \in \mathcal{P}_k} ||Bp_k(\Lambda)B^{-1}b|| \leq \kappa(B) \min_{p_k \in \mathcal{P}_k} ||p_k(\Lambda)||\,||b||. \tag{106}$$

Hence,

$$\frac{||r_k||}{||b||} \leq \kappa(B) \min_{p_k \in \mathcal{P}_k} \max_{i=1\ldots N} |p_k(\lambda_i)|.$$

as was to be shown. □

4.4.3 Jacobian-Vector Products

GMRES requires only products between a matrix and a vector. These can be approximated by finite differences. For our problem, the product between the Jacobian matrix $J(x)$ and a vector s can be approximated by using the first two terms of the Taylor series

$$J(x)s \approx \frac{F(x+hs) - F(x)}{h}, \tag{107}$$

where h is some small user-supplied scalar. The technique described in (107) is sensitive to choices of h. A simple choice of h given in [23] is

$$h \approx \frac{\sqrt{(1+||x||)\epsilon_{mach}}}{||s||}, \tag{108}$$

where ϵ_{mach} is the value of machine epsilon. Another choice of h proposed by Brown and Saad [5] is

$$h = \frac{\sqrt{\epsilon_{mach}}}{||s||} \max\{|x^T s|, \operatorname{typ} x^T |s|\} \operatorname{sign}(x^T s), \tag{109}$$

where $\operatorname{typ} x$ is a typical size of x supplied by the user.

We note that the approximation in (107) is first-order. The second and fourth order approximations are given, respectively, by

$$J(x)s \approx \frac{1}{2h}\Big(F(x+hs) - F(x-hs)\Big), \tag{110}$$

$$J(x)s \approx \frac{1}{6h}\Big(8F(x+\frac{h}{2}s) - 8F(x-\frac{h}{2}s) - F(x+hs) + F(x-hs)\Big). \tag{111}$$

However, the higher-order approximations such as equations (110) and (111) are not popular because they increase the cost of function evaluations.

4.4.4 Preconditioning of the Newton-GMRES Method

We can reduce the number of GMRES iterations by preconditioning the GMRES method. In general, preconditioning here means that we apply the GMRES method to the equivalent system

$$(P_1^{-1} J P_2^{-1})(P_2 s) = P_1^{-1} b, \tag{112}$$

where $b = -F$. We choose the matrices P_1 and P_2 in advance so that the preconditioned system will be easier to solve than the original system. The matrices P_1 and

P_2 are referred to as left and right preconditioners, respectively. The system (112) is called *two-sided preconditioning*. If P_1 (or P_2) is an identity matrix, then the linear system is called *right (or left) preconditioning*. The GMRES algorithm using left and right preconditioning is shown in Figures 13 and 14, respectively.

Using the standard residual norm estimate available as a by-product of the Arnoldi iteration, left and right preconditioning terminate the linear iteration differently. Left preconditioning terminates the iteration when $||P_1^{-1}b||$ is minimum while right preconditioning terminates the iteration when $||b||$ is minimum. Only right preconditioning uses the residual for the original linear problem to terminate the linear iteration. However, left preconditioning terminates based on an estimate for the error, rather than the residual, as P_1 approaches the full Jacobian. Therefore, it is sometimes preferred.

In the right preconditioning, we actually solve two linear systems. First, we set $y = P_2 s$ and solve for y in

$$(JP_2^{-1})y = b. \tag{113}$$

Then we solve

$$s = P_2^{-1}y. \tag{114}$$

Ideally, we choose a preconditioner to be close to the inverse of the Jacobian matrix.

4.4.5 Globalization

Globalization of the Newton's method can be achieved by using a line search method. A line search method begins with a search direction s_k and decides how far to move along that direction. Thus, a line search method finds a positive scalar λ in

$$x_{k+1} = x_k + \lambda s_k. \tag{115}$$

The scalar λ is called the *step length*. We want to choose a λ that gives a substantial reduction of F. At the same time, we do not want to spend too much time making an optimal choice for any particular intermediate step λ. A simple condition used for choosing a step length λ is that the step length λ should provide a reduction in F, i.e.,

$$F(x_k + \lambda s_k) < F(x_k). \tag{116}$$

Algorithm

1 *Compute* $r_0 = P_1^{-1}(b - Js_0)$, $\beta = ||r_0||$ *and* $v_1 = r_0/||r_0||$
2 *For* $j = 1, 2, \ldots, k$ *Do*
3 *Compute* $w_{j+1} = P_1^{-1} J v_j$
4 *For* $i = 1, \ldots, j$ *Do*
5 $h_{ij} = (w_{j+1}, v_i)$
6 $w_{j+1} = w_{j+1} - h_{ij} v_i$
7 *EndDo*
8 $h_{j+1,j} = ||w_{j+1}||$
9 *If* $h_{j+1,j} = 0$ *set* $k = j$ *and go to 12*
10 $v_{j+1} = w_{j+1}/||w_{j+1}||$
11 *EndDo*
12 *Compute* y_k *the minimizer of* $||\beta e_1 - \tilde{H}_k y_k||$ *and* $s_k = s_0 + V_k y_k$
13 *End*

FIG. 13. *Left-preconditioned GMRES algorithm.*

Algorithm

1 *Compute* $r_0 = b - Js_0$, $\beta = ||r_0||$ *and* $v_1 = r_0/||r_0||$
2 *For* $j = 1, 2, \ldots, k$ *Do*
3 *Compute* $w_{j+1} = JP_2^{-1}v_j$
4 *For* $i = 1, \ldots, j$ *Do*
5 $h_{ij} = (w_{j+1}, v_i)$
6 $w_{j+1} = w_{j+1} - h_{ij}v_i$
7 *EndDo*
8 $h_{j+1,j} = ||w_{j+1}||$
9 *If* $h_{j+1,j} = 0$ *set* $k = j$ *and go to 12*
10 $v_{j+1} = w_{j+1}/||w_{j+1}||$
11 *EndDo*
12 *Compute* y_k *the minimizer of* $||\beta e_1 - \tilde{H}_k y_k||$ *and* $s_k = s_0 + P_2^{-1}V_k y_k$
13 *End*

FIG. 14. *Right-preconditioned GMRES algorithm.*

A popular line search condition is called *sufficient decrease condition* or the *Armijo condition*. Under this condition, the reduction in F is proportional to both step length and directional derivative, and it is given by [27]

$$F(x_k + \lambda s_k) \leq F(x_k) + c_1 \lambda \nabla F(x_k)^T s_k, \tag{117}$$

where $\nabla F(x) = (\partial F/\partial x_1, \cdots, \partial F/\partial x_N)^T$ and $c_1 \in (0, 1)$ is some constant. We refer the interested reader to [13] and [4] for detaild concerning global convergence of inexact Newton and Newton-GMRES methods, respectively.

CHAPTER 5

FIRESPREAD MODEL

In this chapter, we examine simple empirical models of wildland firespread. Most material covered in this chapter comes from [16].

5.1 FENDELL-WOLFF MODEL

In [16], Fendell and Wolff use experiments in an experimental fire tunnel test facility to motivate a semi-empirical firespread model. We outline their study of the model here. At any time t, the burned and unburned zones of the fire are demarcated by a smooth two-dimensional curve $\Gamma(s,t)$, where s parameterizes distance along the curve (Figure 15). The direction of the firespread at any point $x_i(s,t)$ is perpendicular to the interface where smooth. A finite set of cusps along Γ can be accommodated by defining the direction of firespread as the average of the normals on either side. The speed of the interface F at any point on the interface depends on model parameters $u^{(j)}$. We keep the number of parameters to a minimum: properties of the surface fuel, the terrain, and the weather believed necessary to reproduce important gross fire behavior. Thus, for any time $t > 0$, $i = 1$ or 2, a general formula for the advance of a point on in the interface is given by

$$\frac{\partial x_i(s,t)}{\partial t} = F\left(u^{(j)}\Big(x_i(s,t)\Big),\ n_i(s,t)\right),\ \text{ given }\ x_i(s,0), \tag{118}$$

where $n_i(s,t)$ denotes the outward normal (or a suitable average of normals). We rewrite equation (118) to identify the dependence of the propagation speed function F on the environment-describing variables as,

$$\frac{\partial x_i}{\partial t} = F\Big(U,\theta;m,...\Big)n_i,\ \text{ given }\ x_i(\theta,0),\ i=1,\ 2, \tag{119}$$

where U denotes the magnitude of the wind velocity (see Figure 16), n_i denotes the local outward normal to the firefront, θ denotes the angle (measured counterclockwise) between front normal and wind direction, m denotes the mass loading of thin fuel burned, and other parameters independent of the wind speed U and the angle θ can be added as arguments of the propagation speed function F.

For simplicity, we first assume a homogeneous fuel distribution so that a value of the angle θ is uniquely defined for each point on the firefront. The speed function

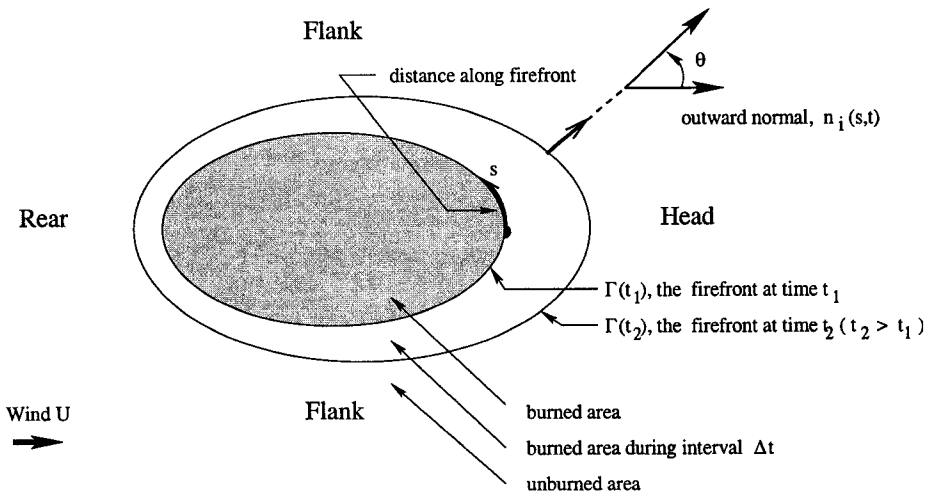

FIG. 15. *Burned area during the time interval Δt under the wind of magnitude U in the direction from left to right ($\theta = 0$). At any time t, the interface of the fire is denoted by $\Gamma(t)$.*

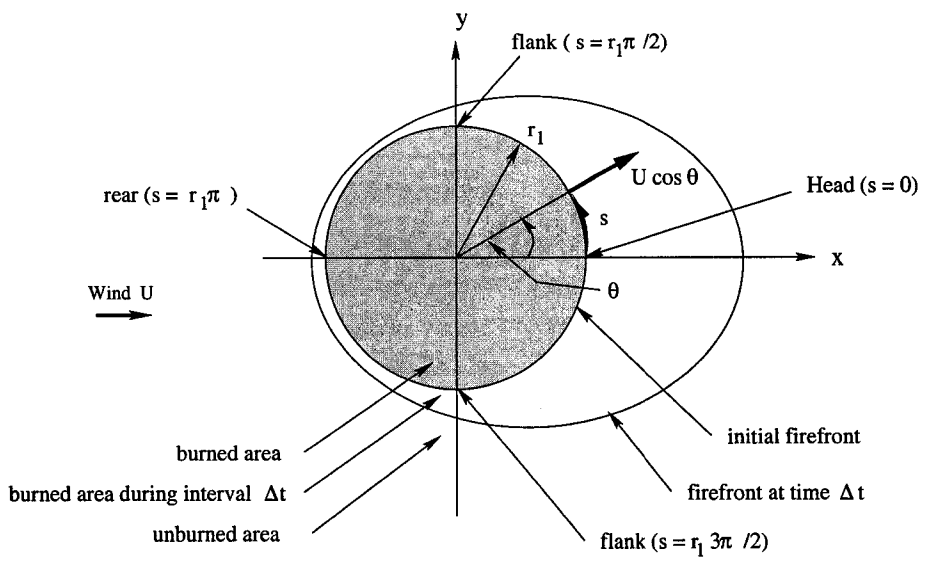

FIG. 16. *A circular initial burned area with radius r_1. The wind of magnitude U in the direction $\theta = 0$ results the fire growing in an elliptical shape during the time interval Δt.*

should depend only on the wind speed U and the angle θ. We parameterize the speed function in three key directions: downwind ($\theta = 0$), upwind ($\theta = \pi$), and cross-wind ($\theta = \frac{\pi}{2}, \frac{3\pi}{2}$). Following [16], we smoothly interpolate in angle between these directions. We denote the speed function in the downwind direction $F(U, 0)$, upwind direction $F(U, \pi)$ and cross-wind direction $F(U, \pi/2)$ by the symbols v_f, $\beta(U)$ and $\epsilon(U)$, respectively. In [16], Fendell and Wolff suggest that the local speed for a firefront, based on the wind speed and direction with respect to the local front, is given by ($\mu \approx 2 - 4$)

$$F(U,\theta) = \begin{cases} v_f(U\cos^2\theta) + \beta(U\sin^\mu\theta) & \text{if } |\theta| < \frac{\pi}{2} \\ \\ \beta(U\sin^\mu\theta) + \epsilon(U\cos^2\theta) & \text{if } |\theta| > \frac{\pi}{2} \end{cases}. \tag{120}$$

Since the value of the speed rate F at the head is higher than the value of the speed rate at the flank or at the back, we expect

$$\epsilon(U) < \beta(U) < v_f(U). \tag{121}$$

Let a vector $\vec{u}$ be the direction of the wind. Then the angle between the front normal $\vec{n}$ and the wind direction $\vec{u}$ is given by

$$\theta = \cos^{-1}\left(\frac{\vec{n}\cdot\vec{u}}{||\vec{n}||\,||\vec{u}||}\right). \tag{122}$$

If we set $\mu = 2$ in the equation (120) and the wind speed at the head, flank and back to be

$$v_f(U) = \epsilon_0 + c_1\sqrt{U}, \tag{123}$$

$$\beta(U) = \epsilon_0 + aU\exp(-bU), \tag{124}$$

$$\epsilon(U) = \epsilon_0\exp(-\epsilon_1 U), \tag{125}$$

respectively, then the equation for the speed of a firefront will be

$$F(U,\theta) = \begin{cases} \left(\epsilon_0\cos^2\theta + c_1\sqrt{U\cos\theta}\right) \\ +\left(\epsilon_0\sin^2\theta + aU\sin^2\theta\exp[-bU\sin^2\theta]\right) & \text{if } |\theta| < \frac{\pi}{2} \\ \\ \left(\epsilon_0\cos^2\theta\exp[-\epsilon_1 U\cos^2\theta]\right) \\ +\left(\epsilon_0\sin^2\theta + aU\sin^2\theta\exp[-bU\sin^2\theta]\right) & \text{if } |\theta| > \frac{\pi}{2} \end{cases}, \tag{126}$$

where the parameters c_1, ϵ_0, ϵ_1, a and b generally depend only on the mass loading of fuel parameter m. We use the equation (126) for a speed function F in the level set equation (7) in our research.

5.2 FENDELL-MALLET MODEL

In [25], Mallet applies the narrow band level set method to a simplified Fendell-Wolff model. He proposes that the speed of the firefront is given by

$$F(U,\theta) = \begin{cases} \epsilon_0 + c_1\sqrt{U}\cos^{\mu}\theta & \text{if } |\theta| < \frac{\pi}{2} \\ \\ \epsilon_0\big(\nu + (1-\nu)|\sin\theta|\big) & \text{if } |\theta| > \frac{\pi}{2} \end{cases}, \tag{127}$$

where ν is the CFL number given in (50). This model is not directly based on physical theories of firespread but qualitatively captures essential features of wind-driven firespread. Mallet also proposes that using $\mu = 3/2$ in equation (120) or (127) yields the best results in eye-ball comparisons to the test facility experiments of Fendell and Wolff.

CHAPTER 6

NUMERICAL EXPERIMENTS

In this chapter, we perform some tests of numerical accuracy of the level set method and illustrate the firespread model. We implement the implicit level set method using the PETSc (Portable, Extensible Toolkit for Scientific Computation) [2] software library. The code is run in parallel on either a SUN Enterprise 2900 server (Hydra) or a SUN Enterprise 10000 Starfire (E10k) server (Helios) at Old Dominion University (ODU).

6.1 ERROR ANALYSIS

We use a circle expanding with unit speed to test the accuracy of the numerical scheme. The parameters for the test are shown in Table 1.

6.1.1 Spatial Error

To perform spatial error analysis, we allow a circle of radius 0.5 expanding with speed $F = 1$. Then we compute the radius of the circle at the end of simulation and compare it with the exact radius. We measure an effective radius at the end of simulation by

$$r_{approx.} = \frac{1}{card(\Gamma)} \sum_{(x,y)\in\Gamma} \sqrt{(x-1.5)^2 + (y-1.5)^2}, \tag{128}$$

where Γ is a set of points on the circle, and $card(\Gamma)$ is the number of elements (points) in Γ. Then, the spatial error can be estimated by

$$e_s = |r_{exact} - r_{approx.}|. \tag{129}$$

In this test, we allow the circle to expand until the final time $T_f = 0.3$. Hence the exact radius of circle, r_{exact}, at the final time is 0.8. We choose time step $\Delta t = 0.0008$. Then, we choose small spatial steps satisfying CFL condition ($\Delta x \geq \Delta t$ and $\Delta y \geq \Delta t$). In our test, we keep $\Delta x = \Delta y$. The spatial error is shown in the Table 2. We can see that the spatial error of Lax-Friedrichs scheme is higher than those of Upwind scheme due to a dissipation presented in the Lax-Friedrichs scheme. We plot the absolute spatial error for both upwind and Lax-Friedrichs scheme on logarithmic

TABLE 1
Data relating to the test.

Data	Value
Domain	$\Omega = [0,3] \times [0,3]$
Initial front	Circle
Center of circle	$(1.5, 1.5)$
Initial radius	0.5
Speed of front	$F = 1.0$

TABLE 2
Spatial Error analysis when Δt, Δx and Δy satisfy CFL condition.

Number of points	$\Delta x = \Delta y$	Upwind ($\times 10^{-2}$)	Lax-Friedrichs ($\times 10^{-2}$)
50×50	0.06122	0.7033	1.4278
100×100	0.03030	0.3336	0.7234
225×225	0.01339	0.1434	0.3200
500×500	0.00601	0.0636	0.1414

scale. The graphs of the absolute spatial error are shown in Figure 17. We obtain perfectly linear curves with slope 1.0355 for the upwind scheme and 0.9962 for the Lax-Friedrichs schemes. This confirms that both schemes are of order 1.

Next, we set time step $\Delta t = 0.1$ and then choose small spatial step Δx and Δy such that they violate the CFL condition. The spatial errors for this test are shown in Table 3. We can see that the spatial error when Δt, Δx and Δy violate CFL condition is slightly higher than the spatial error when Δt, Δx and Δy satisfy the CFL condition. Graphs of the absolute spatial error using upwind and Lax-Friedrichs scheme are shown in Figure 18. The slopes of the graph for upwind and Lax-Friedrichs scheme are 1.0416 and 0.9952, respectively. Again, both the upwind and Lax-Friedrichs schemes are of order 1.

6.1.2 Temporal Error

To perform error analysis in time, we use a circle of radius 0.5 centered at (1.5,1.5) as an initial curve. Then, we allow the circle to expand with speed $F = 1$ until the circle crosses the point $p = (2.3, 1.5)$. We keep a time history of the local value of ϕ at point p and use interpolation in time to find the time t_c when ϕ equals to zero. This is the time to take the zero level set to reach the point p. We refer that time as the *crossing time*. The exact crossing time t_{exact} can be computed by using the equation of a sign distance function

$$\phi(x, y) = \sqrt{(x - x_0)^2 + (y - y_0)^2} - a - F * t, \tag{130}$$

where (x_0, y_0) is a center of the circle, and a is an initial radius. Hence, the exact crossing time, when the front crosses the point $p = (2.3, 1.5)$, is 0.3. Thus, the absolute crossing time error can be computed by

$$e_t = |t_{exact} - t_c|. \tag{131}$$

In this test, we keep $\Delta x = \Delta y = 0.01339$. Thus, the dimension of the computational grid is 225×225. We use a forward Euler method to perform a discretization in time. The absolute crossing time errors for both upwind and Lax-Friedrichs schemes are shown in Table 4. There is nonnegligible monotonic improvement in temporal error for the upwind scheme with decreasing Δt. However, for the Lax-Friedrichs scheme, the improvement is not consistent. Temporal convergence needs further investigation. From [9] we expect a one-half order convergence rate in Δt. We conclude that the

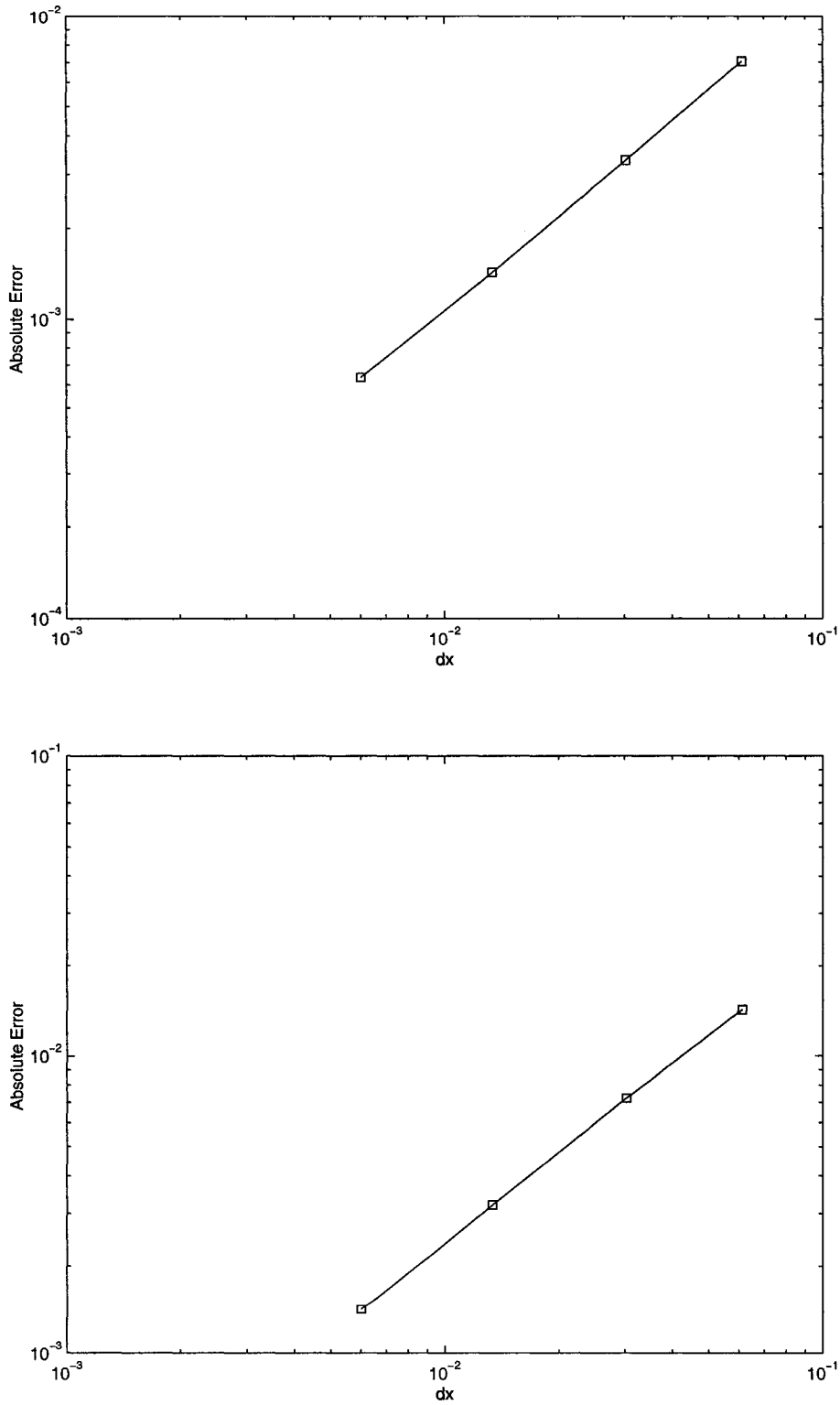

FIG. 17. *Absolute spatial error (log-log scale) when Δt, Δx and Δy satisfy CFL condition. Top: upwind scheme. Bottom: Lax-Friedrichs scheme. The slopes are 1.0355 and 0.9962, respectively.*

TABLE 3
Spatial Error analysis when Δt, Δx and Δy violate CFL condition.

Number of points	$\Delta x = \Delta y$	Upwind ($\times 10^{-2}$)	Lax-Friedrichs ($\times 10^{-2}$)
50×50	0.06122	0.7225	1.4447
100×100	0.03030	0.3388	0.7328
225×225	0.01339	0.1453	0.3244
500×500	0.00601	0.0644	0.1434

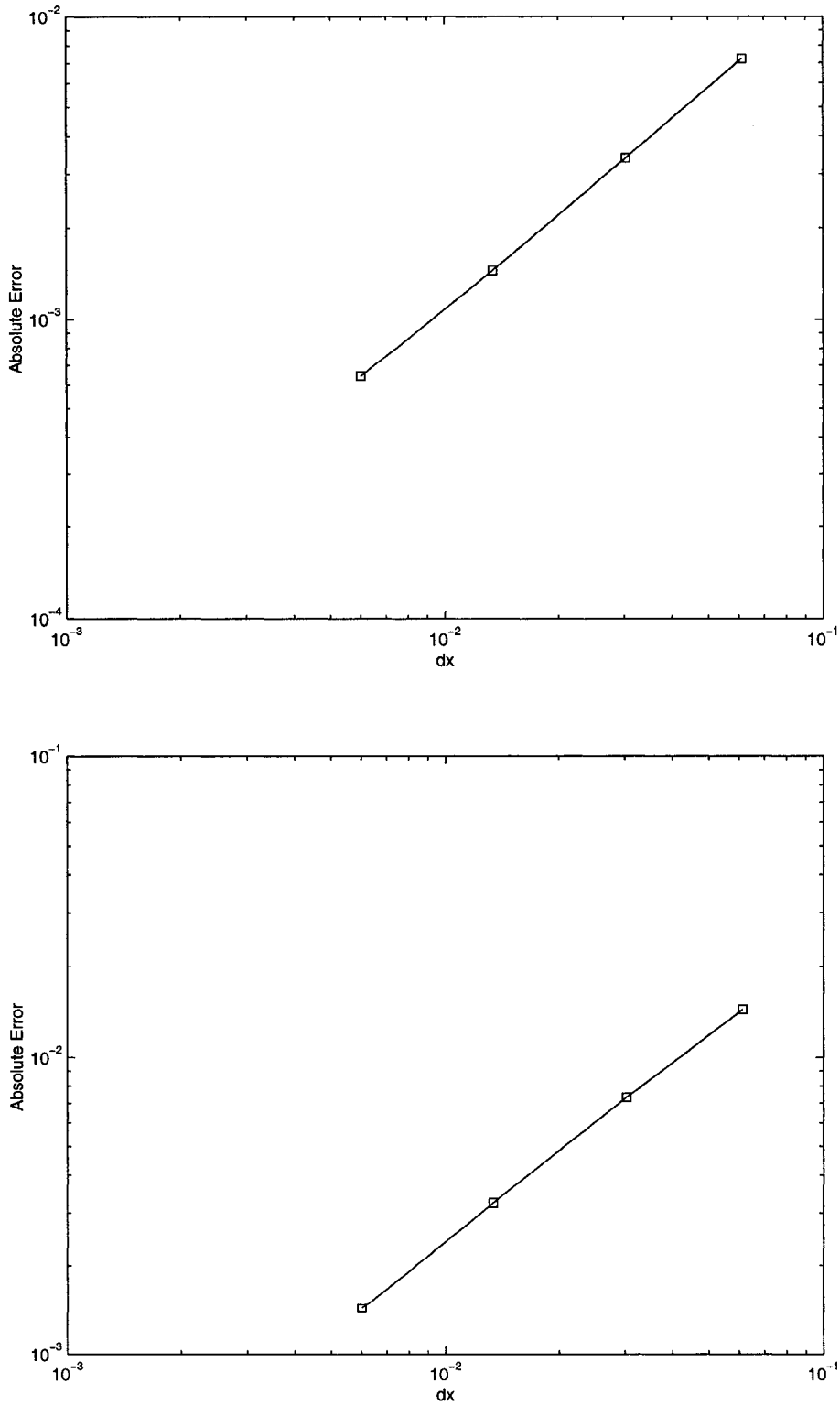

FIG. 18. *Absolute spatial error (log-log scale) when Δt, Δx and Δy violate CFL condition. Top: upwind scheme. Bottom: Lax-Friedrichs scheme. The slopes are 1.0416 and 0.9952, respectively.*

TABLE 4
Absolute crossing time solution error of upwind scheme and Lax-Friedrichs scheme.

Number of iteration	Δt	Upwind ($\times 10^{-2}$)	Lax-Friedrichs ($\times 10^{-2}$)
30	0.0100	0.0310	0.3078
100	0.0030	0.0288	0.2708
250	0.0012	0.0246	0.3004
3000	0.0001	0.0049	0.2951

dependence of the accuracy of the solution on the timestep is weak within practical stable ranges.

6.2 EXPLICIT AND IMPLICIT METHODS

In this section, we perform an numerical experiment that demonstrates superior stability properties for the implicit level set method relative to explicit. We use a circle of radius 0.5 centered at $(1, 1.5)$ as the initial curve. We choose a computational domain $\Omega = [0, 4] \times [0, 4]$ and use the Fendell-Wolff model with $U = 1.0$, $a = 0.1$, $b = 1.0$, $\epsilon_0 = 0.1$, $\epsilon_1 = 0.003$, $c_1 = 0.5$ and $\mu = 1.5$ in this experiment. The spatial step $\Delta x = \Delta y = 0.02$. We choose $\Delta t = 0.1$, in violation of the CFL condition. The results of the experiment using explicit and implicit forms of the level set method are shown in Figure 19 and 20, respectively. The explicit level set method with a sufficiently large time step yields an unstable result (see Figure 19). The head of the fire front, where the CFL violation is greatest disintegrates as time progresses. When we use the implicit method with a large time step, we maintain a stable result (see Figure 20). The head of the fire front is smooth for all the time. This robustness is a useful feature in a code intended to be passed on fire modelers and incorporated in a parameter identification loop for tuning model parameters. For the implicit method, the average number of Newton iterations per time step is 4 for the Lax-Friedrichs scheme and 6 for the upwind scheme, using PETSc's rather stringent relative-absolute error tolerance defaults.

6.3 NUMERICAL EXPERIMENTS

In this section, we perform numerical experiments on our firespread models. We use both Fendell-Wolff model (120) and Fendell-Mallet firespread model (127) with $\mu = 1.5$ in our experiment. The code is run parallel using 8 processors on a SUN enterprise 2900 server. We use matrix free method and PETSc's default convergence criterion in the experiments. We choose computation domain $\Omega = [0, 3] \times [0, 3]$. The spatial step $\Delta x = \Delta y = 0.01$. Thus, the dimension of the computational grid is 301×301. We choose time step $\Delta t = 0.02$, and the final time T_f is 0.8. Since data for realistic fires is not available at the time of writing, we arbitrarily choose fire parameters as shown in Table 5. Note that only parameters U, ϵ_0 and c_1 are needed for the Fendell-Mallet model.

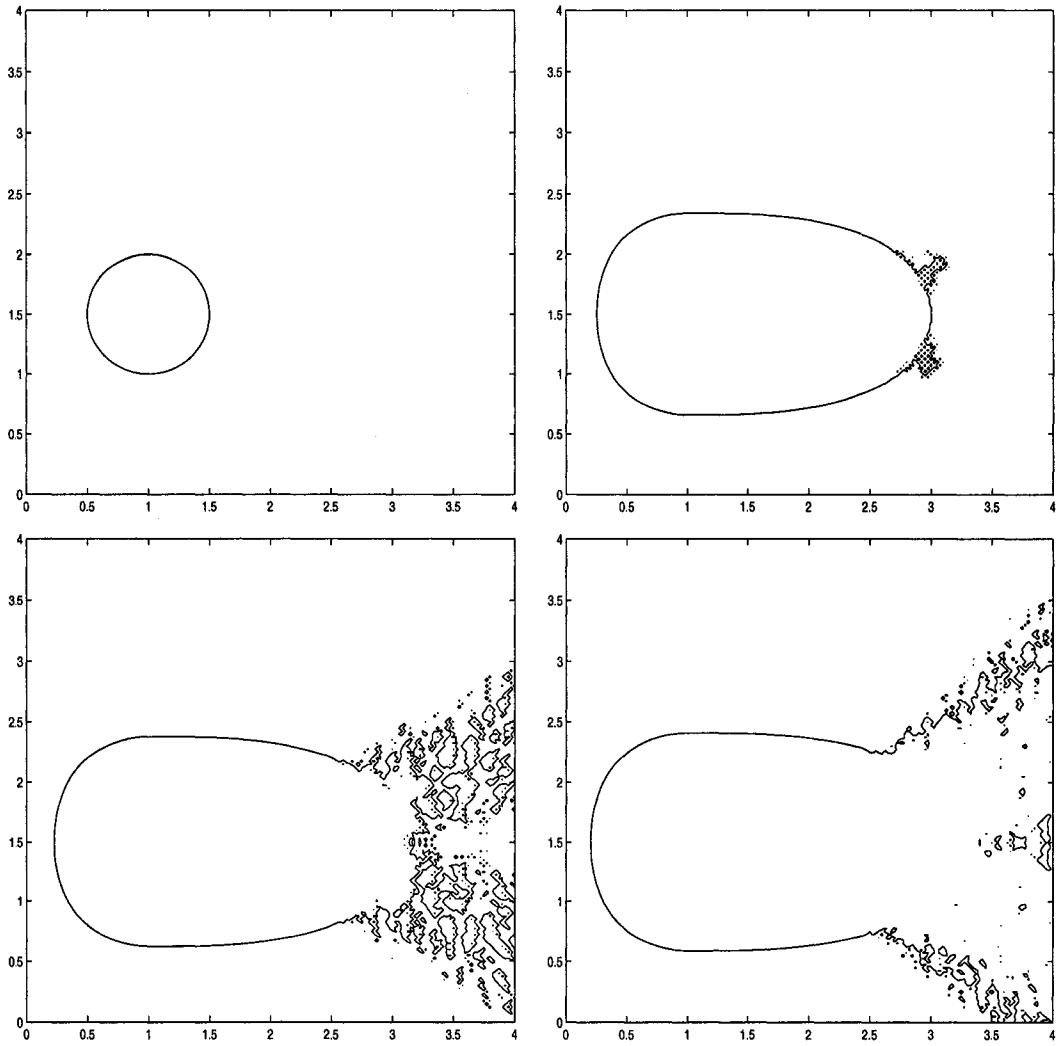

FIG. 19. *Instability results from using a too large a time step ($\Delta t = 0.1$) in the explicit level set method. Top left corner: $t = 0$. Top right corner: $t = 2.0$. Bottom left corner: $t = 2.2$. Bottom right corner: $t = 2.4$.*

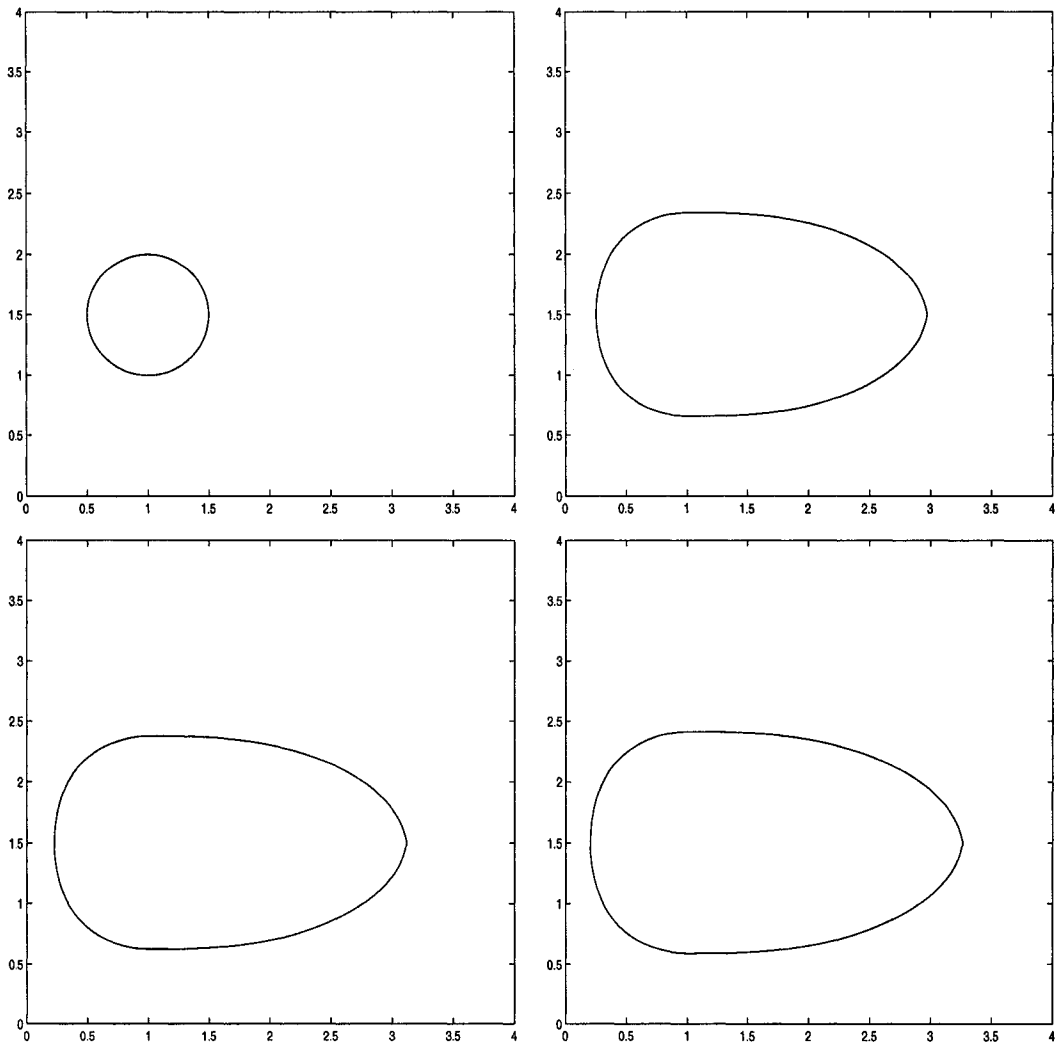

FIG. 20. *Stability results from using a large time step ($\Delta t = 0.1$) in the implicit level set method. Top left corner: $t = 0$. Top right corner: $t = 2.0$. Bottom left corner: $t = 2.2$. Bottom right corner: $t = 2.4$.*

TABLE 5
Value of fire parameters related to the experiments.

Parameter	Value
U	5.0
a	0.1
b	1.0
ϵ_0	0.1
ϵ_1	0.003
c_1	0.5
μ	1.5

For comparison purposes, each figure shows superimposed front positions at regularly spaced times. Also, two figures of the same numerical experiment are shown together on the same page. In most of the figures, the top shows the history obtained from the upwind scheme, while the bottom one shows that obtained from Lax-Friedrichs.

We begin our experiment with a comparison between Fendell-Wolff and Fendell-Mallet model. We use a circle of radius 0.5 centered at $(1, 1.5)$ as an initial curve. We allow a wind constantly blows from the west to the east. The result of experiment using upwind scheme and Lax-Friedrichs scheme are shown in Figure 21 and 22, respectively. For both fire model, the fire advance at the head much faster than at the flank and rear as intended. Figure 23 shows a fire simulation of the Fendell-Mallet model using upwind scheme and Lax-Friedrichs scheme from the same experiment. Since Lax-Friedrichs scheme introduces considerable diffusion into the solution, the fire front at the head computed by using Lax-Friedrichs scheme is less sharp than those computed by using upwind scheme.

Next, we perform an experiment when a wind blows from the west and slowly turns to acquire a southerly component. We use a Fendell-Mallet model in this experiment. All set ups for the initial curve and fire parameters are as in Table 5. The result is shown in Figure 24. Both upwind and Lax-Friedrichs scheme yield a similar result. The fire initially advances to the east and then gradually advances to the north-east.

Next, we perform an experiment in which we have three initial curves. The first curve is a circle of radius 0.5 centered at $(1, 1)$, the second is a circle of radius 0.3 centered at $(1.5, 1.8)$, and the third is a circle of radius 0.25 centered at $(1, 1)$. Here, the third curve demarcates an unburnt island inside of a burning region. We use the Fendell-Wolff model in this experiment. The fire parameters are as shown in Table 5. All $\Delta x, \Delta y$ and Δt are the same. The final time is $T_f = 0.5$. The result is shown in Figure 25. We observe the unburnt island gradually disappearing and the two fire fronts eventually merging into a single fire.

Finally, we perform an experiment for which the energy density of the fuel varies across an east-west line. We begin with a circular fire of radius 0.4 centered at $(1.5, 1.5)$. We set the region above the line $y = 2$ to have a low energy density while the region below the line $y = 2$ has high density. In this test, there is no wind. We use $\Delta x = \Delta y = 0.01$, $\Delta t = 0.02$ and $T_f = 0.8$. The result is shown in Figure 26. As

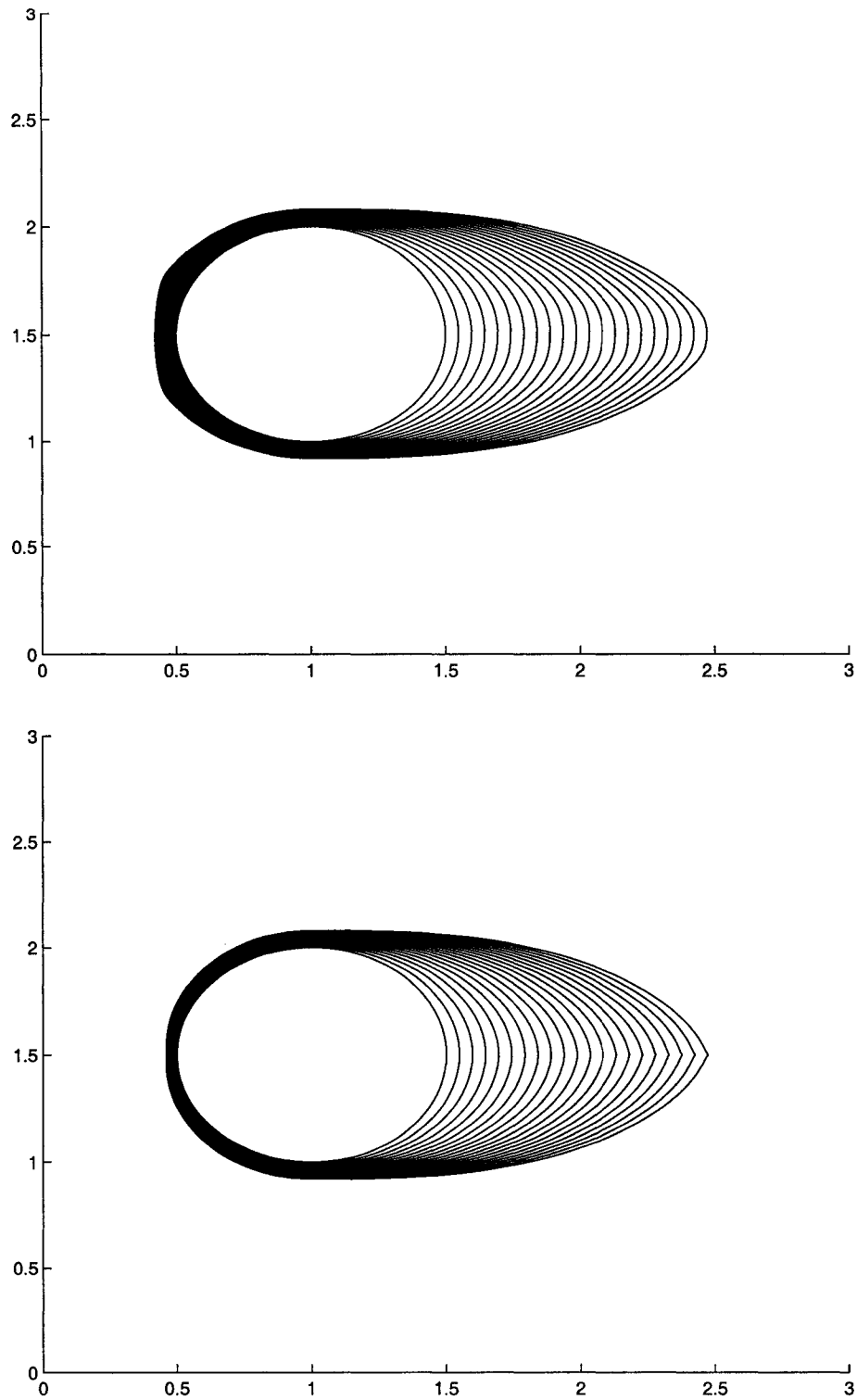

FIG. 21. *A comparison between Fendell-Wolff and Fendell-Mallet model using upwind scheme. Top: Fendell-Wolff model. Bottom: Fendell-Mallet model.*

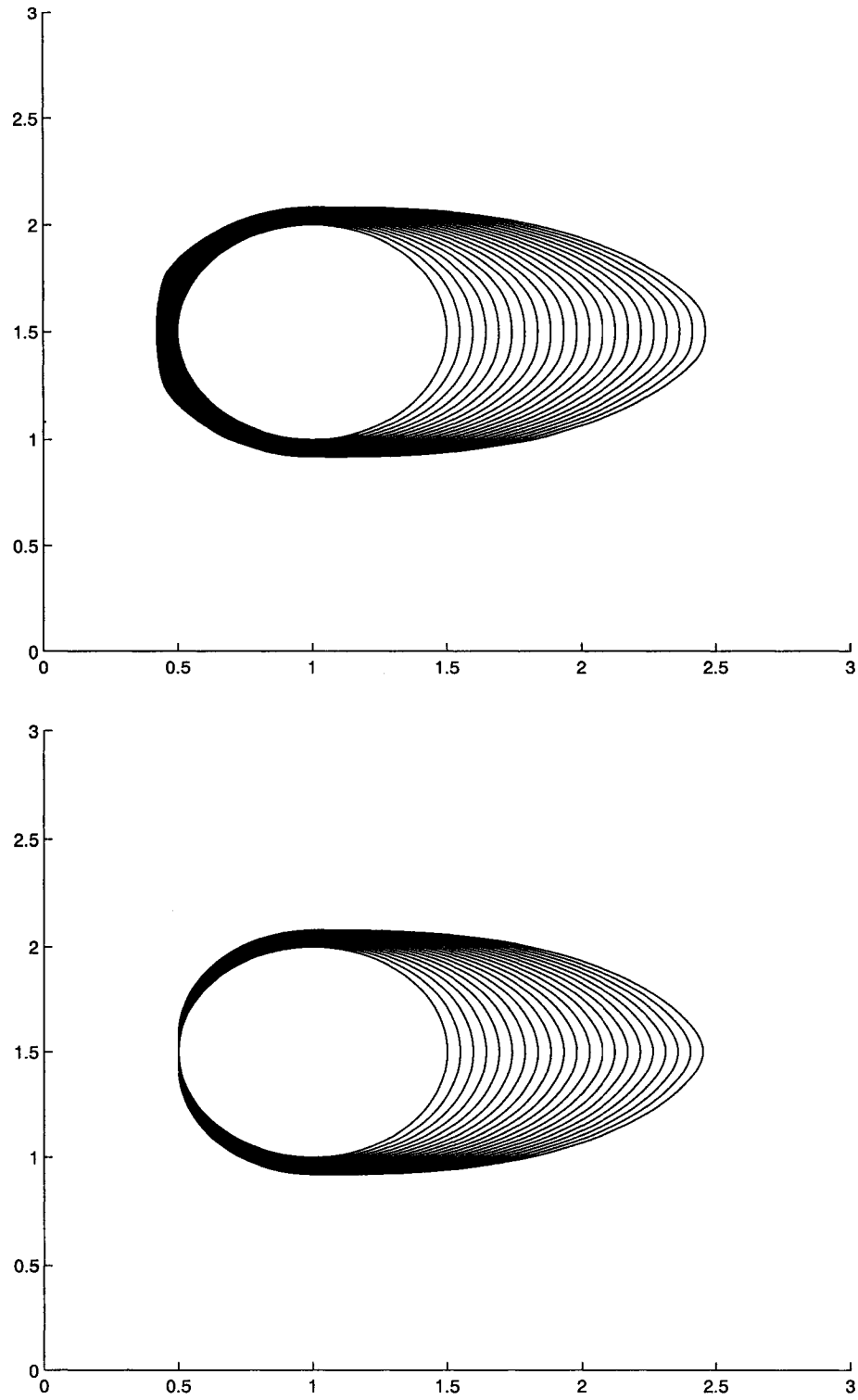

FIG. 22. *A comparison between Fendell-Wolff and Fendell-Mallet model using Lax-Friedrichs scheme. Top: Fendell-Wolff model. Bottom: Fendell-Mallet model.*

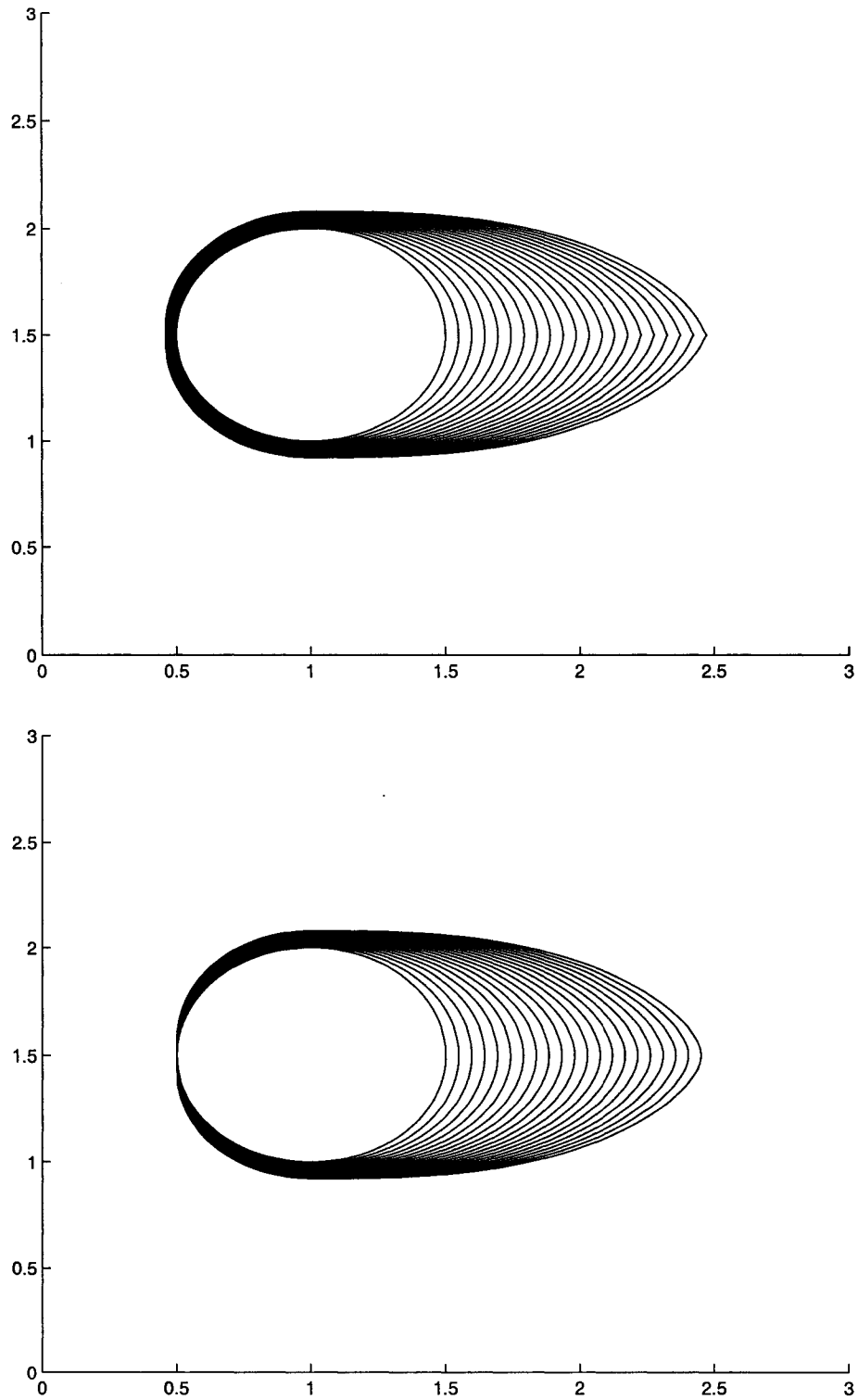

FIG. 23. *Fire propagation under a steady wind. Top: upwind scheme. Bottom: Lax-Friedrichs scheme.*

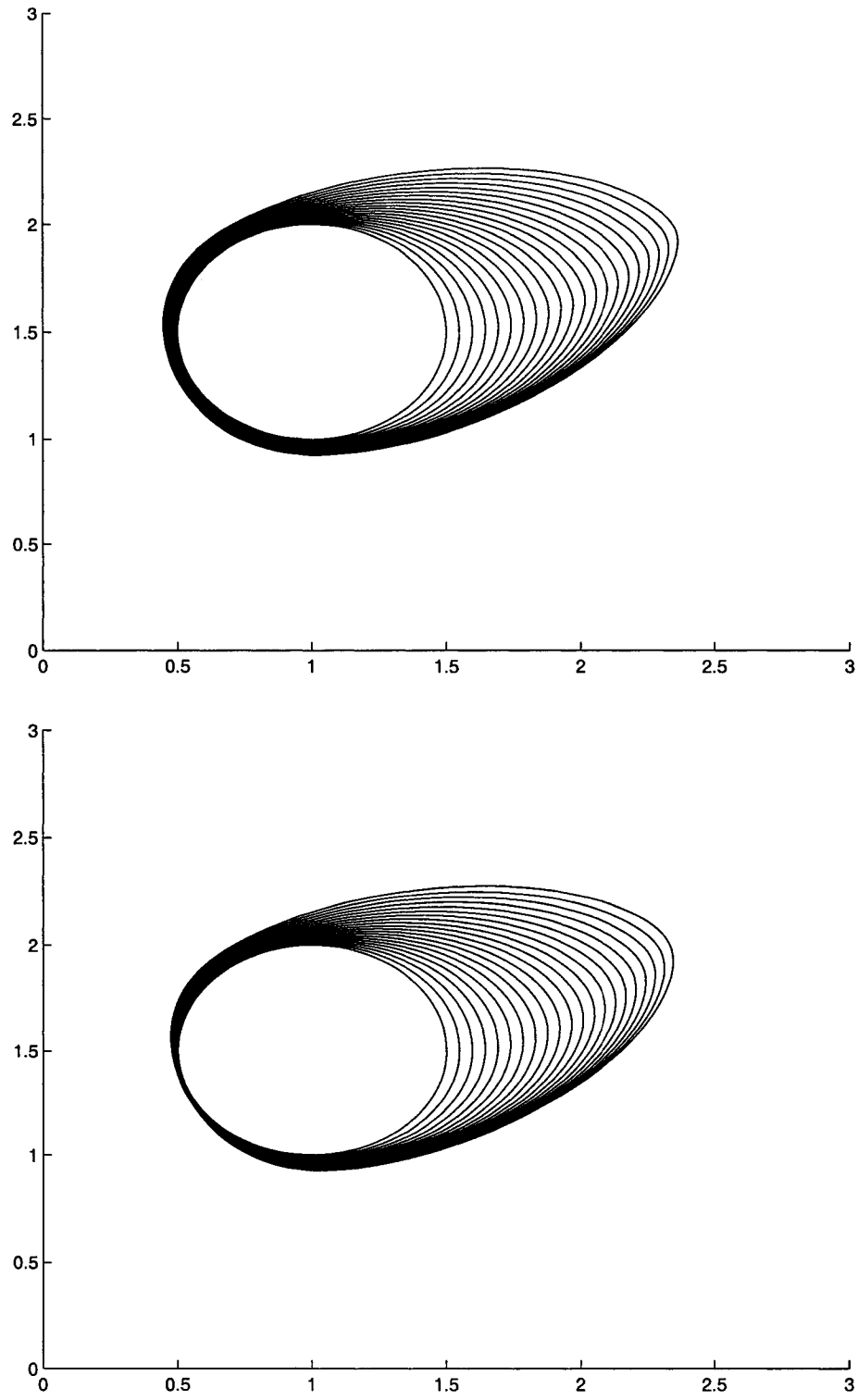

FIG. 24. *Fire propagation under a turning wind. Top: upwind scheme. Bottom: Lax-Friedrichs scheme.*

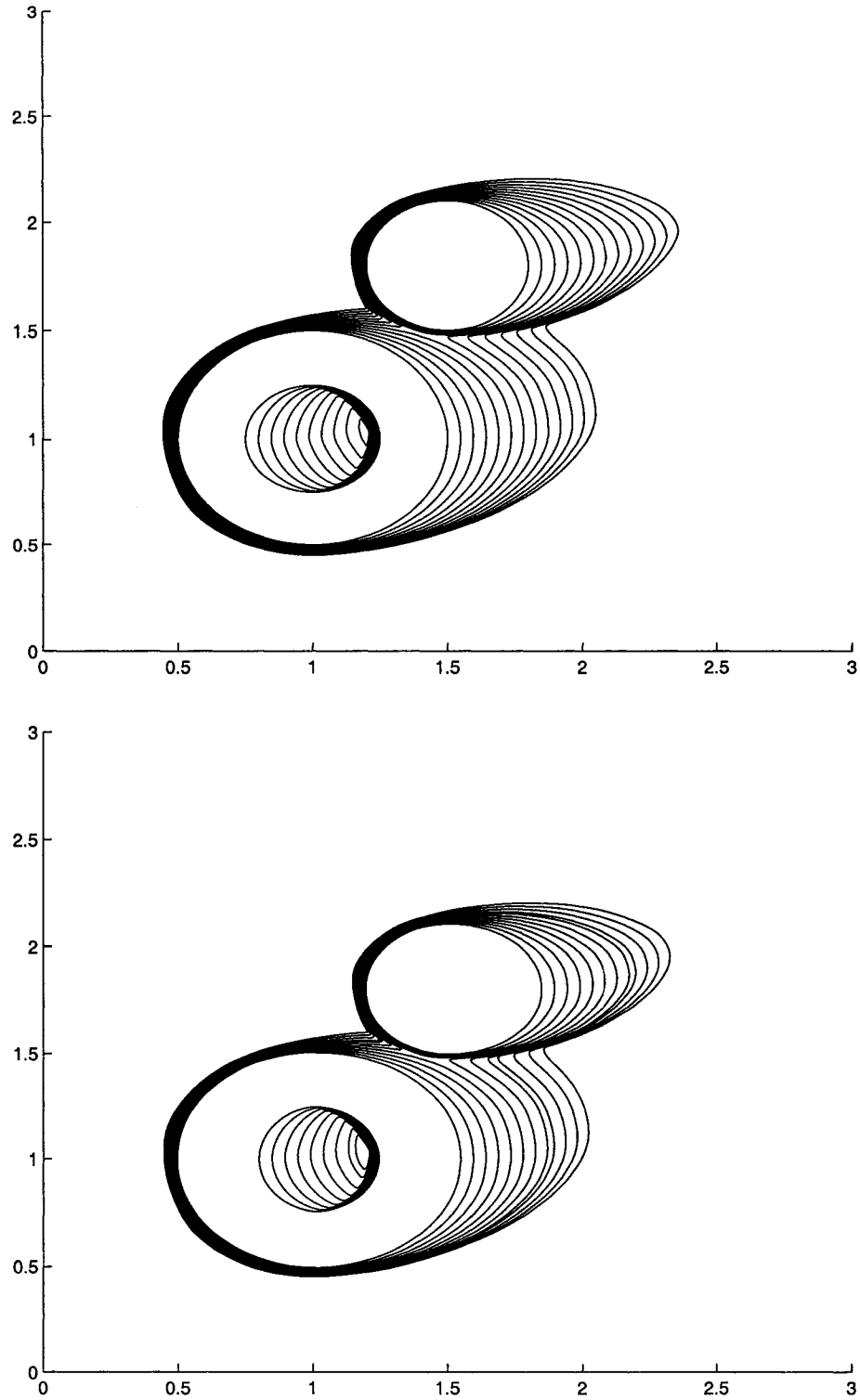

FIG. 25. *Fire propagation of two elliptical fires, one with an unburnt area. The fires evolve under a wind from the west and gradually turns to be from the south. Top: upwind scheme. Bottom: Lax-Friedrichs scheme.*

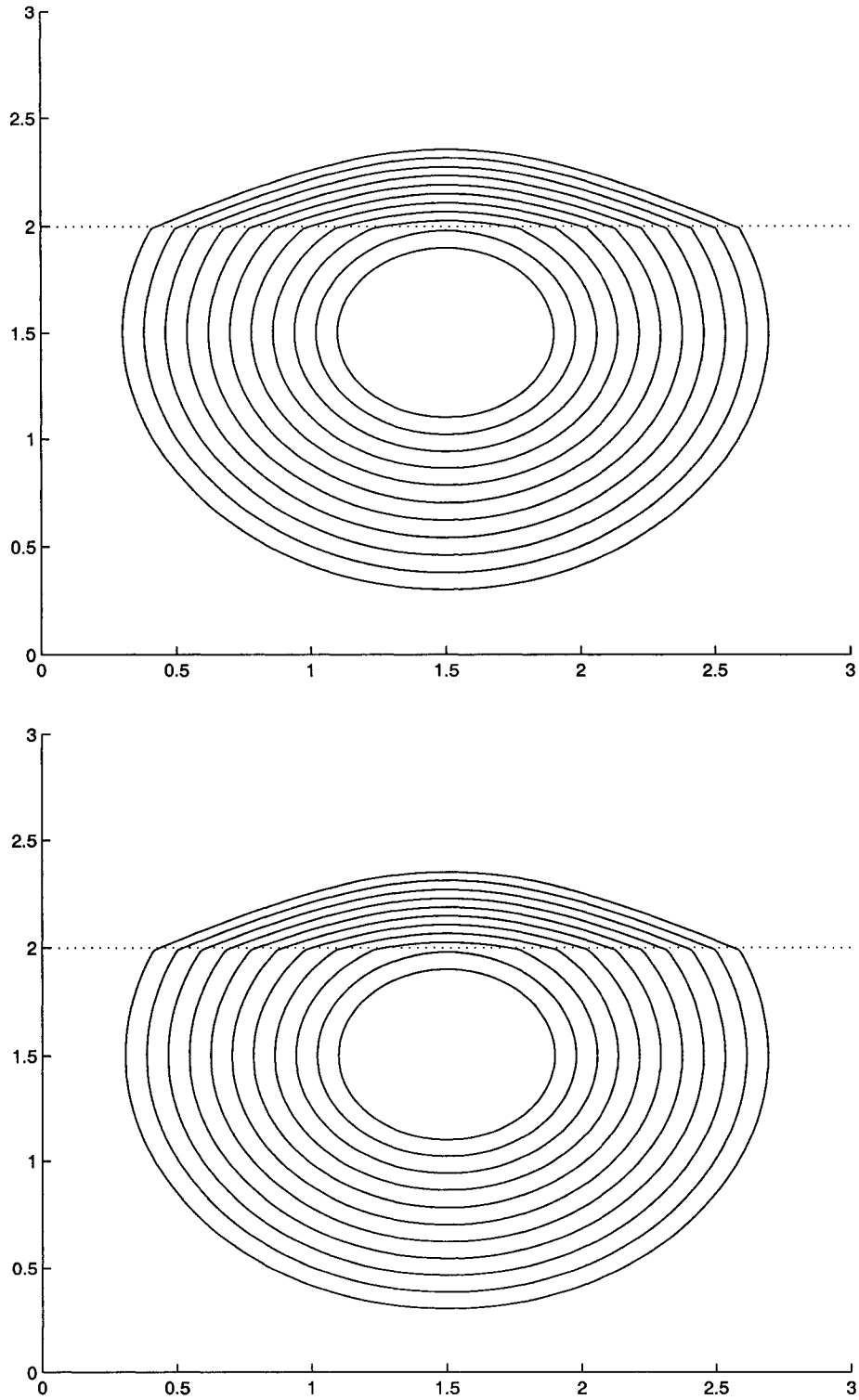

FIG. 26. *An elliptical fire propagates without wind from a region of high fuel density into a region of low fuel density. Top: upwind scheme. Bottom: Lax-Friedrichs scheme.*

intended, the fire advances more quickly within the high energy density region. Once the fire reaches the low density region, it advances more slowly.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis, we apply the level set method to a two-dimensional empirical model of wildland firespread. The derivation of the level set method and some theoretical background of the Hamilton-Jacobi equation and its numerical approximation are reviewed. This leads to a discussion of methods of solving large systems of nonlinear algebraic equations. We implemented the implicit level set method for a firespread model using the PETSc [2] software library. We have used matrix-free Newton-GMRES as a nonlinear solver in order to avoid explicitly forming a Jacobian matrix. The code is natively parallel.

The main goal of this thesis is to introduce a implicit new level set method for wildland firespread. An advantage of an implicit method is that it is possible to use a time step of arbitrary size, determined only by accuracy criteria and free of instability. Our numerical experiments show that the implicit level set method works well with a large time step.

We have checked spatial and temporal errors. Although we do not possess exact solutions for general problems, we can compare the numerical results to the analytical solution of a circle expanding with unit speed. We confirm that both upwind and Lax-Friedrichs schemes are of first order in space.

For temporal error analysis, we use crossing time to measure error. We calculate a time when the circle crosses a given point, and then we compare it with the exact crossing time. We use a forward Euler method as a time discretization. However, in our experiments, we cannot verify that the method is of order 0.5 in time, as expected from the theory. The same result was reported in [30] and remains to be explained.

We have implemented the implicit level set methods on firespread models. The Fendell-Mallet model gives results similar to those of the Fendell-Wolff model, for which it is intended as a simple replacement, particularly for follow-on studies in parameter estimation, where Fendell-Mallet is easier to differentiate. We also compare upwind and Lax-Friedrichs scheme on the same model.

The upwind and Lax-Friedrichs schemes yield similar results. However, the Lax-Friedrich scheme appears slightly more diffusive, as indicated by the smoothing of corners in Figure 23. In the Lax-Friedrichs scheme, we need to calculate dissipation

coefficients. Hence, for each time step, the Lax-Friedrichs scheme requires more calculations than the upwind scheme. However, introducing some dissipation in the Lax-Friedrichs scheme seems to allow the Newton-GMRES algorithm to more easily find a solution.

Although our implicit level set method works well, several improvements can be made to the method to make it more efficient. In this thesis, we have used a full matrix approach of the level set method where the value of ϕ is calculated at every point on the computational domain. This can be expensive. A narrow band approach in which only computational grids close to the zero level set are used would be worthwhile to implement. This would require dynamic data structures, but would significantly reduce the computational complexity. We can also improve the solver by by developing preconditioners for the GMRES solves in Newton-GMRES that use partial Jacobian information.

REFERENCES

[1] D. H. ANDERSON, E. A. CATCHPOLE, N. J. DE MESTRE, AND T. PARKES, *Modeling the spread of grass fires*, J. Aust. Math. Soc. (Ser. B), 23 (1982), pp. 451–466.

[2] S. BALAY, K. BUSCHELMAN, V. EIJKHOUT, W. GROPP, D. KAUSHIK, M. KNEPLEY, L. C. MCINNES, B. F. SMITH, AND H. ZHANG, *PETSc home page.* http://www-unix.mcs.anl.gov/petsc/petsc-2, 2005.

[3] R. BARRETT, M. BERRY, T. F. CHAN, J. DEMMEL, J. M. DONATO, J. DONGARRA, V. EIJKHOUT, R. POZO, C. ROMINE, AND H. VAN DER VORST, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.

[4] S. BELLAVIA AND B. MORINI, *A globally convergent Newton-GMRES subspace bethod for systems of nonlinear equations*, SIAM J. Sci. Comput., 23 (2001), pp. 940–960.

[5] P. N. BROWN AND Y. SAAD, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 450–481.

[6] D. CHOPP, *Computing minimal surfaces via level set curvature flow*, J. Comput. Phys., 106, pp. 77-91, 1993.

[7] T. L. CLARK, J. COEN, AND D. LATHAM, *Description of a coupled atmosphere-fire model*, Int. J. Wildland Fire, 13 (2004), pp. 49–63.

[8] R. COURANT, K. FRIEDRICHS, AND H. LEWY, *Ueber die partiellen Differenzengleichungen der mathematischen Physik*, Mathematisiche Annalen, 100 (1928), pp. 32–74.

[9] M. G. CRANDALL AND P. L. LIONS, *Viscosity solutions of Hamilton-Jacobi equations*, Trans. Am. Math. Soc., 227 (1983), pp. 1–42.

[10] M. G. CRANDALL AND P. L. LIONS, *Two approximation of solutions of Hamilton-Jacobi equations*, Math. Comput., 43 (1984), pp. 1–19.

[11] R. S. DEMBO, S. C. EISENSTAT, AND T. STEIHAUG, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[12] S. C. EISENSTAT AND H. F. WALKER, *Choosing the forcing terms in an inexact Newton method*, SIAM J. Sci. Comput., 17 (1996), pp. 16–32.

[13] S. C. EISENSTAT AND H. F. WALKER, *Globally convergent inexact Newton methods*, SIAM J. Opt., 4 (1994), pp. 393-422.

[14] L. C. EVANS, *Partial Differential Equations*, American Mathematical Society, Rhode Island, 1998.

[15] M. A. FINNEY, *FARSITE: Fire Area Simulator-Model Development and Evaluation*, Tech. Rep., USDA Forest Service, 1998.

[16] F. E. FENDELL AND M. F. WOLFF, *Wildland fire spread models*, in Forest Fires: Behavior and Ecological Effects, E. A. Johnson and K. Miyanishi, eds., Academic Press, New York, 2001, pp. 171–223.

[17] J. GLIMM, J. W. GROVE, X. L. LI, AND N. ZHAO, *Simple front tracking*, Contemporary Math., 238 (1999) pp. 133-149.

[18] A. GREENBAUM, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, PA, 1997.

[19] A. HARTEN, B. ENGQUIST, S. OSHER, AND S. R. CHAKRAVARTHY, *Uniformly high order accurate essentially non-sscillatory schemes, III*, J. Comput. Phys., 71 (1987), pp. 231-303.

[20] C. T. KELLEY, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, PA, 1995.

[21] C. T. KELLEY, *Solving Nonlinear Equations with Newton's Method*, SIAM, Philadelphia, PA, 2003.

[22] D. KINCAID AND W. CHENEY, *Numerical Analysis*, 2nd Edition, Brooks/Cole, Pacific Grove, CA, 1996.

[23] D. A. KNOLL AND D. E. KEYES, *Jacobian-free Newton-Krylov methods: a survey of approaches and applications*, J. Comput. Phys., 193 (2004), pp. 357–397.

[24] P. D. LAX, *Linear Algebra*, John Willey & Sons, NY, 1996.

[25] V. MALLET, *Simulation numérique de la propagation d'interfaces*, M.S. thesis, Ecole Centrale de Lyon, France, 2002.

[26] NATIONAL INTERGENCY FIRE CENTER, Retrieved March 29, 2006 from http://www.nifc.gov

[27] J. NOCEDAL AND S. J. WRIGHT, *Numerical Optimization*, Springer–Verlag, 1999.

[28] W. NOH AND P. WOODWARD, *A Simple Line Interface Calculation*, Springer–Verlag, 1976.

[29] S. OSHER AND R. P. FEDKIW, *Level set methods: an overview and some recent results*, J. Comput. Phys., 169 (2001), pp. 463–502.

[30] S. OSHER AND R. P. FEDKIW, *Level Set Methods and Dynamic Implicit Surfaces*, Springer–Verlag, 2003.

[31] S. OSHER AND J. A. SETHIAN, *Fronts propagating with curvature dependent seed: algorithms based on Hamilton-Jacobi formulations*, J. Comput. Phys., 79 (1988), pp. 12–49.

[32] S. OSHER AND C.-W. SHU, *High-order essentially non-oscillatory schemes for Hamilton-Jacobi equations*, SIAM J. Numer. Anal., 28 (1991), pp. 907–922.

[33] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, 2000.

[34] Y. SAAD, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comput., 37 (1981), pp. 105–126.

[35] Y. SAAD AND M. H. SCHULTZ, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[36] J. A. SETHIAN, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Cambridge University Press, Cambridge, 2002.

[37] C.-W. SHU, *Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws*, Tech. Report 1997-65, ICASE, NASA Langley Research Center, Hampton, VA, 2000.

[38] C.-W. SHU AND S. OSHER, *Efficient implementation of essentially non-oscillatory shock capturing schemes, II*, J. Comput. Phys., 83 (1989), pp. 32–78.

[39] G. A. SOD, *Numerical Methods in Fluid Dynamics: Initial and Initial Boundary-Value Problems*, Cambridge University Press, Cambridge, 1985.

[40] L. N. TREFETHEN AND D. BAU, III, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[41] H. VAN DER VORST, *Iterative Krylov Methods for Large Linear Systems*, Cambridge University Press, Cambridge, 2003.

APPENDIX A

PETSC CODE FOR NUMERICAL EXPERIMENTS

A.1 PETSC ROUTINE FOR SOLVING A LEVEL SET EQUATION

The following PETSc routine contains the setup for solving a level set equation using upwind scheme.

```
/*$Id:levelsetUpwind.c 07/27/05 $*/
/* Program usage: qsub -pe mpich <procs> -q
                  hpc-mpich@hydra.lions.odu.edu levelset2d */

static char help[] = "Levelset equation in 2d.\n\
We solve the levelset problem in a 2D rectangular\n\
domain, using distributed arrays (DAs) to partition.\n\
the parallel grid. The command line options include:\n\
  -par <parameter>, where <parameter> indicates\n\
    problem's nonlinearity\n\n";
/*T
   Concepts: SNES^parallel
   Concepts: DA^using distributed arrays;
   Processors: n
T*/
/* ------------------------------------------------------------------
    Level set problem.  This problem is modeled by
    the partial differential equation

            du/dt + v*|grad(u)| = 0,

    This code was programmed by Pallop Huabsomboon based on ex5.c
  ------------------------------------------------------------------ */
/*
   Note: The outputs will be written into ASCII files (x??.m) in the
   directory OUTPUT. Make sure that you create the directory OUTPUT.
   Then a MATLAB routine will read the ASCII file and produce a
```

```
   contour plot. However, users can use a command line -contour to
   see the contour plot on x-window interface.
*/

#include "petscda.h"
#include "petscsnes.h"

typedef struct {
   DA           da;        /* distributed array data structure */
   PetscReal    dx,dt;
   PetscReal    current_time;
   Vec          phi_old;   /* phi at time n */
   int          iter;      /* iteration */
   PetscScalar  x_start, x_stop;
   int          N;
   PetscReal    CenterX,CenterY;
   PetscReal    radius;
   PetscTruth   RotateWind;
   int          model;             /* type of fire model */
   PetscReal    U;                 /* wind speed */
   PetscReal    epsilon0,epsilon1; /* fire parameter */
   PetscReal    alpha,beta;        /* fire parameter */
   PetscReal    c1;                /* fire parameter */
} AppCtx;

typedef struct {
   char txt[20];
} TextCtx;

extern int FormInitialCurve(DALocalInfo*,PetscScalar**,
                            PetscScalar**,AppCtx*);
extern int FormInitialGuess(AppCtx*,Vec);
extern int FormFunctionLocal(DALocalInfo*,PetscScalar**,
                             PetscScalar**, AppCtx*);
```

```
extern int FormText(TextCtx*,int);

#undef __FUNCT__
#define __FUNCT__ "main"
int main(int argc,char **argv)
{
  SNES              snes;           /* nonlinear solver */
  KSP               ksp;            /* KSP context */
  PC                pc;             /* PC context  */
  MPI_Comm          comm;           /* contour plot parameter */
  PetscDraw         draw;           /* contour plot parameter */
  Vec               x,r;            /* solution, residual vectors */
  Mat               A,J;            /* Jacobian matrix */
  AppCtx            user;           /* user-defined work context */
  int               its;            /* iterations for convergence */
  int               ierr;
  int               i,j,k;          /* indexes */
  int               N;
  int               xs,ys,xm,ym;
  int               max_t;                /* final time */
  Vec               u,v;
  PetscScalar       **phi;
  PetscScalar       grid_st, grid_sp;

  MatFDColoring     matfdcoloring = 0;
  ISColoring        iscoloring;
  PetscScalar       t;                   /* time */
  PetscReal         dx,dt;
  PetscReal         CenterX,CenterY,radius;
  PetscReal         aa,bb;
  PetscTruth        draw_contour;
  PetscTruth        write_output;
  PetscTruth        rotate_wind;
  PetscTruth        flag;
```

```
FILE              *fp;       /* file pointers */
int               model;     /* type of firespread model*/
PetscReal         U;                  /* wind speed */
PetscReal         epsilon0,epsilon1; /* fire parameter */
PetscReal         alpha;              /* fire parameter */
PetscReal         beta;               /* fire parameter */
PetscReal         c1;                 /* fire parameter */

PetscLogDouble    v1,v2,elapsed_time;
PetscReal         minute,second;
PetscViewer       view_out;
TextCtx           msgs;

PetscInitialize(&argc,&argv,(char *)0,help);
comm = PETSC_COMM_WORLD;
ierr = PetscGetTime(&v1);CHKERRQ(ierr);
ierr = SNESCreate(comm,&snes);CHKERRQ(ierr);

/*- -  Parameter setup - - - */
dx = 0.01;        dt = 0.02;
grid_st = 0;      grid_sp = 3;
CenterX = 1;      CenterY = 1.5; radius  = 0.5;
max_t = 40;
model = 2; /*
            Set model = 1 for Fendell-Wolff model,
                model = 2 for Fendell-Mallet model,
            Otherwise
                we will have an expanding circle with unit speed
           */
U = 5.0;
alpha = 0.1;
beta = 1.0;
epsilon0 = 0.1;
epsilon1 = 0.003;
```

```
c1 = 0.5;

/*- -   end setup  - - */

draw_contour = PETSC_FALSE;
write_output = PETSC_FALSE;
rotate_wind  = PETSC_FALSE;
ierr=PetscOptionsGetReal(PETSC_NULL,"-dx",&dx,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-dt",&dt,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-a",&radius,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-x_st",&grid_st,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-x_sp",&grid_sp,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetInt(PETSC_NULL,"-max_t",&max_t,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsHasName(PETSC_NULL,"-contour",&draw_contour);
     CHKERRQ(ierr);
ierr=PetscOptionsHasName(PETSC_NULL,"-output",&write_output);
     CHKERRQ(ierr);
ierr=PetscOptionsHasName(PETSC_NULL,"-rotate_wind",&rotate_wind);
     CHKERRQ(ierr);
ierr=PetscOptionsHasName(PETSC_NULL,"-snes_mf",&flag);
     CHKERRQ(ierr);

/**  Set up for the wind speed parameter **/
ierr=PetscOptionsGetInt(PETSC_NULL,"-model",&model,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-U",&U,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-ep0",&epsilon0,PETSC_NULL);
```

```
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-ep1",&epsilon1,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-alpha",&alpha,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-beta",&beta,PETSC_NULL);
     CHKERRQ(ierr);
ierr=PetscOptionsGetReal(PETSC_NULL,"-c1",&c1,PETSC_NULL);
     CHKERRQ(ierr);

N = (int)ceil(grid_sp/dx)+1;
user.N = N;  user.dx = dx;  user.dt = dt;
user.x_start = grid_st;  user.x_stop  = grid_sp;
user.CenterX = CenterX;  user.CenterY = CenterY;
user.radius  = radius;
user.model = model;
user.RotateWind = rotate_wind;
user.U = U;
user.epsilon0 = epsilon0;
user.epsilon1 = epsilon1;
user.alpha    = alpha;
user.beta     = beta;
user.c1       = c1;

ierr= PetscPrintf(comm,"Fire model %d :N = %d\n",user.model,N);
      CHKERRQ(ierr);
ierr=DACreate2d(comm,DA_NONPERIODIC,DA_STENCIL_STAR,
                  N,N,PETSC_DECIDE, PETSC_DECIDE,1,1,PETSC_NULL,
                  PETSC_NULL,&user.da);CHKERRQ(ierr);
ierr=DACreateGlobalVector(user.da,&x);CHKERRQ(ierr);
ierr=VecDuplicate(x,&r);CHKERRQ(ierr);
ierr=VecDuplicate(x,&u);CHKERRQ(ierr);
ierr=VecDuplicate(x,&v);CHKERRQ(ierr);
```

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - -
                Form initial curve
- - - - - - - - - - - - - - - - - - - - - - - - - - - - */
ierr=DASetLocalFunction(user.da,(DALocalFunction1)
                        FormInitialCurve);CHKERRQ(ierr);
ierr=SNESSetFunction(snes,r,SNESDAFormFunction,&user);
     CHKERRQ(ierr);
SNESComputeFunction(snes,v,u);
user.phi_old = u;
/* Write initial curve to an ASCII file (matlab format) */
if(write_output){
   ierr=PetscViewerASCIIOpen(comm,"OUTPUT/xstart.m",
                             &view_out);CHKERRQ(ierr);
   ierr=PetscViewerPushFormat(view_out,
                   PETSC_VIEWER_ASCII_MATLAB);CHKERRQ(ierr);
   ierr=PetscObjectSetName((PetscObject)u,"x");CHKERRQ(ierr);
   ierr=VecView(u,view_out); CHKERRQ(ierr);
}
/* Write some informations to a text file */
if(write_output){
   fp = fopen("OUTPUT/info.txt","w");
   ierr = PetscFPrintf(comm,fp,"%d\n",N);
   ierr = PetscFPrintf(comm,fp,"%d\n",max_t);
   ierr = PetscFPrintf(comm,fp,"%g\n",grid_st);
   ierr = PetscFPrintf(comm,fp,"%g\n",grid_sp);
   ierr = PetscFPrintf(comm,fp,"%g\n",user.dx);
   ierr = PetscFPrintf(comm,fp,"%g\n",user.dt);
   ierr = PetscFPrintf(comm,fp,"%d\n",user.model);
   if(rotate_wind){
      ierr=PetscFPrintf(comm,fp,"1\n"); CHKERRQ(ierr);
   }
   else{
      ierr=PetscFPrintf(comm,fp,"0\n"); CHKERRQ(ierr);
   }
```

```
   fclose(fp);
}

if(flag){
    ierr=PetscPrintf(comm,"Use Matrix-Free method!\n");
         CHKERRQ(ierr);
}
else{
    ierr=PetscPrintf(comm,"Use finite difference approx.!\n");
         CHKERRQ(ierr);
}
t = dt;
user.current_time = t;
for (k=1;k<=max_t;k++)
{
  user.iter = k;
  ierr = PetscPrintf(comm,"k = %d t = %g ",k,t); CHKERRQ(ierr);
  ierr = DASetLocalFunction(user.da,
              (DALocalFunction1)FormFunctionLocal);CHKERRQ(ierr);
  ierr = SNESSetFunction(snes,r,SNESDAFormFunction,&user);
         CHKERRQ(ierr);

  ierr = DAGetMatrix(user.da,MATMPIAIJ,&J);CHKERRQ(ierr);
  A    = J;

  /*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    Form a jacobian matrix:
        - With the petsc option -snes_mf, the Matrix-Free method
    will be used. Otherwise, use finite difference approximation.
   -- - - - -  - - - - - - - - - - - - - - - - - - - - - - - - */

  if(flag){
    ierr = SNESGetKSP(snes,&ksp); CHKERRQ(ierr);
    ierr = KSPGetPC(ksp,&pc); CHKERRQ(ierr);
```

```
  ierr = PCSetType(pc,PCNONE); CHKERRQ(ierr);
}
else{
  ierr=DAGetColoring(user.da,IS_COLORING_LOCAL,&iscoloring);
       CHKERRQ(ierr);
  ierr=MatFDColoringCreate(J,iscoloring,&matfdcoloring);
       CHKERRQ(ierr);
  ierr=ISColoringDestroy(iscoloring);CHKERRQ(ierr);
  ierr=MatFDColoringSetFunction(matfdcoloring,
                      (int (*)(void))SNESDAFormFunction,&user);
       CHKERRQ(ierr);
  ierr=MatFDColoringSetFromOptions(matfdcoloring);
       CHKERRQ(ierr);
  ierr=SNESSetJacobian(snes,A,J,SNESDefaultComputeJacobianColor,
                           matfdcoloring);
 CHKERRQ(ierr);
}
ierr = SNESSetFromOptions(snes);CHKERRQ(ierr);

if (k <= 1){
     /*-- After 1st time step, use the previous solution as
          the initial guess --*/
    ierr = FormInitialGuess(&user,x);CHKERRQ(ierr);
}

/* - - - - - - - - - - - - - - - - - - - - - - - - -
          Solve nonlinear problem
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
ierr=SNESSolve(snes, x);CHKERRQ(ierr);
ierr=SNESGetIterationNumber(snes,&its);
ierr=PetscPrintf(comm,"Number of Newton iterations");
     CHKERRQ(ierr);
ierr=PetscPrintf(comm," = %d\n",its);
     CHKERRQ(ierr);
```

```
  ierr=VecCopy(x,user.phi_old);CHKERRQ(ierr);
  t = t+dt;
  user.current_time = t;

  /* - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Write the solution to ASCII files (matlab format).
     - - - - - - - - - - - - - - - - - - - - - - - - - - - */

  if(write_output){
      FormText(&msgs,k); /* Form a new variable name */
      ierr=PetscViewerASCIIOpen(comm,msgs.txt,&view_out);
   CHKERRQ(ierr);
      ierr=PetscViewerPushFormat(view_out,
          PETSC_VIEWER_ASCII_MATLAB);CHKERRQ(ierr);
      ierr=PetscObjectSetName((PetscObject)x,"x");CHKERRQ(ierr);
      ierr=VecView(x,view_out); CHKERRQ(ierr);
  }

  if(draw_contour){
      ierr=PetscViewerDrawGetDraw(PETSC_VIEWER_DRAW_(comm),
                           0,&draw);CHKERRQ(ierr);
      ierr=PetscDrawSetDoubleBuffer(draw); CHKERRQ(ierr);
      ierr=VecView(x,PETSC_VIEWER_DRAW_(comm)); CHKERRQ(ierr);
  }
} /** End k loop **/

/* - - - - - - - - - - - - - - - - - - -
            Free work space
 - - - - - - - - - - - - - - - - - - -- - */
if(write_output){
   ierr = PetscViewerDestroy(view_out); CHKERRQ(ierr);
}

if (A != J) {
```

```
    ierr = MatDestroy(A);CHKERRQ(ierr);
  }
  ierr = MatDestroy(J);CHKERRQ(ierr);
  if (matfdcoloring) {
    ierr = MatFDColoringDestroy(matfdcoloring);CHKERRQ(ierr);
  }

  ierr = VecDestroy(x);CHKERRQ(ierr);
  ierr = VecDestroy(r);CHKERRQ(ierr);
  ierr = VecDestroy(u);CHKERRQ(ierr);

  ierr = SNESDestroy(snes);CHKERRQ(ierr);
  ierr = DADestroy(user.da);CHKERRQ(ierr);

  ierr = PetscGetTime(&v2);CHKERRQ(ierr);
  elapsed_time = v2 - v1;
  minute = floor(elapsed_time/60);
  second = (int)elapsed_time%60;
  ierr = PetscPrintf(comm,"Total time used in computation = ");
         CHKERRQ(ierr);
  ierr = PetscPrintf(comm,"%g min. %g sec.\n",minute,second);
         CHKERRQ(ierr);

  ierr = PetscFinalize();CHKERRQ(ierr);
  PetscFunctionReturn(0);
}

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
#undef __FUNCT__
#define __FUNCT__ "FindInitialCurve"
/*
   FormInitialCurve - Forms initial curve.
*/
int FormInitialCurve(DALocalInfo *info,PetscScalar **X,
```

```
                    PetscScalar **phi, AppCtx *user)
{
  int             i,j,xs,ys,xm,ym;
  PetscReal       dx,dy,dt;
  PetscReal       aa,bb;
  PetscReal       xc,yc;
  PetscReal       radius;

  PetscFunctionBegin;
  dx = user->dx;       dt = user->dt; dy = dx;
  xc = user->CenterX; yc = user->CenterY;
  radius = user->radius;

  for (j=info->ys; j<info->ys+info->ym; j++) {
    aa = user->x_start+j*dx;
    for (i=info->xs; i<info->xs+info->xm; i++) {
       bb = user->x_start + i*dx;
       phi[j][i] = sqrt(pow(aa-yc,2.0)+pow(bb-xc,2.0))-radius;
    }
  }
  PetscFunctionReturn(0);
}

/* ------------------------------------------------------------*/
#undef __FUNCT__
#define __FUNCT__ "FormInitialGuess"
/*
   FormInitialGuess - Forms initial approximation.
*/
int FormInitialGuess(AppCtx *user,Vec X)
{
  int         i,j,Mx,My,ierr,xs,ys,xm,ym;
  PetscReal   temp,hx,hy;
  PetscScalar **x;
```

```
  PetscFunctionBegin;
  ierr = DAGetInfo(user->da,PETSC_IGNORE,&Mx,&My,PETSC_IGNORE,
                   PETSC_IGNORE,PETSC_IGNORE,PETSC_IGNORE,
                   PETSC_IGNORE,PETSC_IGNORE,PETSC_IGNORE,
                   PETSC_IGNORE); CHKERRQ(ierr);
  ierr = DAVecGetArray(user->da,X,(void**)&x);CHKERRQ(ierr);
  ierr = DAGetCorners(user->da,&xs,&ys,PETSC_NULL,
                      &xm,&ym,PETSC_NULL);CHKERRQ(ierr);
  for (j=ys; j<ys+ym; j++) {
    for (i=xs; i<xs+xm; i++) {
        x[j][i] = 0.0;
    }
  }
  ierr = DAVecRestoreArray(user->da,X,(void**)&x);CHKERRQ(ierr);
  PetscFunctionReturn(0);
}

/* ------------------------------------------------------------------- */
#undef __FUNCT__
#define __FUNCT__ "FormFunctionLocal"
/*
   FormFunctionLocal - Evaluates nonlinear function, F(x).
*/
int FormFunctionLocal(DALocalInfo *info,PetscScalar **phi,
                      PetscScalar **f, AppCtx *user)
{
  int          ierr,i,j;
  PetscReal    dx,dy;
  PetscReal    speed;           /* Fire front speed */
  PetscReal    temp;            /* Temporary valiable */
  PetscReal    Dxplus, Dxminus; /* Forward, backward diff. in x */
  PetscReal    Dyplus, Dyminus; /* Forward, backward diff. in y */
  PetscScalar  **Phi_Old;
```

```
PetscReal    pi = 22/7;
PetscReal    theta;  /* Angle between the wind direction and
                        unit normal of the curve */
PetscReal    sintheta,costheta; /* sin(theta), cos(theta) */
PetscReal    sin2,cos2;   /* sin(theta)^2, cos(theta)^2 */
PetscReal    gamma;       /* angle-for the case rotating wind */
PetscReal    alpha, beta, c1, G; /* Fire parameters */
PetscReal    epsilon0, epsilon1; /* Fire parameter */
PetscReal    U;                  /* Wind Speed */
PetscReal    U1, U2;             /* Wind direction (U1,U2) */
PetscReal    norm, NormalX, NormalY;
PetscReal    dot_product;
PetscReal    Dx,Dy;    /* Derivertive of Phi wrt. x and y*/
PetscReal    CFL;

PetscFunctionBegin;
speed = 1.0;
dx = user->dx; dy = dx;
epsilon0 = user->epsilon0;  epsilon1 = user->epsilon1;
alpha    = user->alpha;     beta     = user->beta;
c1       = user->c1;        U        = user->U;

U1 = 1; U2 = 0;
ierr = DAVecGetArray(user->da,user->phi_old,(void**)&Phi_Old);
       CHKERRQ(ierr);
/*
   Compute function over the locally owned part of the grid
*/
for (j=info->ys; j<info->ys+info->ym; j++) {
  for (i=info->xs; i<info->xs+info->xm; i++) {
    if (i==0)                          /* Bottom boundary. */
        Dxminus = (phi[j][i]-Phi_Old[j][i])/dx;
    else
        Dxminus = (phi[j][i]-phi[j][i-1])/dx;
```

```
if (j==0)                          /* Left boundary. */
    Dyminus = (phi[j][i]-Phi_Old[j][i])/dy;
else
    Dyminus = (phi[j][i]-phi[j-1][i])/dy;

if (i == info->mx-1)               /* Top boundary. */
    Dxplus = (Phi_Old[j][i]-phi[j][i])/dx;
else
    Dxplus = (phi[j][i+1]-phi[j][i])/dx;

if (j == info->my-1)               /* Right boundary. */
    Dyplus = (Phi_Old[j][i]-phi[j][i])/dy;
else
    Dyplus = (phi[j+1][i]-phi[j][i])/dy;

/*- - - - - - - - - - - - - - - - - -
     Compute a speed function F
- - - - - - - - - - - - - - - - - - - -*/
Dx =  0.5*(Dxplus+Dxminus);
Dy =  0.5*(Dyplus+Dyminus);
norm = sqrt(Dx*Dx+Dy*Dy);

if( norm != 0.0){
    NormalX =  Dx/norm; NormalY =  Dy/norm;
}
else{
    NormalX = 0.0;  NormalY = 0.0;
}
if (user->RotateWind){
   gamma = (100*user->current_time/2)*pi/180;
   U1 = cos(gamma); U2 = sin(gamma);
}
dot_product = NormalX*U1+NormalY*U2;
```

```
theta = acos(dot_product/sqrt(U1*U1+U2*U2));
costheta = cos(theta);    sintheta = sin(theta);
cos2 = costheta*costheta; sin2 = sintheta*sintheta;
if(user->model == 1){
   if(costheta > 0)
         speed = epsilon0+c1*sqrt(U)*pow(costheta,1.5)
                +alpha*U*sin2*exp(-beta*U*sin2);
   else{
         G = epsilon0*cos2*exp(-epsilon1*U*cos2);
         speed = epsilon0*sin2
                + alpha*U*sin2*exp(-beta*U*sin2)+G;
   }
}
else if(user->model == 2){
      if(costheta > 0)
            speed = epsilon0+c1*sqrt(U*costheta)*costheta;
      else{
            CFL = 0.1;
            speed = epsilon0*(CFL + (1-CFL)*fabs(sintheta));
      }
}
if (speed > 0){
  temp = PetscSqr(PetscMax(Dxminus,0))
           + PetscSqr(PetscMin(Dxplus,0));
  temp += PetscSqr(PetscMax(Dyminus,0))
           + PetscSqr(PetscMin(Dyplus,0));
}
else{
  temp = PetscSqr(PetscMin(Dxminus,0))
         + PetscSqr(PetscMax(Dxplus,0));
  temp += PetscSqr(PetscMin(Dyminus,0))
 + PetscSqr(PetscMax(Dyplus,0));
}
```

```
      f[j][i] = phi[j][i]-Phi_Old[j][i]+user->dt*speed*sqrt(temp);
    }
  }
  ierr = PetscLogFlops(11*info->ym*info->xm);CHKERRQ(ierr);
  ierr = DAVecRestoreArray(user->da,user->phi_old,(void**)&Phi_Old);
         CHKERRQ(ierr);
  PetscFunctionReturn(0);
}

/* ------------------------------------------------------------ */
#undef __FUNCT__
#define __FUNCT__ "FormText"
/*
   Form a text and then the text as a MATLAB variable. The text
   will have the form OUTPUT/x??.m where "?" is some number.
*/
int FormText(TextCtx *user,int value)
{
   char         number[11] = "0123456789";
   char         txt2[20];
   int          k;
   int          length = 1;
   int          temp, div = 1;
   PetscTruth   flag = PETSC_TRUE;

   user->txt[0] = 'O'; user->txt[1] = 'U'; user->txt[2] = 'T';
   user->txt[3] = 'P'; user->txt[4] = 'U'; user->txt[5] = 'T';
   user->txt[6] = '/'; user->txt[7] = 'x'; user->txt[8] = '\0';

   /* Find length of value */
   do{
       div*=10;
       if ((int) ( (double)value / (double)(div)) ==0){
            flag = PETSC_FALSE;
```

```
        }
        length +=1;
    }while(flag);
    length -= 1;

    for (k = length; k>=1; k--){
         temp = value % 10;     /* Modulo to get the last number */
         txt2[k-1] = number[temp]; /* Add Number to the String */
         value = (value - temp)/10; /* Substract the last number
                                       from the Ingeter */
         temp *= 10 ;
    }
    txt2[length] = '\0';

    strcat(user->txt,txt2);
    strcat(user->txt,".m");

    PetscFunctionReturn(0);
}
```

A.2 PETSC ROUTINE FOR FINDING THE DISSIPATION COEFFICIENTS

The following PETSc routine contains a function evaluation of dissipation coefficients in the Lax-Friedrichs scheme.

```
int FindAlpha(DALocalInfo *info,PetscScalar **phi, PetscScalar **Vf,
              AppCtx *user)
{
  int          i,j;
  PetscReal    Dxplus, Dxminus; /* Forward, backward diff. in x */
  PetscReal    Dyplus, Dyminus; /* Forward, backward diff. in y */
  PetscReal    dx,dy;
  PetscReal    a_x_max, a_x_min, a_y_max, a_y_min;
  PetscReal    max, min;
```

```
PetscReal    phi_x, phi_y;
PetscReal    alpha_x, alpha_y, delta_x, delta_y;
int          N;

PetscFunctionBegin;
dx = user->dx; dy = dx;
a_x_max = -1e+16; a_x_min = 1e+16;
for (j=info->ys; j<info->ys+info->ym; j++){
  for (i=info->xs; i<info->xs+info->xm; i++){
    if (i==0)
        Dxminus = (phi[j][i+1]-phi[j][i])/dx;
    else
        Dxminus = (phi[j][i]-phi[j][i-1])/dx;
    if (j==0)
        Dyminus = (phi[j+1][i]-phi[j][i])/dy;
    else
        Dyminus = (phi[j][i]-phi[j-1][i])/dy;
    if (i == info->mx-1)
        Dxplus = (phi[j][i]-phi[j][i-1])/dx;
    else
        Dxplus = (phi[j][i+1]-phi[j][i])/dx;
    if (j == info->my-1)
        Dyplus = (phi[j][i]-phi[j-1][i])/dy;
    else
        Dyplus = (phi[j+1][i]-phi[j][i])/dy;

    max = PetscMax(Dxplus,Dxminus);
    if(max > a_x_max)
        a_x_max = max;
    min = PetscMin(Dxplus,Dxminus);
    if(min < a_x_min)
        a_x_min = min;

    max = PetscMax(Dyplus,Dyminus);
```

```
      if(max > a_y_max)
          a_y_max = max;
      min = PetscMin(Dyplus,Dyminus);
      if(min < a_y_min)
          a_y_min = min;
    }
  }
  N = user->N;
  alpha_x = -1e+10;  alpha_y = -1e+10;
  delta_x = (a_x_max-a_x_min)/N;
  delta_y = (a_y_max-a_y_min)/N;
  for(i = 0; i < N; i++){
     phi_x = a_x_min + i*delta_x;
     for(j = 0; j < N; j++){
       phi_y = a_y_min +j*delta_y;
       max = fabs(phi_x/sqrt(phi_x*phi_x+phi_y*phi_y));
       if(max > alpha_x)
           alpha_x = max;
       max = fabs(phi_y/sqrt(phi_x*phi_x+phi_y*phi_y));
       if(max > alpha_y)
           alpha_y = max;
     }
  }
  user->alpha_x = alpha_x;  user->alpha_y = alpha_y;
  PetscFunctionReturn(0);
}
```

A.3 SET UP FOR TWO-CIRCLE FIRE WITH AN UNBURNT ISLAND

The following PETSc routine contains an initial curve setup for two-circle fire with an unburnt island shown in Figure 25.

```
int FormInitialCurve(DALocalInfo *info,PetscScalar **X,
                     PetscScalar **phi, AppCtx *user)
```

```
{
  int             i,j,xs,ys,xm,ym;
  PetscReal       dx,dy,dt;
  PetscReal       aa,bb;
  PetscReal       first_cir,second_cir,third_cir;
  PetscReal       xc1,yc1,radius1;
  PetscReal       xc2,yc2,radius2;
  PetscReal       xc3,yc3,radius3;
  PetscReal       distance;

  PetscFunctionBegin;
  dx = user->dx;  dt = user->dt; dy = dx;
  xc1 = user->CenterX1; yc1 = user->CenterY1;
  xc2 = user->CenterX2; yc2 = user->CenterY2;
  xc3 = user->CenterX3; yc3 = user->CenterY3;
  radius1 = user->radius1; radius2 = user->radius2;
  radius3 = user->radius3;
  for (j=info->ys; j<info->ys+info->ym; j++){
    aa = user->x_start+j*dx;
    for (i=info->xs; i<info->xs+info->xm; i++){
       bb = user->x_start + i*dx;
       first_cir  = sqrt(pow(aa-yc1,2)+pow(bb-xc1,2.0))-radius1;
       second_cir = sqrt(pow(aa-yc2,2)+pow(bb-xc2,2.0))-radius2;
       third_cir = -1*(sqrt(pow(aa-yc3,2)+pow(bb-xc3,2.0))-radius3);
       distance = PetscMin(first_cir,second_cir);
       if (distance < 0)
           phi[j][i] = PetscMax(distance,third_cir);
       else
           phi[j][i] = distance;
    }
  }
  PetscFunctionReturn(0);
}
```

APPENDIX B

MATLAB ROUTINE FOR DISPLAYING THE RESULTS

For each step k ($k = 0, 1, \ldots$), the PETSc main routine will create an output file named $xk.m$. The following MATLAB routine (print_curve.m) will read the file $xk.m$ and then show a contour plot of the zero level of the level set function.

```
% print_curve.m
%  -- Read ASCII files 'x*.m' and then plot contour

% Open an information file
[fp,msg]  = fopen('info.txt','r');
if fp == -1
    disp(msg)
end
f = fscanf(fp,'%f');
fclose(fp);

Nx = f(1);   max_t = f(2);  Xbegin = f(3); Xend = f(4);
dx = f(5);   dt = f(6);     model = f(7);  rotate_wind = f(8);
fprintf('Nx = %g\n',Nx);
t = 0;
for k = 0:max_t
    if k == 0
        txt = 'xstart';
        fprintf('k = %g <intial curve> t = %g \n',k,t);
    else
        txt = ['x',num2str(k)];
        fprintf('k = %g  t = %g \n',k,t);
    end;
    eval(txt);
    figure(1);
    CI = contour(reshape(x,Nx,Nx)',0,'b');
    N = length(CI);
```

```
    C = CI(:,2:N)*dx-dx;
    figure(2);
    hold on;
    if (mod(k,2) == 0)
      plot(C(1,:),C(2,:));
      % Change viewing window
      hold on; plot(0,0,'w'); plot(3,3,'w'); hold off;
    end;
    t = t + dt;
end;
```

VITA

PALLOP HUABSOMBOON

Department of Computational and Applied Mathematics
Old Dominion University
Norfolk, VA 23529

PREVIOUS DEGREES:
B.S. Mathematics, April 1995, Mahidol University
M.S. Mathematics, December 2000, Oregon State University

EMPLOYMENT:
Department of Mathematics and Statistics
Mahidol University
Bangkok, Thailand

Typeset using LaTeX.