RANDOM WALK COMPUTATIONS OF DIFFUSIVE FIELDS

A Thesis

by

GREGORY SCOT LINDSTROM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 1993

Major Subject: Mathematics

ABSTRACT

Random Walk Computations of Diffusive Fields. (May 1993)

Gregory Scot Lindstrom, B.S., Harding University

Chair of Advisory Committee: Dr. Prabir Daripa

A numerical study of three problems is carried out using the gradient random walk method. These problems include the heat equation, Fitzhugh-Nagumo equation, and Burgers' equation. Each problem illustrates various aspects of the operation of the numerical method. The heat equation with no reaction term illustrates the numerical algorithm, the Fitzhugh-Nagumo equation is a system with an explicit reaction term, Burgers' equation has an advection term.

The gradient random walk numerical method is well suited for the diffusion problem due to the connection between Gaussian distributions and the kernel of the heat equation. The numerical results compare well with known analytical solutions. Burgers' equation is studied to examine any effects the advection term has on the results of the numerical method.

RANDOM WALK COMPUTATIONS OF DIFFUSIVE FIELDS

A Thesis

by

GREGORY SCOT LINDSTROM

Approved as to style and content by:

May 1993

for Rachel

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The author realizes acknowledging individuals introduces the probability of omitting some who merit thanks; however, claiming all credit for the completion of this work would constitute a greater wrong.

I would like to thank the members of my graduate committee for their guidance. Bart Childs has offered time above and beyond the call of duty for a committee member, teaching me the ways of TEX and LaTEX. Neil Burleson spent hours answering every question I had concerning the operation of the Vax computer system at Texas A&M University.

An invaluable debt is owed to Harding University, in particular to: Dean Priest, Don England, James Mackey and Lambert Murray, and Linda Thompson, who have shown infinite patience with me while I finished this thesis on their time. Everything I have asked for has been supplied. I consider it a pleasure to work for an organization which places such a high value on the personal development of its employees.

For taking the time to encourage a frustrated freshman eight years ago, Charles Pittman has contributed as much as any to the completion of this work.

I must also thank my mother, Betty Lindstrom, for always loving me. Likewise, I acknowledge the influence of my father, Albin Ross Lindstrom, who passed away before seeing me attend college. The memory of a conversation in which he informed me that "*only a 'total fruitcake' likes partial derivatives*," has come to mind often during the past few years. Even with his faults (or perhaps *because* of them), I consider my father to be one of the finest men I have ever known.

My wife has been my strength over the past two years. From proofreading this text, to acting interested in partial differential equations (yes, honey, I *knew*), she has always supported me. If justice were to be served, her name would be printed on the diploma as well as mine.

And finally, for keeping me awake nights, for spitting up on me, for scattering my notes in the name of fun and showing me what life is *really* about, I thank the (currently) 26 pounds of pure love God has given me in the form of a daughter. Rachel, though the people named above helped me finish this degree, you alone are the reason I have endured. There is no other possible way I would have kept trying other than the hope of your future. I love you, little girl.

1 INTRODUCTION

The computation of diffusive fields is of great interest to the scientific community. These fields describe the flow of heat in various media [8], aspects of the clotting of blood [16], the propagation of signals in nerve tissue [23], and other applications. There has been substantial research in the computation of diffusive fields using finite difference [2], finite element [1], and Monte Carlo [4] methods using serial and parallel computers [25]. Random walk methods have been studied and used for many years [8] along with the gradient random walk family of numerical methods, but have not been generally used due to the computational cost of the algorithm. Particle methods, of which the gradient random walk is a subset, are well suited for systems with infinite (or semi-infinite) domains as it is impossible to set a finite grid to cover such domains. They also are appealing for systems which develop steep gradients, or shocks, because the particles will tend to concentrate near the areas where the shock is located [27], optimizing computing complexity.

Though the idea motivating the gradient random walk method was first envisioned as a three dimensional model, the first working models constructed were of one-dimensional space (excluding time, of course). Only recently has the theory been extended to two spatial dimensions [17] for a limited number of systems, and though many different types of systems have been successfully modeled with this method [5, 28, 31], most have coupled the gradient random walk with some other method, usually to aid in modeling the reaction term of the system. Modeling the reaction term has required imposition of a grid to aid in the computation[11], a result which nullifies the implementation of a grid-free method.

This thesis explores the implementation of the gradient random walk numerical method as a totally statistical numerical method. Systems with reaction terms require computation of a "reaction statistic" in order to carry out the simulation. If the explicit form of this statistic is not available, the value must be computed from the

This thesis follows the style of the *Journal of Computational Physics.*

numerical data during the run. This (normally) involves introducing a grid over the domain of the system and using another numerical method to derive the value. Though there is nothing "wrong" in such an approach, one of the attractive features of particle methods is their ability to handle infinite domains.

The idea behind the gradient random walk method is reviewed in section 2, which covers the connection between the Gaussian distribution and the heat equation, boundary conditions and the role played by any reaction term in the system, along with concerns dealing with random numbers generated on a computer and the choice of a data structure for the program.

In section 3, the one-dimensional heat equation is numerically solved using the gradient random walk method in order to illustrate the various conditions which may be handled with the method. These conditions include semi-infinite and infinite domains, Dirichlet and Neumann boundary conditions on a finite domain, along with continuous and discontinuous initial data. All results are shown in the form of x-y graphs with the numerical results plotted against known analytic solutions. In the case of the finite domain, the variance between the numerical and exact solutions is computed for various time steps and number of computational elements. A table suggesting that the accuracy of the method is improved as either Δt is reduced or as the number of elements is increased is presented

Section 4 considers the Fitzhugh-Nagumo equation, a simple system with an explicit reaction term which is solved using the gradient random walk method. The system develops a traveling wave solution on an infinite domain, and is set up with continuous, piece-wise continuous, and discontinuous initial data.

Burgers' equation, considered in section 5, represents a challenge because the reaction term requires the values for u, u_x and u_{xx}. Two numerical approaches to computing these values are discussed.

The final section draws conclusions concerning the use of the gradient random walk method as applied to these three systems. Along with this discussion are suggestions for further research which may lead to overcoming some of the problems discovered when applying the gradient random walk method to various systems.

All simulations were run on a Sun SPARCstation IPC computer running SunOS 4.1.2 (UNIX). The program is written in the C programming language (see appendix B for source code) with all graphical work performed under "PV-Wave" software from Precision Visuals, Inc.

2 THE GRADIENT RANDOM WALK METHOD

The gradient random walk method was first introduced by Chorin as a means of modeling combustion reactions [8]. We consider the following system:

$$\begin{cases} u_t = \alpha u_{xx} + f(u), & a \le x \le b, \ t > 0, \\ u(t,a) = u_a(t), \\ u(t,b) = u_b(t), \\ u(0,x) = u_0(x), \end{cases} \tag{1}$$

where a and b are constants (or $\pm\infty$, respectively), $u_a(t)$ and $u_b(t)$ are functions in x (which may or may not be constant) and α is the constant of diffusivity. For clarity in the explanation of the method, we choose $a = 0.0$, $b = 1.0$ and $u_0(x)$ as the following shifted Heaviside unit function:

$$u_0(x-1) = H(x-1) = \begin{cases} 0, & x < 1 \\ 1, & x > 1. \end{cases} \tag{2}$$

To model the gradient of this system, it is first differentiated with respect to x, then the u_x is replaced with v giving the following system:

$$\begin{cases} v_t = \alpha v_{xx} + g(u), & 0 \le x \le 1, \ t > 0 \\ v(t,0) = v_0(x), \\ v(t,1) = v_1(t), \\ v(0,x) = v_0(t), \end{cases} \tag{3}$$

where $g(u) = f'(u)u_x$ and $v_0(x) = u_0'(x)$. The initial data in system (1) is modeled with a step function, then differentiated with respect to x. This process produces a single δ-jump, located at $x = 1$ in system (3). The height of the δ-jump reflects the the range of the initial data $u_0(x)$ in (1), namely 1 unit. The range of the δ-jump will be divided into N_0 parts, each representing a portion of the total jump [18], but not necessarily all the same value [20]. Though this is a particle method, current convention is to refer to each of the computational elements as "globs" in order to distinguish them as representing the gradient of the original system, not the actual system.

The initial position of the globs is determined by looking at the function $v_0(x)$. In the above system (3), a single δ-jump is located at $x = 1.0$ so all of the globs are initially located at this position.

Each step through the main loop of the method will be carried out in two phases. Initially, each of the computational elements will undergo a random walk step drawn from a Gaussian distribution with a given variance (see below). Then each glob will undergo a creation/destruction algorithm based on a differential equation derived from the reaction term of the original system [10]. This allows the code to be divided into subroutines for each phase.

2.1 Random Walk Process

If a large number of particles are initially at a single location, then each particle is allowed to move a particular distance in either the positive or negative direction drawn from a Gaussian distribution of mean zero and variance σ^2, then after the first of these steps the probability of a particle being located in a neighborhood dx around a specified point x is given by

$$P_1(x,\sigma)\ dx = \frac{\exp(-x^2/2\sigma^2)}{\sqrt{2\pi}\sigma}\ dx. \tag{4}$$

If each particle is then to undergo another similar step, the probability of locating a particle in a particular neighborhood is

$$P_2(x,\sigma)\ dx = \ dx \int_{-\infty}^{\infty} \frac{\exp(-\nu^2/2\sigma^2)\exp(-(x-\nu)^2/2\sigma^2)}{2\pi\sigma^2} d\nu. \tag{5}$$

This integral may be evaluated [15] to obtain the probability of a single particle reaching a prescribed destination x given that it had reached a destination ν on the first step. The evaluated form of equation (5) is

$$P_2(x,\sigma)dx = \frac{\exp(-x^2/4\sigma^2)}{\sqrt{4\pi}\sigma}\ dx \tag{6}$$

which through induction yields

$$P_k(x,\sigma) = \frac{\exp(-x^2/2k\sigma^2)}{\sqrt{2k\pi}\sigma}. \tag{7}$$

The above process demonstrates that the sum of Gaussian random variables with mean zero is another Gaussian variable with variance which is the sum of the original variances[18]. If $\sigma^2 = 2\alpha\Delta t$, where Δt is a time step and $t = k\Delta t$ $(k = 1, 2, 3, \ldots, T)$, then

$$P(x,t) = (4\pi\alpha t)^{-1/2}\exp(-x^2/4\alpha t), \tag{8}$$

which is the kernel of the diffusion equation:

$$u_t = \alpha u_{xx} \tag{9}$$

This illustrates that the heat equation (9) may be numerically modeled using the gradient random walk method.

Using a random number generator, one may displace the globs using values drawn from a Gaussian distribution with mean 0 and variance $2\alpha\Delta t$ as the size of the displacement (step). If, at each time step of the numerical simulation, every particle undergoes a random walk , the resulting distribution of the computational elements, when integrated–as the method models the gradient of the original system–will return the solution of the original equation [12]. An exact solution assumes an infinite number of particles to begin with which, of course, is not possible with the computer. The limitations caused by using a finite number of computational elements will be addressed in section 3.

2.2 Creation and Destruction of Computational Elements

Once the globs have completed the random walk portion of the method, the second phase, effect of the reaction term, must be considered [9]. Suppose there is a region S_i where u is a constant. Then solve the ordinary differential equation

$$\frac{dw}{dt} = g(w), \quad w(0) = u_i. \tag{10}$$

Define $\psi_i = w(k) - u_i$, the change in u due to the reaction term in S_i. Then u_i becomes $u_i + \psi_i$ in S_i. After completing this same routine for all S_i at each time step, break Δu_i into smaller pieces if $|\Delta u_i| > \Delta u_{max}$ and delete the corresponding glob if it becomes sufficiently small ($|\Delta u_{min}| < \epsilon$, for some ϵ). Doing so concentrates the globs where the gradient is steepest, and removing the computational elements where the gradient is very small [13].

In order to carry out this process on the computer, a reaction statistic will be computed for each glob, with the magnitude of the statistic representing the probability per time step that the glob will either be duplicated or destroyed, according to the sign of the statistic [10]. The function to be used for computing the reaction statistic is derived from the reaction term of the original system [19]. Recall the gradient random walk method deals with the derivative of the original system which transforms the original reaction term $f(u)$ into $f'(u)u_x$. The function $f'(u)$ provides the formula needed to compute the reaction statistic (though it will be shown the actual computation of this statistic may present a substantial challenge).

In order to compute the reaction statistic, the glob list must be sorted; then $u(t, x)$ must be computed at each time level. If the reaction statistic formula calls for the values of u_x or u_{xx}, then these values must be computed as well. The actual routine for the creation/destruction of the globs requires one pass through the glob list per time level. At each pass, a statistic, z, for each glob is computed which represents the probability of the glob being replicated or deleted from the glob list.

2.3 Boundary Conditions

If the equation being simulated is defined on a finite (or semi-infinite) domain, then the effect of the boundary condition(s) must be considered. The boundaries will only affect those globs which travel outside the domain during the random walk section of the algorithm.

When the boundary condition is Dirichlet, that is the value of the function at the endpoint is known and is constant, the globs traveling across the end of the domain are to be "reflected" into the domain by precisely the distance they traveled out. If two identical systems are envisioned sharing a common boundary with a constant value, a glob traveling out of the first system would be passed by a similar glob passing out of the second, hence keeping the total number of globs in each system constant. This algorithm known as symmetric reflection and is used to handle all Dirichlet boundary conditions[18].

If the boundary is of the von Neumann variety, that is the derivative of the function is known at the endpoint, the globs traveling out of the system will be reflected back into the domain as above, but in this case the value of the glob will be changed from dy to $-dy$. Von Neumann systems may envisioned similar to Dirichlet except that a glob passing out one one system will be passed by a similar glob passing out of the second system with the opposite computational value. This is known as anti-symmetric reflection.

2.4 Numerical Integration

Since the gradient random walk method deals with the gradient of the system being modeled, the final process in computing the value of u for any time level is integrating the glob list. This is a simple process of summing all of the glob values to the left of the position being computed. In practice, this is a simple loop to execute in the program which starts at the left most glob and computes the sum of all of the glob values preceeding any given location, printing out the location and value for each of the globs in the list. If the $i^{th}\delta$-jump is represented by δ^i_j, where $t = j\Delta t$, then the numerical integration may be performed as

$$u(t, x_n) = \sum_{i|x_i \leq x_n} \delta^i_j. \qquad (11)$$

In order to facilitate the plotting of the final data, the domain is divided into an arbitrary number of "bins." The value for a particular bin is the sum of the globs which fall within its domain, with the position of the bin being the center value. This has the practical benefit of allowing the plotting device to print out a fraction of the computational elements used (necessary when N_0 is thousands of globs) and the cosmetic benefit of smoothing the statistical variation of the final curve–a feature which will be exploited when the derivative needs to be numerically computed.

2.5 Data Structure

Though not a theoretical aspect of the method, the data structure plays a crucial role in the practical application of any algorithm. In order to accommodate the fluctuating number of computational elements during each run, a doubly linked list is used with each element of the list holding the position and value of the element.

Sorting of the elements plays a major role in the time complexity of the method. The quicksort algorithm[33] has been adapted for the linked data structure. It is, on most machines, on average, for large N the fastest known sorting method [33] if the initial data is in random order. Other methods, such as a Heapsort or a modified insertion sort might prove to be quicker in the application the gradient random walk method because the data is in "almost" ascending order.

2.6 Random Numbers

The gradient random walk method is a statistical method to simulate the reaction/diffusion equations and is therefore dependent on a great supply of random numbers for the steps during the random walk process and for the reaction statistic critical values during the creation and destruction of the globs. As true random numbers can not (yet) be generated on a computer [33], care must be taken to insure a sufficiently random supply of numbers can be produced for each simulation. All of the runs for this study have been performed on the UNIX operating system. UNIX provides a function, *drand48()*, a 48-bit generation routine which has proved to be quite reliable. In order to provide a "new" stream of random numbers for each simulation, the function is seeded with a number which itself is a function of the date, time and process identification number (PID) at the time the run is initiated. In order to produce identical runs for testing purposes, the seed may be statically set in order to provide the exact same stream of numbers.

To produce the Gaussian distribution required for the generation of the step sizes, the Box–Muller[29] method has been used to create a distribution with mean zero and variance $2\alpha\Delta t$. If r_1 and r_2 are random numbers from a uniform distribution between 0 and 1 with $r_2 \neq 0$, then

$$x_i = -4\alpha\Delta t \log(r_2)\cos(2\pi r_1) \tag{12}$$

will generate a normal distribution with mean zero and variance $2\alpha\Delta t$.

3 THE HEAT EQUATION

The first case to consider with the gradient random walk will be the heat equation, which represents diffusive fields where the factor of proportionality, α, between the flux and concentration gradient is a scalar constant [18]. Though much research has been conducted on both the analytic [6, 14, 26] and numerical [3, 30, 32] solutions to this equation, study is warranted here, not so much as to shed new light on the heat equation, but rather to insure the gradient random walk method can simulate a case where the exact solution is known. Once the reliability of the numerical method has been established, more complex systems may be explored with some assurance that the numerical results are in fact a good approximation of the exact solution. Such confidence is vital with systems where the analytical solutions are not available.

The heat equation in one dimension with no reaction term may be expressed as

$$\begin{cases} u_t = \alpha u_{xx}, & a \leq x \leq b, \quad t > 0, \\ u(t,a) = u_a(t), \\ u(t,b) = u_b(t), \\ u(0,x) = u_0(x), \end{cases} \tag{13}$$

where a and b are fixed endpoints of the domain (which may or may not be $\pm\infty$), α is the diffusivity constant, u_a and u_b are the boundary values (which may or may not be constant) and $u_0(x)$ is the initial data for the system.

3.1 Semi-Infinite Domain

Choosing $a = -\infty$ and $b = 0.0$ allows a semi-infinite domain to be simulated. This illustrates an attractive feature of the gradient random walk method (and *all* particle methods) in that it is impossible to construct any static grid over such a domain [16]. If the initial data is defined as $u_0(x) = 0.0$, with the boundary condition $u_b(t) = 1.0$, a system is constructed for which a known solution exists. This may be considered a semi-infinite rod initially at 0° units until $t = 0.0$, when the temperature at the right end of the rod is instantly raised to 1° unit and held constant. The heat profile would be expected to progress down the rod (to the left), never reaching a steady state profile due to the infinite length of the domain toward $-\infty$.

In order to prepare the system for simulation using the GRW method, the gradient of the initial data must be computed. Since

$$u_0(x) = \begin{cases} 0.0, & x < 0.0, \\ 1.0, & x \geq 0.0, \end{cases} \tag{14}$$

then the gradient is a single δ-jump, located at $x = 0$. The height of the jump is 1 unit, corresponding to the range of the initial data, $u_0(x)$.

The number of globs to start with is a matter of balancing the degree of accuracy desired against the computational time cost and physical limitations of the machine and situation (there may be a limitation on the amount of time allowed before a solution is required). In the case of a semi-infinite domain, as the heat profile spreads, the particles will eventually separate so far as to hinder the accuracy. For this case, however, the amount of time spent computing will be kept finite—setting $N_0 = 1000$ will be sufficient. The reflection on the right end of the domain will be set as symmetric, since the function value of the field at the boundaries is known and constant.

Figure 1 shows the computed solution to this system plotted for 0.3, 0.6 and 0.9 time units. As expected, the heat profile is shown to progress to the left. Computation is speeded due to the absence of a reaction term: since there is no creation/destruction of globs over the entire run, there is no need to sort the list until the final pass when the actual function value is computed. The "jagged" appearance of the numerical solution is a result of it being a statistical solution— one would expect the smoothness of the solution to increase as the number of globs increases (this is explored in the next section).

3.2 Finite Domain, Dirichlet Boundaries

Consider the heat equation on a finite domain with constant boundary conditions

$$\begin{cases} u_t = 0.1u_{xx}, & 0.0 \leq x \leq 1.0, \quad t > 0, \\ u(t,0) = 0, & t > 0, \\ u(t,1) = 1, & t > 0, \\ u(0,x) = H(x-1). & \end{cases} \tag{15}$$

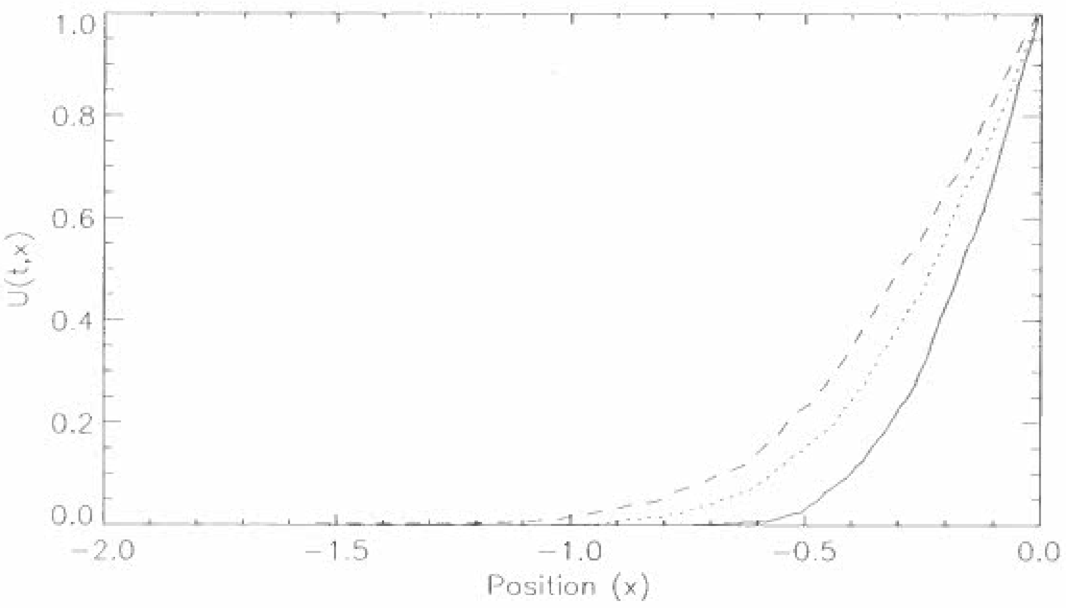

Figure 1. Heat equation on semi-infinite domain. The plot shows the numerical solution to system (14) for 0.3, 0.6 and 0.9 time units from right to left with $\alpha = 0.1$ and $N_0 = 1000$. The heat profile is shown to progress towards the left, as expected.

System (15) may be envisioned as a one-dimensional rod initially set at 0 degree units. At time $t = 0.0$, $u(1,t)$ is held constant at 1 degree unit. This produces a system with a temperature profile progressing towards the left of the domain. Since the function at the right end of the domain is held constant at 1 degree units, a steady state profile of $u(t > t_c, x) = x$ will result for all t greater than some "critical t", t_c (this time is a function of the diffusivity constant, α). Since the boundary conditions are constant, symmetric reflection will be employed at both ends of the domain.

The initial data for the system is the Heaviside unit function centered at $x = 1$ The special derivative of this function is a single δ-jump located there [24]. The height of the jump is 1 unit (corresponding to the range of the $u_0(x)$). To model the initial δ-jump, N_0 globs will be placed in the system, all located at $x = 1$ and all with equal values of $\delta_j^i = 1/N_0$. As in the previous example, there is no reaction term in this system, so the sorting of the glob list will be put off until all of the time loops have been completed.

Setting $N_0 = 1000$ and $\Delta t = 0.01$ gives the results in figure 2. The results of the gradient random walk process are plotted in solid lines, while the analytic solution (truncated Fourier series, see appendix A) solution is plotted in dotted lines. Each pair of lines represent (15) solved for $t = 0.3$,0.6 and 0.9 time units. The non-smooth appearance of the numerical solution is the result of it being a statistical method, which might be expected to improve as N_0 increases (see below). As time increases past the "critical time," the steady state case discussed above forms and remains, within statistical fluctuation, constant(see figure 3).

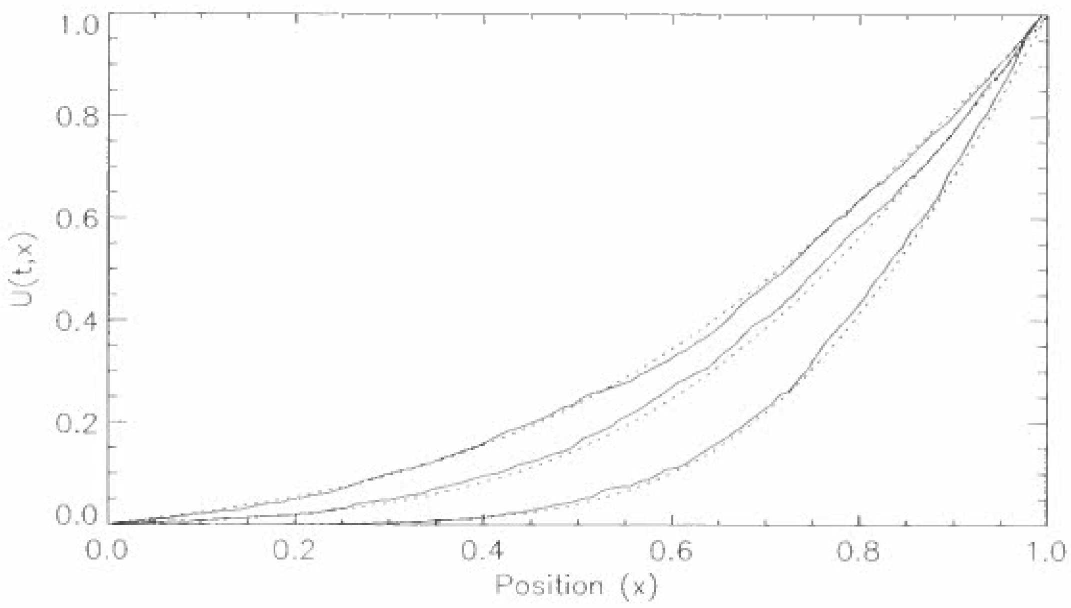

Figure 2. GRW vs. Fourier solution with 1000 globs and $\Delta t = 0.01$. The solid lines show the numerical results, the dotted lines show the analytic solution for $t = 0.3$,0.6 and 0.9 time units from right to left, respectively ($\alpha = 0.1$).

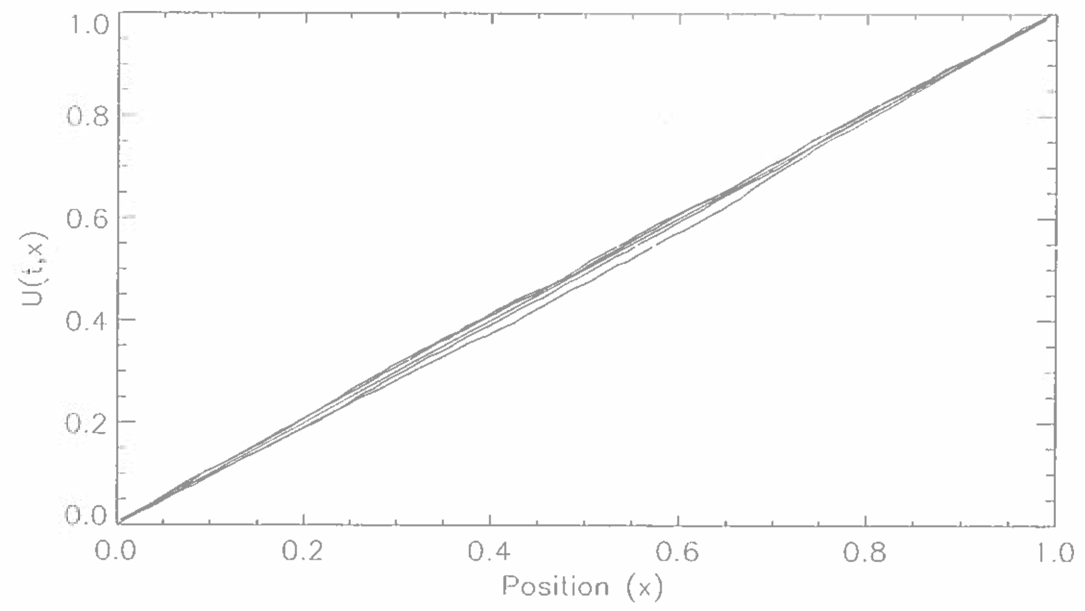

Figure 3. Steady State Solution of Heat Equation. Computed solution (numerical) of system 15 for 3,4,5 and 6 time units plotted with the exact steady state solution (dotted). Notice the computed solutions approximate the analytical steady state solution of $u(x,t) = x$ with statistical fluctuation.

To obtain a measure of the accuracy of the results, the variance between the numerical method and the analytic solution is computed [18]. In the above system, with the initial parameters cited, the variance is computed to be 0.0381. The variance is expected to lessen as the number of globs increases or as the size of the time step is reduced. Figure 4 shows the results when N_0 is set to 5000 globs. The numerical results are not only shown to be smoother, but the variance is reduced to 0.0359, a reduction of 0.0022 units or 5.6%, giving a noticeably more accurate solution to the original system 15. Increasing the number of globs for the run increases the physical memory needed and the CPU time required, which increases from 3.522 to 3.867 ms.,

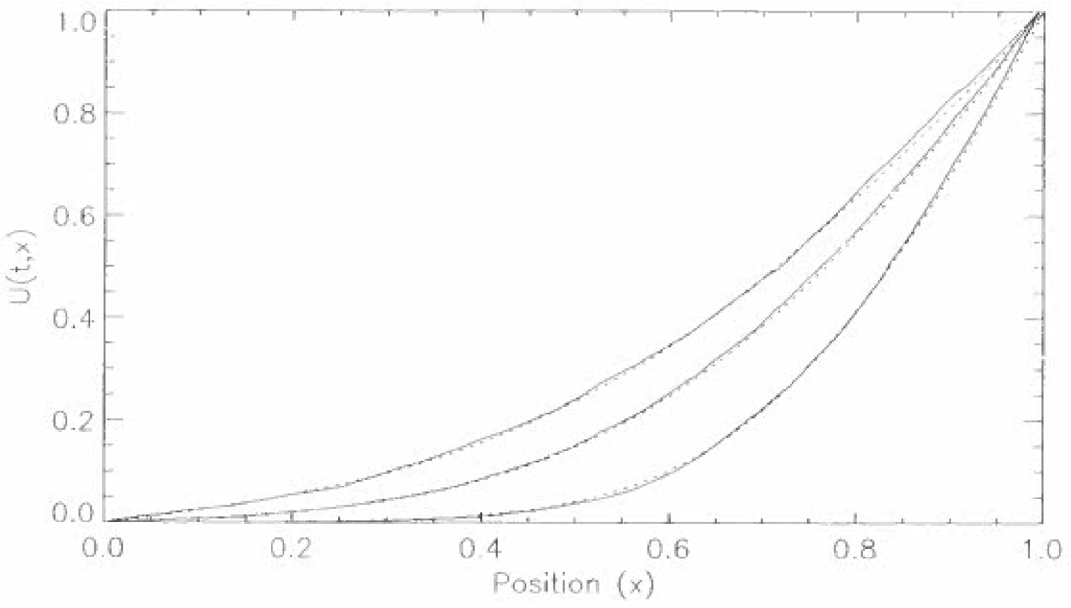

Figure 4. GRW vs. Fourier solution with 5000 globs and $\Delta t = 0.01$. The solid lines show the numerical results and the dotted lines show the analytic solution for $t = 0.3$, 0.6 and 0.9 time units from right to left, respectively. Alpha is set to 0.1.

an increase of 0.345 ms or 9.8%.

As discussed above, the accuracy of the gradient random walk results may be improved by decreasing the size of the time step [22]. However, decreasing the time step also increases the computational cost of the algorithm since a greater number of passes through the loop are required for a given T. Figure 5 shows the above system (15) using 1000 globs and $\Delta t = 0.001$—a tenfold reduction in the size of the time step. The computed variance for this system is 0.0327—a 14% improvement over the original system variance of 0.0381 with the larger time step, and even better than the

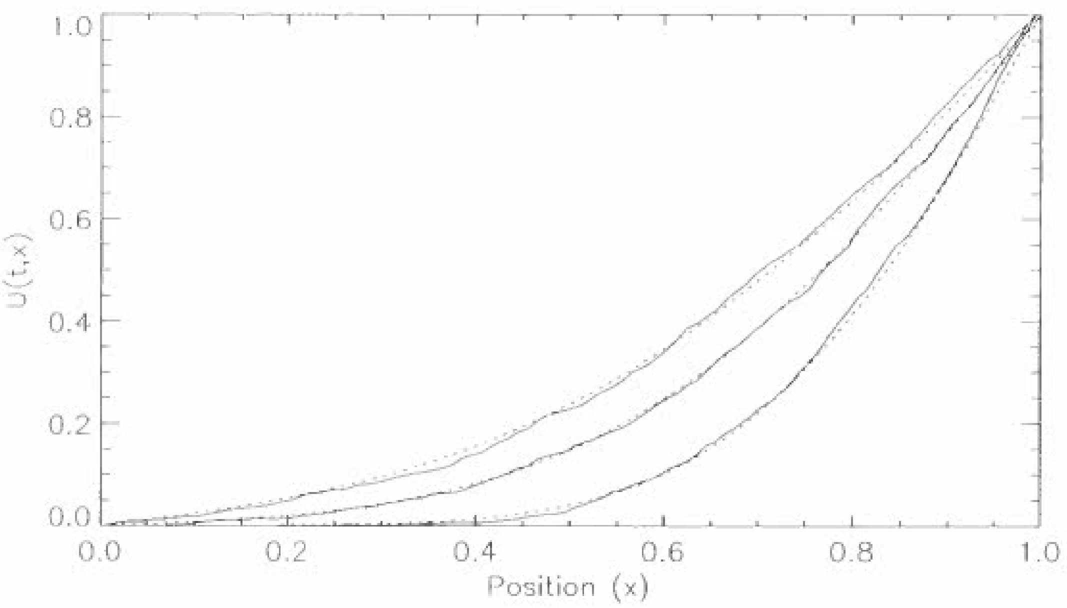

Figure 5. GRW vs. Fourier solution with 1000 globs and $\Delta t = 0.001$. The solid lines show the numerical results and the dotted lines show the analytic solution for $t = 0.3$, 0.6 and 0.9 time units from right to left, respectively. Alpha is set to 0.1.

variance with 5000 globs (0.0359).

Table 1 shows the variance analysis of the gradient random walk for various values of N_0 and Δt. As expected, increasing N_0 and decreasing Δt improves accuracy, but also increases the computational time. The balance between the two (accuracy vs. computational time) must be decided by the individual user—a "best case" value for the initial parameters is elusive at best.

3.3 Finite Domain, Richardson Boundaries

The gradient random walk method has been shown to simulate the heat equation on a semi-infinite domain and a finite domain with Richardson boundary conditions. To

Table 1. Variance analysis for Δt and N_0. The table illustrates that the accuracy of the numerical method is increased as the size of the time step is decreased or as the number of computational elements is increased.

	$N_0 = 100$	$N_0 = 500$	$N_0 = 1000$	$N_0 = 5000$
$\Delta t = 0.1$	0.0479	0.0376	0.0381	0.0359
$\Delta t = 0.01$	0.0396	0.0375	0.0366	0.0357
$\Delta t = 0.001$	0.0382	0.0371	0.0360	0.0353
$\Delta t = 0.0001$	0.0365	0.0362	0.0357	0.0349

test the method with von Neumann boundaries, the original system 13 is changed to the following:

$$\begin{cases} u_t = \alpha u_{xx}, & 0 \leq x \leq 1, \quad t > 0, \\ u(t, 0) = 0, \\ u_x(t, 1) = 0, \\ u(0, x) = \sin(\pi x/2). \end{cases} \tag{16}$$

This system may be thought of as a one dimensional rod (heat may only flow along the x-axis) with the temperature at the left end of the domain held constant at 0.0 and the right end insulated so that no heat may flow in or out across the end.

The initial data is set up so that both of the boundary conditions are met for time $t = 0$. by taking the inverse function of the initial data and assigning the particle locations along the domain so that the initial profile is generated.

Since the boundary condition at the right end is the spatial derivative of the function, anti-symmetric reflection will be employed at this end – that is, any glob traveling out of the domain via the right end will be reflected back into the domain as before, but with the opposite value for the δ-jump.

Figure 6 shows this system for 0.3, 0.6 and 0.9 time units for $\alpha = 0.1$, $N_0 = 5000$ and $\Delta t = 0.001$. Notice the temperature profile retains the initial shape but "deflates" as time progresses. The right end of the profile shows no heat transferred over the boundary (the spatial derivative is held at zero). In time, the profile would come to

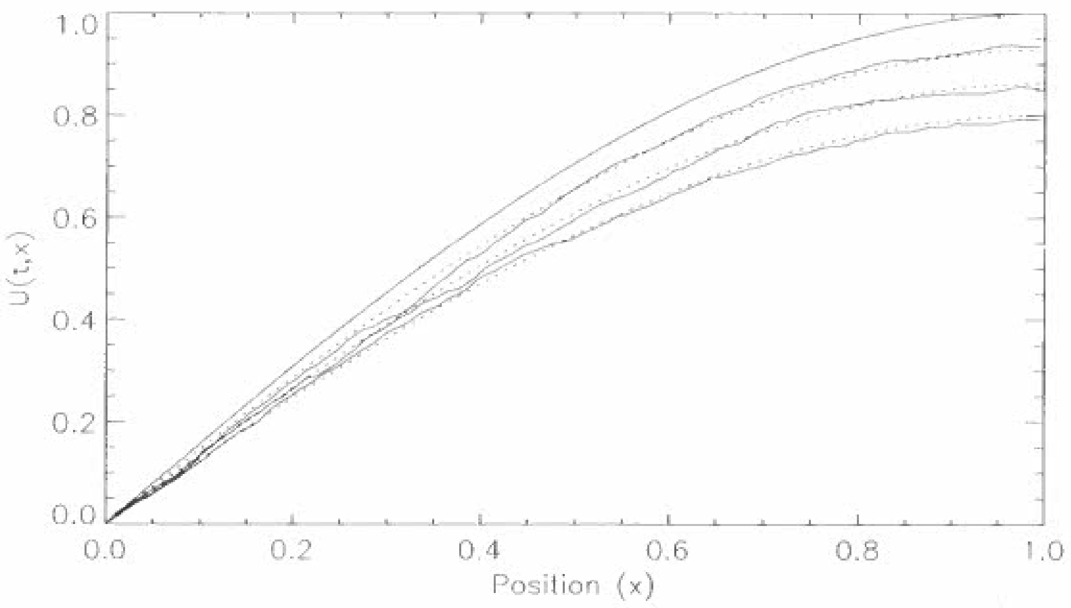

Figure 6. Heat Equation with Von Neumann Boundary Conditions. The dotted line shows the initial state of system 16 with $\alpha = 0.1$ and $\Delta t = 0.001$ for 0.3, 0.6 and 0.9 time steps from top to bottom, respectively ($N_0 = 5000$).

a steady state of zero throughout the domain, as all of the heat would eventually exit through the left boundary.

Again, the numerical method gives the results expected for this system. This serves to illustrate that not only do the mechanics of the method perform as required, but also gives some assurance the random number stream produced by the computer is returning variates which are sufficiently random to meet the needs of the method. This will be critical in the simulations below, which require random numbers not only for the stepping of the globs, but for the creation and destruction statistics as well.

4 THE FITZHUGH-NAGUMO EQUATION

The Fitzhugh–Nagumo equation [31] is a simplified case of the Hodgkin–Huxley equations [7] which model conduction of nerve impulses. The equation may be stated as:

$$u_t = u_{xx} + u(u-a)(1-u) + z, \quad z = \epsilon u. \tag{17}$$

By setting $z = 0$ this system generates a traveling monotonic wave front solution which travels towards the left of the domain and is given by:

$$u(t,x) = \tilde{u}(\xi) = \frac{1}{1+\exp(-\xi/2)} \tag{18}$$

where

$$\xi = x + \theta t \tag{19}$$

and

$$\theta = \sqrt{2}(0.5 - a) \tag{20}$$

which is referred to as the "steady solution."

4.1 Fitzhugh-Nagumo with "Steady Solution" Initial Data

The first case of this system to consider will consist of initial data which is the steady solution (18) with $t = 0$. This should generate a profile already in the shape of the steady solution with a wave profile traveling towards the left of the domain at a constant rate.

Before any simulation can be run with this initial data, the globs must be positioned in order to generate the initial profile. In order to place the globs into the initial position, the steady solution must be inverted, giving

$$x = -2\log\left(\frac{1}{u_i} - 1\right), \quad i = 1, 2 \ldots N_0. \tag{21}$$

Since the solution has a range of $R = [0, 1]$, dividing it into N_0 equal segments and inserting into equation (21) will position the globs in the correct location for the first pass through the algorithm.

The reaction term present in (22) requires that the glob list be sorted and the value for $u(t, x)$ computed through each pass. This will be done immediately after the globs have undergone the random walk step in each time loop. Once the sorting is finished, the reaction algorithm may be conducted as the numerical integration is taking place. The value for the correct glob, along with the current time for the simulation is passed to the explicit reaction computation routine.

Figure 7 shows the system with this initial data for 0.0, 3.0, 6.0 and 9.0 time units (from right to left) using 1000 globs to start and a time step size of 0.1 time unit. The solid lines represent the results of the numerical method with the corresponding analytical solution plotted over the computed results with a dotted line.

The figure shows the initial data in the solution profile of the steady solution, as expected. The wave travels with a constant speed towards the left of the domain while it holds the same profile. Since the analytic solution of the system is known, the exact solution can be plotted with the numerical results to verify accuracy of the method. With the exception of statistical fluctuation, the gradient random walk does an excellent job of modeling this equation.

4.2 Fitzhugh-Nagumo Equation with Non-Smooth Initial Data

Modifying the initial data should affect the computed solution only for the time it takes for the data to reform itself into the shape of the steady solution profile. Once this is accomplished, the solution will assume the same steady state wave speed as in the above example. To illustrate this case, initial data will be constructed in the form:

$$u_0(x) = \begin{cases} 0, & x < -b, \\ 1/2b + 1/2, & -b \leq x \leq b, \\ 1, & x > b. \end{cases} \tag{23}$$

Inverting the initial data equation to place the globs prepares the system for this run. Figure 8 shows the results of running this initial data when $b = 5$. The solution

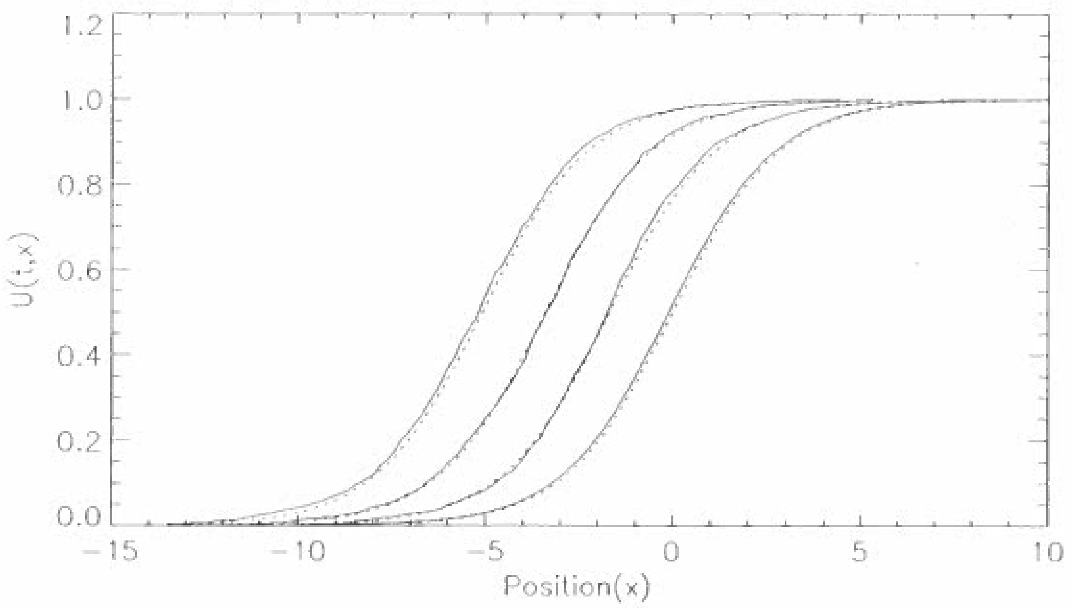

Figure 7. Fitzhugh-Nagumo equation with continuous initial data. The initial data is the steady profile with $t = 0.0$. The time step for this run us $\Delta t = 0.1$. The numerical solution is plotted with solid lines for 0.0,3.0,6.0 and 9.0 time units (from right to left) with the corresponding analytic solutions plotted using dotted lines.

profile at $t = 0$ is line expected– as time continues, the initial curve transforms into the profile of the steady solution (18). Once this profile is achieved, the system is identical to the above case (using the steady solution as the initial data) with the exception of the displacement (the wave front has moved slightly from the initial position by the time the steady solution profile is realized).

4.3 Fitzhugh-Nagumo Equation with Discontinuous Initial Data

To test the robustness of the algorithm for this system, discontinuous initial data will now be used in order to observe how the method handles an extreme case. By choosing the initial data to be the Heaviside function (0 for $x < 0$, 1 for $x > 0$), all of the initial globs may be placed at the origin before the initial pass through the algorithm.

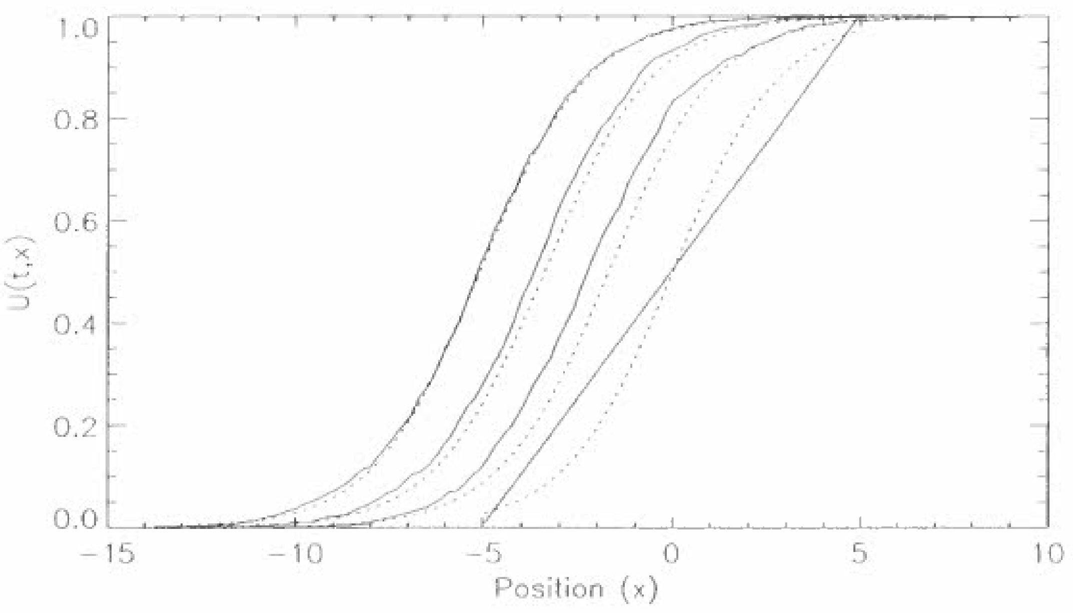

Figure 8. Fitzhugh-Nagumo equation with non-smooth initial data. The plot illustrates the initial profile transforming into the steady solution profile and then proceeding to the left of the domain. The plots are for $t = 0.0$, 3.0, 6.0 and 9.0 time units with $\Delta t = 0.1$.

Figure 9 shows the results of using the gradient random walk method to model this particular initial data. The initial jump is shown at the origin. As time progresses, the solution profile gradually loses the sharp corners of the initial jump and takes on the smooth appearance of the steady solution profile. The wave shows little movement until the steady profile is achieved, at which time the wave progresses towards the left of the domain as both of the above cases.

4.4 Numerical Computation of Reaction Statistic

In preparation for systems with implicit reaction terms (those for which the analytic solution is not yet known), a numerical scheme for the computation of the reaction statistic must be developed. The reaction statistic may involve a function with input parameters of x, t, u, u_x and/or u_{xx}. Systems which require any of the first three of

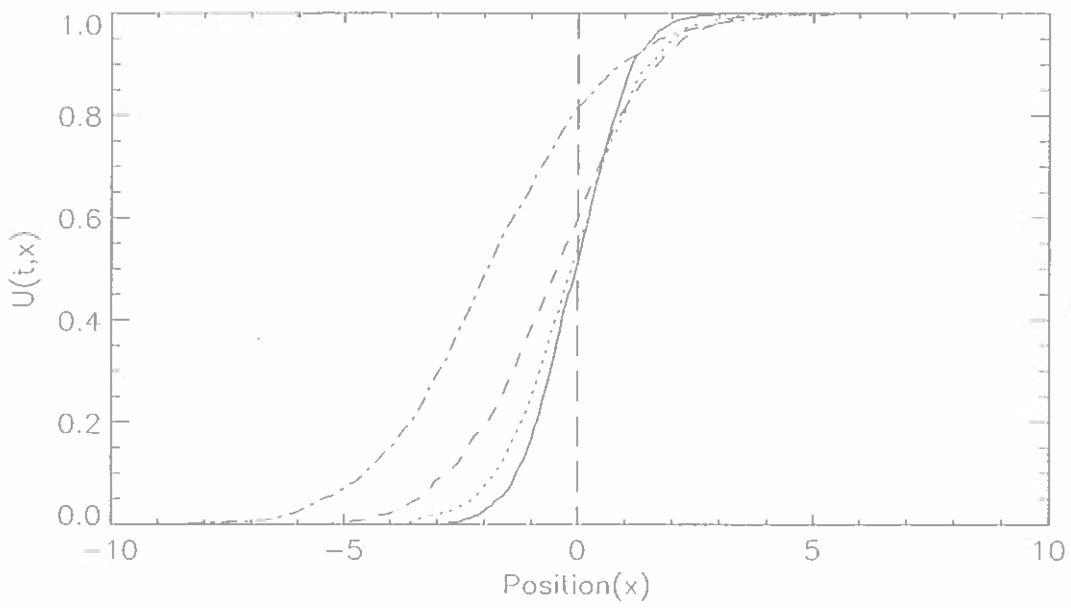

Figure 9. Fitzhugh-Nagumo equation with discontinuous initial data. The plot shows the wave front profile forming and then moving towards the left of the domain. Plots are for $t = 0.0$(vertical dash),0.5(solid), 1.0(dotted), 2.0(dashed) and 4.0(dot-dashed) time units with $\Delta t = 0.1$.

these terms can be computed with relatively little effort, however systems which call for the numerical computation of u_x and u_{xx} call for extra care.

The system presently under consideration lends itself to testing a numerical approach to the computation of the reaction statistic because the exact solution to the system is known. This knowledge allows the numerical representation to be compared to the analytic reaction statistic for analysis. Knowing the reaction term to the system allows the analytic term to be computed as a cubic equation in terms of $u(t,x)$ only. The specific form of the cubic is

$$-3u(t,x)^2 + 2(0.5-a)u(t,x) - a. \tag{24}$$

Since the accuracy of the method of computation is being discussed, the system will be simulated using the exact form of the reaction statistic while a numerical version will be computed for time levels 0.0 and 6.0. With a time step of $\Delta t = 0.1$, this gives a run of 60 iterations. Figure 10 shows the reaction term of this system simulated under these conditions. Though the numerical results show some of the statistical variation expected with the method, overall the numerical results are in good agreement with the analytic solutions.

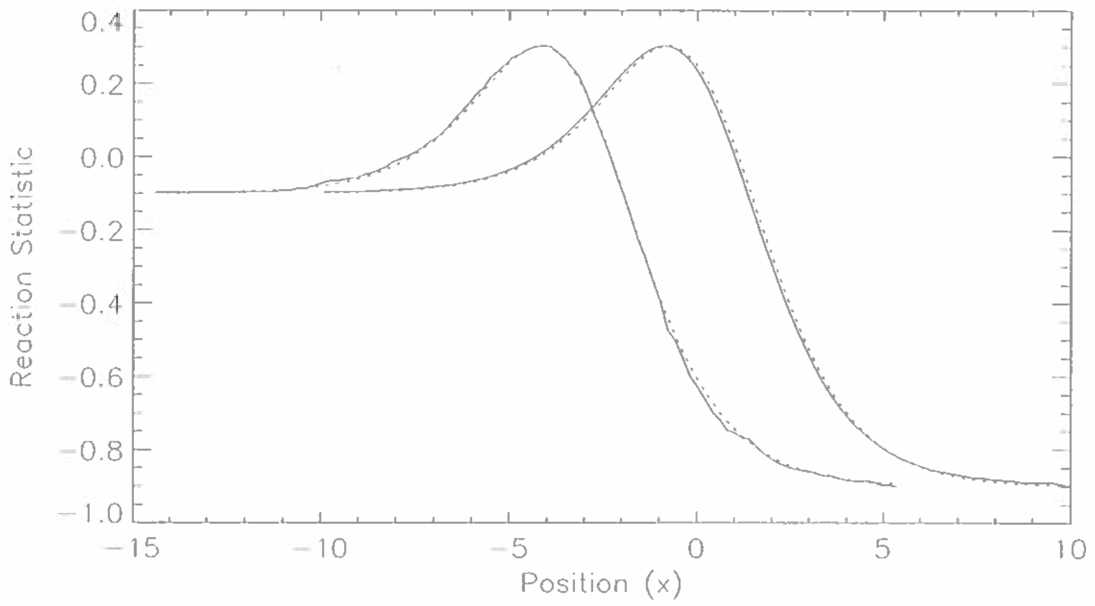

Figure 10. Numerical representation of the reaction statistic. The plot shows the numerical reaction statistic plotted with a solid line with the analytic counterpart for times $t = 0.0$ and 6.0 from right to left with $\Delta t = 0.1$ unit.

4.5 Numerical Computation of $u_x(t, x)$

Though not the case here, the reaction statistic may require the use of either the first or second spatial derivative. While methods are known to compute these values for smooth (and near smooth) data [33], the data produced by the gradient random walk method is not smooth enough for these methods. Figure 11 shows results from employing the method of least squares for a given point, as obtained by using 10 points on either side of the point and computing the slope of the line of "best fit." While this strategy produces a curve with the same general shape and position of the actual derivative, the use of points on either side of a location introduces a "rounding" effect that prohibits the curve from reaching the proper maximum value. As time progresses, the curve becomes so jagged that it is of no practical use in computing a reaction statistic.

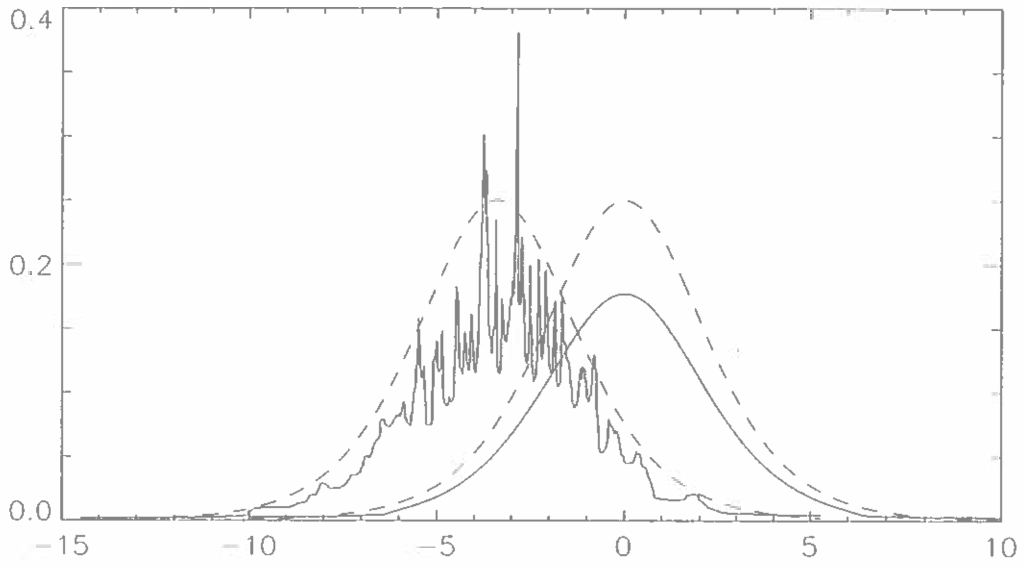

Figure 11. Numerical computation of u_x using the method of least squares. Ten points are used on either side of the given point to compute the slope of the line of "best fit" for $t = 0.0$(right) and 6.0(left) time units (the solid lines represent the computed values and the dashed lines are the exact value).

Another approach to the numerical computation of the derivative might be to recall the procedure used to generate the final data for plotting. The domain has

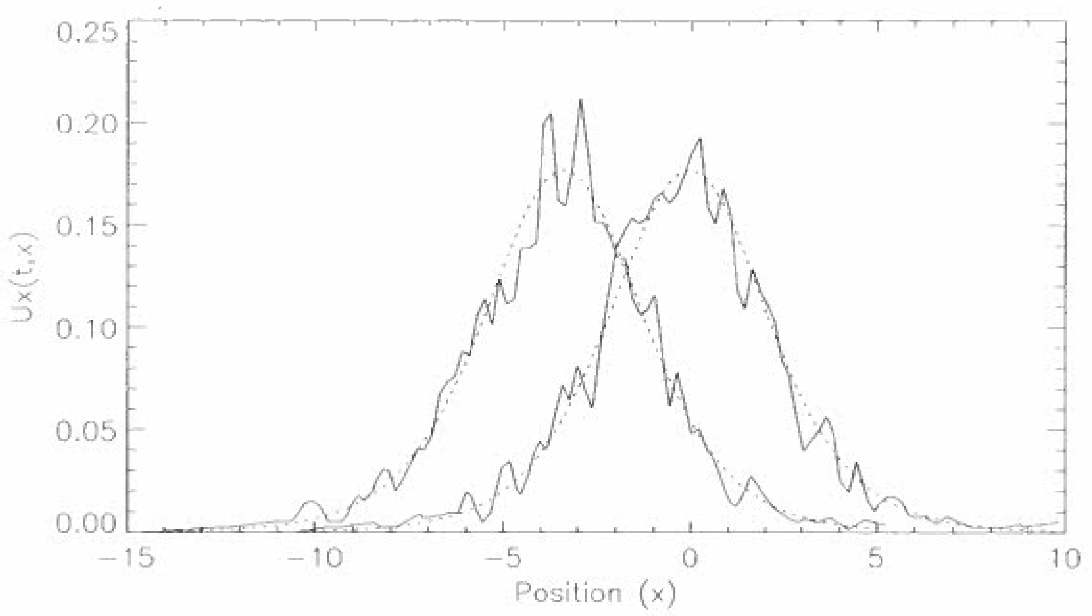

Figure 12. Numerical computation of u_x interpolating values from the grid list. Numerical derivatives for times $t = 0.1$(right) and 6.0(left) time units are shown in solid lines, with the analytic counterparts in dotted lines.

been divided into an arbitrary number of "bins" which collect any of the globs which fall into any individual bin. The value of each bin is then computed as the value of all of the globs contained in it, with the position being the center of the bin. This not only gives the benefit of less points to plot for each run, but also has the tendency to smooth out the rough edges on the final data. Since the curve is smoother than its counterpart from the glob list, the numerical derivative of the grid curve should be more consistent with the actual derivative. Figure 12 shows the results from using the least squares best fit method on the grid list (using one point on either side of the point in question) to compute the slope at any given point on the grid list. These values were then interpolated to every glob in the glob list to assign values to all of the computational elements. While the curves still retain some of the jagged appearance in the previous example, the shape of the numerical derivatives is much closer to the shape of the actual derivatives, and more importantly, the numerical values more closely reflect the true values.

While this method gives results which will work in the computation of reaction statistics which require $u_x(t, x)$, the same approach will not work for computing $u_{xx}(t, x)$ due to the fluctuations in u_x.

5 BURGERS' EQUATION

The gradient random walk method has been shown to simulate systems which include reaction terms which are either implicit or explicit. The method will be applied to Burger's equation

$$\begin{cases} u_t = \alpha u_{xx} + u u_x, \quad a \leq x \leq b, \\ u(t,a) = u_a(t), \\ u(t,B) = u_b(t), \\ u(0,x) = u_0(x), \end{cases} \tag{25}$$

to further observe the behavior of the numerical method.

Setting $a = \infty$, $b = -\infty$, and $u_0(x) = 1 - 2\sqrt{\alpha}\tanh(x/\sqrt{\alpha})$ defines a system that develops a traveling wave with the solution profile of the initial data traveling to the left in the domain, allowing analysis of every aspect of the gradient random walk method for one-dimensional systems. Burgers' equation is of particular interest because it is a (relatively) simple equation combining both the properties of a diffusive wave along with an advection term requiring the values of u, u_x and u_{xx} for the computation of the reaction statistic.

5.1 Initial Results

To model this system, 5000 globs are positioned in the domain to produce the initial data. Figure 13 shows the initial data profile for $\alpha = 0.1$, 0.01 and 0.001, illustrating that not only is the slope of the solution determined by the choice of α, but so is the amplitude of the profile. For this simulation a value of $\alpha = 0.01$ is entered.

Figure 14 shows the system simulated with 5000 initial globs and a time step of $t = 0.001$ for 0.0, 0.3 and 0.6 time units. The numerical results are plotted in solid lines with the analytic counterparts in dotted lines. Though the computed results appear to match up with the exact solution, a closer examination will reveal the "foot" of the wave is drifting away from the correct value. The drifting is compounded in further time steps as the reaction statistic is incorrect not only due to the numerical fluctuations encountered, but also due to the fact that the curve being approximated is not the correct shape to start with.

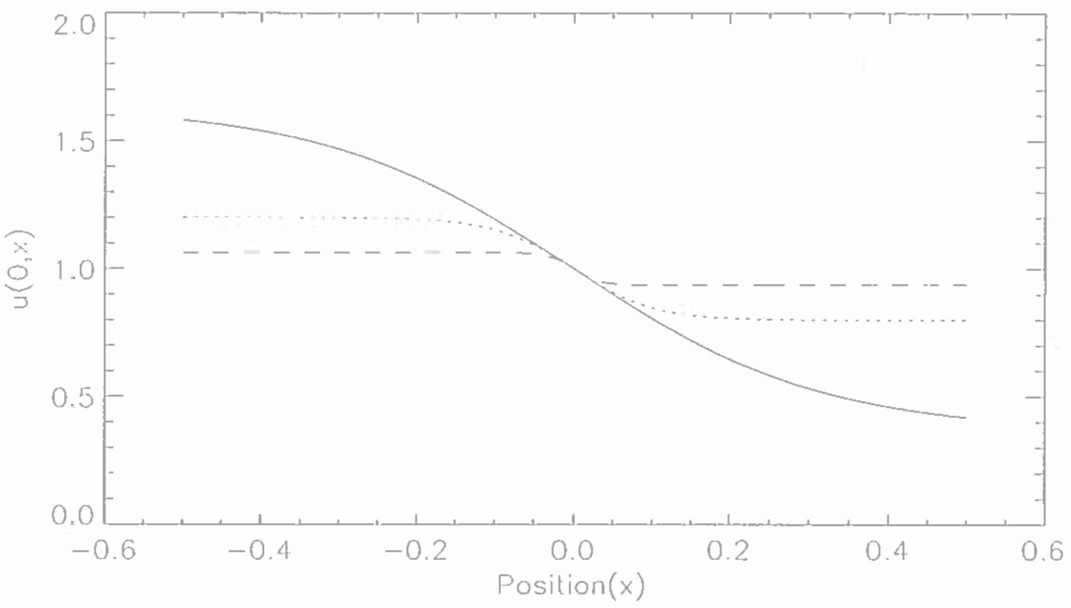

Figure 13. Solution profile for various values of α. Initial data is plotted for $\alpha = 0.1$(solid), 0.01(dotted) and 0.001(dashed) units. This plot illustrates both the slope and amplitude of the solution profile are determined by the initial choice of α.

5.2 Numerical computation of u_x and u_{xx}

The original system (25) shows an advection term of $-uu_x$. Taking the derivative, this yields

$$g(u) = -uu_{xx} - u_x^2. \tag{26}$$

Factoring out a term u_x in order to produce the method form for the transformed system gives

$$-\Big(\frac{uu_{xx}}{u_x} + u_x\Big)u_x, \tag{27}$$

producing a reaction statistic of

$$-\Big(\frac{uu_{xx}}{u_x} + u_x\Big). \tag{28}$$

This illustrates a problem, as unlike the preceeding example which solely required the value of $u(t,x)$ for the computation of the reaction statistic, this system calls for the

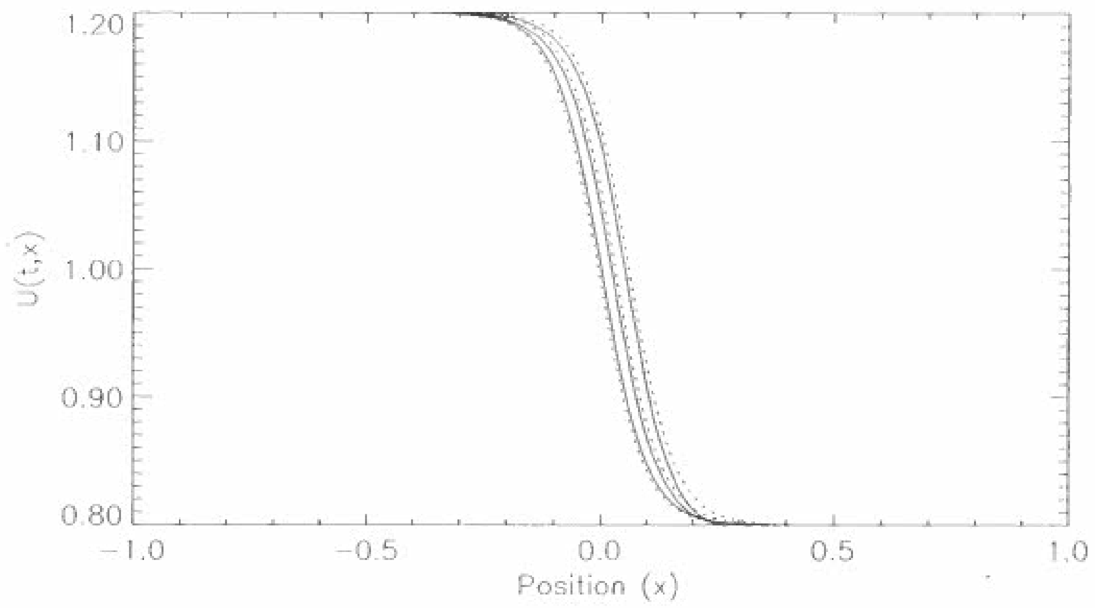

Figure 14. Numerical results computing reaction statistic directly. Numerical results are shown for $t = 0.03$, 0.06 and 0.09 time units in solid lines (from left to right) with their analytic counterparts in dotted lines.

knowledge of u, u_x and u_{xx} for every glob at each pass through the time loop. The reaction statistic is sensitive to fluctuations in the computed values due to the ratio of the two partial derivatives found above.

Figure 15 shows the numerically computed derivative of the solution profile. While the general shape of the curve represents the profile expected, as did the previous example, the statistical fluctuations encountered affect the numerical computation of the second derivative. Figure 16 shows the results of numerically differentiating u_x for time units 0.00 and 0.06 (the curves for $t = 0.03$ have been left out for clarity). Though there appears to be a trend to the profile expected, the fluctuations are too

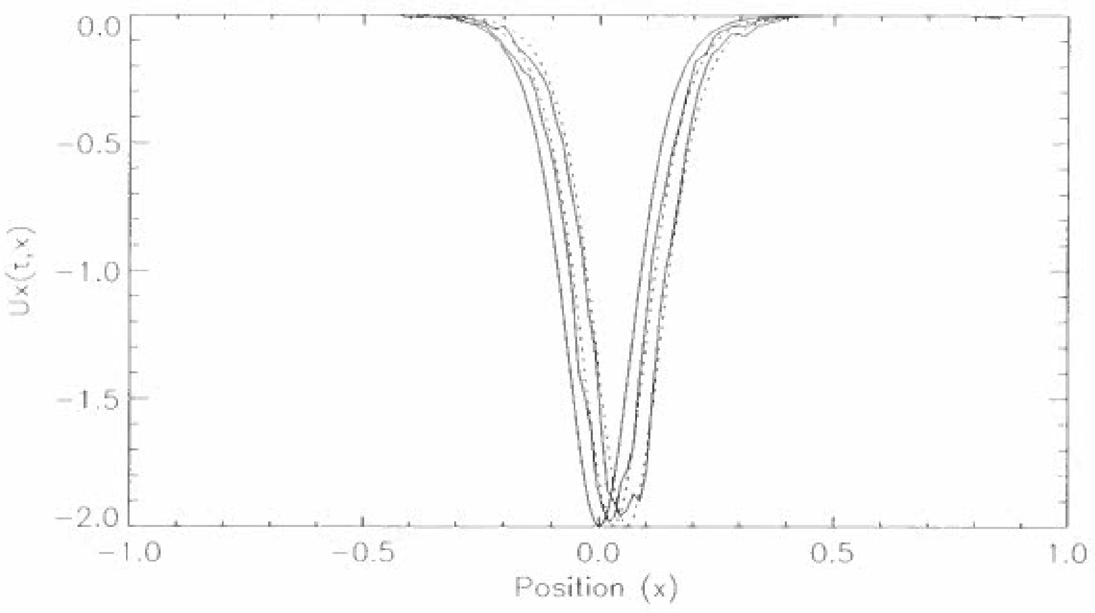

Figure 15. Numerical computation of u_x. The figure shows the first spatial derivative computed numerically (solid) and analytically (dotted) for time $t = 0.00$, 0.03 and 0.06 time units with a time step of 0.001 and $\alpha = 0.001$.

drastic and the amplitude swings prohibit using these results for computation of the reaction statistic.

The computation of the reaction statistic is further hindered by the form of the formula to compute it (27). Not only are both spatial derivatives required, but the ratio between the second to first derivatives is computed in the first half of the formula. This ratio magnifies the error involved because both values are tending towards zero, and a small relative error in this ratio produces statistical noise on the order of many magnitudes. As discussed earlier, the choice for Δt requires the product of the reaction statistic and the time step be much less than one. In order to compensate for the size of the fluctuations involved, Δt would have to be set to a

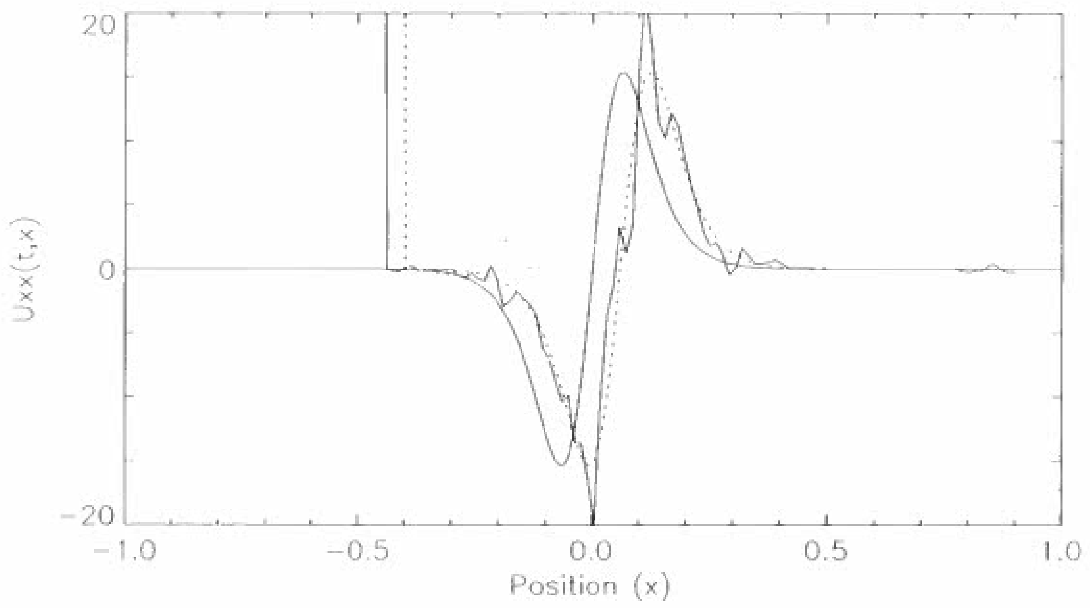

Figure 16. Numerical computation of u_{xx}. The figure shows the second spatial derivative computed numerically(solid) and analytically (dotted) for $t =$ 0.00 and 0.06 time units. Though following the trend of the exact profile, the numerical fluctuations in the computation of the first derivative cause too much noise to overcome.

value too small for any practical use (this still would not correct for the noise involved in the computations).

In order to use the gradient random walk method as a purely statistical method to solve Burgers' equation, a method to compute second derivatives numerically must be developed. In the absence of such a method, one must turn to other strategies.

5.3 The Cole–Hopf Transformation

Cole and Hopf independently uncovered a remarkable non-linear transform which reduces Burgers' equation into a simple heat/diffusion case [34]. Starting with a slightly different form of Burgers' equation,

$$c_t + cc_x = \nu c_{xx}, \tag{29}$$

and defining

$$c = -2\nu\frac{\psi_x}{\psi}, \tag{30}$$

the system reduces to a linear model. First introduce

$$c = \psi_x, \tag{31}$$

giving

$$\begin{aligned} c_t &= \psi_{xt}, \\ c_x &= \psi_{xx}, \\ c_{xx} &= \psi_{xxx}. \end{aligned} \tag{32}$$

Inserting this into equation 29 gives

$$\psi_{xt} + \psi_x\psi_{xx} = \nu\psi_{xxx}. \tag{33}$$

Integration by parts yields

$$\psi_t + \frac{1}{2}\psi_x^2 = \nu\psi_{xx}. \tag{34}$$

Now introduce

$$\psi = -2\nu\log\varphi, \tag{35}$$

giving

$$\begin{aligned} \psi_t &= -2\nu\frac{\varphi_t}{\varphi}, \\ \psi_x &= -2\nu\frac{\varphi_x}{\varphi}, \\ \psi_{xx} &= -2v\frac{\varphi\varphi_{xx} - \varphi_x^2}{\varphi^2}, \end{aligned} \tag{36}$$

which, inserted into equation 34 gives

$$\begin{aligned} -2\nu\frac{\varphi_t}{\varphi} + \frac{1}{2}\Big(-2\nu\frac{\varphi_x}{\varphi}\Big)^2 &= \nu\Bigg(-2\nu\Big(\frac{\varphi\varphi_{xx} - \varphi_x^2}{\varphi^2}\Big)\Bigg) \\ -2\nu\frac{\varphi_t}{\varphi} + 2\nu^2\frac{\varphi_x^2}{\varphi^2} &= -2\nu^2\Big(\frac{\varphi\varphi_{xx} - \varphi_x^2}{\varphi^2}\Big) \\ \varphi\varphi_t - \nu\varphi_x^2 &= \nu\varphi\varphi_{xx} - \nu\varphi_x^2 \\ \varphi_t &= \nu\varphi_{xx}. \end{aligned} \tag{37}$$

This is the heat equation with no reaction term.

In order to use this transform on a system, the initial data must be transformed along with the equation.

5.4 Numerical Application of the Cole-Hopf Transformation

Though the Cole-Hopf transformation allows straightforward computation of the equation, there is a price to be paid. The equation as well as the appropriate initial data and boundary conditions, if applicable, must be transformed. For the data in the above example, the resulting computations are easy to carry out and program. Once the transformed initial data is entered into the routine, the results follow quickly, as the system to simulate has no reaction term—in fact it is precisely the system which was considered when solving the heat equation. After the system is run, the data is run through the transforms in the reverse order as originally performed, returning the desired results.

6 CONCLUSION AND DIRECTIONS FOR FURTHER RESEARCH

The gradient random walk method is able to simulate a wide variety or reaction/diffusion systems. It is particularly well suited for systems with steep gradients or infinite domains, as the computational globs gather where the solution is "interesting," and thin out where it is constant. The method is not without its drawbacks. Areas which need to be addressed before the gradient random walk may be used as a *completely* stochastic method involve smoothing out the statistical fluctuations in the solution profile and computation of spatial derivatives from the solution profile.

7.1 Smoothing Numerical Results

As illustrated in all of the plots contained in this thesis, the results of the gradient random walk method have statistical fluctuations which should be smoothed. This is particularly important where close approximations to the actual solution are desired, as the present results show more of the trend of the solution profile (this may be looked at as an *advantage* of the method, as all physical processes involve statistical error).

The computation of the first derivative was achieved by taking the average value for n number of points around any particular glob. This returned the average of the slopes around the point, which smoothed the curve, but averaging always results in the loss of something (this is, in fact, why it is employed—to help "lose" the fluctuation). As shown in the section on Burgers' equation, more than the noise was lost. The averaged derivative never achieved the same maximum value as the analytically computed counterpart.

Methods exist for smoothing data, however most warn that they are to be used only "to guide the reader through the data," and not for analytical purposes [33]. Methods used in signal processing, such as filters or Fourier transforms, might help to recover the underlying curve. The development of such algorithms should be fruitful research for computational statisticians.

The systems which required the computation of the derivatives in this work employed a grid over the entire domain to help smooth the results and ease the load on the plotting device. The benefits of this technique are shown in the sections dealing with the Fitzhugh-Nagumo and Burgers' equations, but there is a potential problem which could develop if a system produces a shock profile which becomes narrower than the width of the bins. A possible solution to this problem would be to use an adaptive grid algorithm to deploy the grid, as a function of the gradient for the computation of the derivatives. This would not violate the spirit of the particle method, as all of the computations would be performed from the globs.

7.2 Numerical Computation of Spatial Derivatives

If the preceding obstacle is overcome, the computation of the spatial derivatives will be trivial, as the methods exist to compute as many derivatives as desired[21].

Another problem arises after the derivatives are computed. Since the reaction statistic algorithm calls for U_x to be factored out of the formula, quite often the first derivative will end up in the denominator of one or more of the terms in the formula. This is of great concern in areas where the solution profile is constant (or near constant) as in these regions $U_x \rightarrow 0$, creating a large reaction statistic which must be countered with an inversely proportional time step (recall $|g(*)\Delta t| << 1.0$). In cases where the ratio U_{xx}/U_x appears and both values are tending towards zero, further complications arise as the limit is undefined (at least in the present form).

REFERENCES

1 S. Adjerid and J.E. Flaherty, *SIAM J. Numer. Anal.*, **23**, 778 (1986).

2 H. Aref and P.K. Daripa, *SIAM J. Sci. Stat. Comput.*, **5**, 856 (1984).

3 A.J. Baker, *Finite-Element Computational Fluid Dynamics,* (McGraw-Hill/ Hemispere, New York, 1983).

4 V.C. Bhavsar and J.R. Isaac, *SIAM J. Sci. Stat. Comput.*, **8**, 74 (1987).

5 N. Bressan and A. Quateroni, *SIAM J. Numer. Anal.*, **23**, 46 (1986).

6 H.S. Carslaw and J.C. Jaeger, *Conduction of Heat in Solids,* (Oxford University Press, London, 1949).

7 R. Casten, *Quart. Appl. Math.*, **32**, 365 (1975).

8 A.J. Chorin, *Computer Methods in the Applied Sciences and Engineering*, R. Glowinski and J.L. Lions (editors), (North Holland Publishing Co., Amsterdam, 1980).

9 A.J. Chorin, *J. Comput. Phys.*, **8**, 472 (1971).

10 A.J. Chorin, *J. Comput. Phys.*, **25**, 257 (1977).

11 A.J. Chorin, *J. Comput. Phys.*, **27**, 257 (1977).

12 A.J. Chorin, *J. Comput. Phys.*, **27**, 423 (1978).

13 A. J. Chorin, *Stocastic Solution of Reaction/Diffusion Equations*, (Berkeley Press, Berkeley, CA, 1979).

14 R. Courant and D. Hilbert, *Methods of Mathematical Physics, I, II*, (Interscience, New York, 1953).

15 J.Crank, *The Mathematics of Diffusion*, (Oxford University Press, London, 1956).

16 A.L. Fogelson, *J. Comput. Phys.*, **56**, 111 (1984).

17 A.L. Fogelson, *J. Comput. Phys.*, **100**, 1 (1992).

18 A.F. Ghoniem and F.S. Sherman, *J. Comput. Phys.*, **61**, 1 (1985).

19 A.F. Ghoniem and A.K. Oppenhiemm, *AIAA J.*, **22**, 1429 (1983).

20 L. Greengard and V. Rohklin, *J. Comput. Phys.*, **73**, 325 (1987).

21 A. Griewank, *On Automatic Differentiation*, (Argonne National Laboratory, Argonne, IL, 1988).

22 O.H. Hald, *SIAM J. Sci. Stat. Comput.*, **2**, 85 (1981).

23 A.L. Hodgkin and A.F. Huxley, *J. Physiol.(London)*, **117**, 205 (1952).

24 W. Kaplan, *Advanced Calculus*, (Addison-Wesley, Reading, MS, 1984).

25 O.A. Karakashian and W. Rust, *SIAM J. Sci. Stat. Comput.*, **9**, 1085 (1988).

26 O.D. Kellogg, *Foundations of Potential Theory*, (Dover, New York, 1953).

27 J.J. Monaghan, *SIAM J. Sci. Stat. Comput.*, **3**, 422 (1982).

28 F. Milinazzo and P.G. Saffman, *J. Comput. Phys.*, **23**, 380 (1979).

29 T.G. Newman and P.L. Odell, *The Generation of Random Variates*, (Hafner Publishing Co., New York, 1971).

30 P.J. Roach, *Computational Fluid Dynamics*, (Hermosa, Albuquerque, NM, 1972).

31 A.S. Sherman and C.S. Peskin, *SIAM J. Sci. Stat. Comput.*, **9**, 170 (1988).

32 J.H. Smith, *Survey of Three-Dimensional Finite Difference Forms of Heat Equations*, (SC-M-70-83, Sandia Laboratories, Albuquergue, NM, 1970).

33 W. T. Vetterling, S. A. Teukolsky, W. H. Press and B. P. Flannery, *Numerical Recipes in C,* (Cambridge University Press, NY, 1988).

34 G. B. Whitham, *Linear and Non-Linear Waves*, (John Wiley & Sons, New York, 1974).

APPENDIX A

FOURIER SERIES SOLUTION OF THE HEAT EQUATION

Consider the system:

$$(A1)\begin{cases} u_t = \alpha u_{xx}, & 0 \le x \le 1 \\ u(t,0) = 0.0 \\ u(t,1) = 1.0 \\ u(0,x) = H(x-1) \end{cases} \tag{A1}$$

which may be considered a one dimensional rod initially at $T = 0$. When time, t, is started, the right end of the rod is instantaneously raised to $T = 1$ and held constant there. The heat of the system should be expected to diffuse towards to left end of the domain, which is held constant at $T = 0$.

Suppose there is a function, $u(t,x)$ which satisfies the above system. Let $w(x) = x$ and define:

$$v(t,x) = u(t,x) - x(x) \tag{A2}$$

Notice that $v_t = u_t$ and $v_{xx} = u_{xx}$. This gives the following system:

$$(*)\begin{cases} v_t = \alpha v_{xx}, & 0 \le x \le 1, t > 0 \\ v(t,0) = 0.0 \\ v(t,1) = 0.0 \\ v(0,x) = -x \end{cases} \tag{A3}$$

This system is equivalent to (A1) but with boundary conditions which are easier to handle. Now suppose that this solution to this system (A3) is separable. It may then be expressed as:

$$v(t,x) = \sum_{n=1}^{\infty} X_n(x)T_n(t) \tag{A4}$$

thus $v_t = XT''$ and $v_{xx} = X''T$. This gives $XT'' = \alpha X''T$ or:

$$\frac{X''}{X} = \frac{T''}{\alpha T} = k \tag{A5}$$

for some constant k. If $k = -\lambda^2$ is introduced, then $X'' + \lambda X = 0$ or

$$X(x) = A\sin(\lambda x) + B\cos(\lambda x) \tag{A6}$$

Considering the boundary conditions (A2) shows that $B = 0$ and $\lambda = n\pi$. Solving $T' + \alpha\lambda^2 T = 0$ gives

$$T(t) = e^{-\alpha\lambda^2 t} \tag{A7}$$

Substituting (A6) and (A7) into (A4):

$$v(t,x) = \sum_{n=1}^{\infty} A_n e^{-\alpha n^2\pi^2 t}\sin(n\pi x) \tag{A8}$$

To compute the coefficient A_n, consider the initial condition for system (A2).

$$v(0,x) = \sum_{n=1}^{\infty} A_n \sin(n\pi x) = \begin{cases} -x & 0 \le x < 1 \\ 0 & x = 1 \end{cases} \tag{A9}$$

If the initial data is extrapolated to create an odd extension over $[-1,1]$, then (A9) is exactly the form for a Fourier series solution to the system. To recover A_n, solve

$$A_n = \int_{-1}^{1} -x\sin(n\pi x)\,dx \tag{A10}$$

which is easily done with integration by parts. This gives $A_n = (-1^n 2)/(n\pi)$ which in turn gives

$$v(t,x) = \frac{2}{\pi}\sum_{n=1}^{\infty} \frac{(-1)^n}{n} e^{-\alpha n^2\pi^2 t}\sin(n\pi x) \tag{A11}$$

which solves (A3). Solving (A2) then gives

$$u(t,x) = x + \frac{2}{\pi}\sum_{n=1}^{\infty} \frac{(-1)^n}{n} e^{-\alpha n^2\pi^2 t}\sin(n\pi x) \tag{A12}$$

APPENDIX B

INITIALIZATION ROUTINE

This is the initialization routine for the gradient random walk method. It produces a file which is in a form to be read in by the main program for evaluation. To execute the program, type in **initial** at the command line prompt and enter the following:

- **Name of file to create:**

  This may be any valid UNIX filename. If the file already exists, the program will warn the user and ask if the old file should be overwritten.

- **Nature of the domain:**

  Choices include finite, semi-finite and infinite domains. If the domain is not infinite, the program prompts the user for the location of the endpoint(s) of the domain and the boundary condition if applicable.

- **Diffusivity Constant:**

  The constant of diffusivity (referred to as α in the text).

- **Time Step:**

  The value for Δt.

- **Value For the Jumps (dy):**

  Although the jump size need not be uniform, this routine is set up to create the δ-jumps of uniform size. For profiles which are monotonic, this involves dividing the range of the initial data by the number of initial globs. If the data is not monotonic, the total fluctuation in the y direction must be divided by the initial globs.

- **Is There a Reaction Term:**

  If so, the user is prompted to select between an explicit (a formula is known which may be computed directly) or implicit (values must be computed numerically) reaction term.

• **Location of Initial Jump(s):**

The routine now enters a loop which allows the user to input the location of the initial δ-jumps, along with the number of jumps at any particular location. This loop continues until the value -9999 is entered.

The source code for the routine is on the following pages.

Intialization Source Code.

```
/*****************************************************************
*
* Program: initial.c
*        Author:  Greg Lindstrom
*        Date:    June 24, 1992
*
*   This program generates the input file for the gradient random
* walk program 'rw'.  The output is written to the text file
* 'initial.dat' in the form readable by the program.
*****************************************************************/
#include<stdio.h>
#include<stdlib.h>

int left_end_exists      = 0, left_symmetric,
    rite_end_exists      = 0, rite_symmetric,
    reaction_term_exists = 0,
    jumps,
    i=0;

float left_end = 0.0,
      rite_end = 0.0,
      diff_constant,
      time_step,
      location,
      value;

void  get_end();

```

```
main()
{  FILE *output;
   char response, file_name[80];
   int domain_type, done;
   printf("\fWelcome to the Gradient Random Walk set-up");
   printf(" routine....\n\n");
   printf("Enter a filename to create: ");
   gets(file_name);
 /* See if a file with this name exists already. */
   if((output = fopen(file_name,"r")) != NULL){
      printf("WARNING: %s exists.  ");
      printf("Do you want to overwrite it [y|n]: ",file_name);
      response = getchar();
      if((response == 'N') || (response == 'n')) done = 1;
      if(output != NULL) close(output);
    }

   if ((output = fopen(file_name,"w")) == NULL){
      printf("/nWARNING: Error opening output file......\n");
      printf("ABORT....\n\n");
      done = 1;
    };

   if (!done)
   { domain_type = get_domain_type();
     switch (domain_type)
     {
       /* Case 1 is finite domain. */
       case 1: left_end_exists = 1;
               rite_end_exists = 1;
```

```
                printf("\n For the LEFT end of the domain:\n");
                get_end(&left_end, &left_symmetric);
                printf("\n For the RIGHT end of the domain:\n");
                get_end(&rite_end, &rite_symmetric);
           break;
        /* Case 2 is semi-infinite domain. */
        case 2: i = 0;
                while((i != 1) && (i != 2))
                 {
                   printf("\nWould you like to define the:\n");
                   printf("   1> LEFT end of the domain\n");
                   printf("     or\n");
                   printf("   2> RIGHT end of the domain\n");
                   printf("\n Please enter a 1 or 2: ");
                   scanf("%d",&i);
                 };
                if(i == 1)
                { left_end_exists = 1;
                  get_end(&left_end, &left_symmetric);
                }else{
                  rite_end_exists = 1;
                  get_end(&rite_end, &rite_symmetric);
                };
           break;
        /* Case 3 is infinite domain. */
        case 3:
           break;
      };
    printf("\n\n");
    printf("Enter Diffusivity constant (alpha): ");
    scanf("%f",&diff_constant);
    printf("Enter Time Step: ");
    scanf("%f",&time_step);
    printf("Enter value for the jumps: ");
    scanf("%f",&value);
    response = 'x';
    while((response != 'y') && (response != 'Y')
       && (response != 'n') && (response != 'N'))
     { printf("Is there a reaction term in this run [y|n]? ");
       scanf("%s",&response);
     };
    reaction_term_exists = 0;
    if((response == 'y') || (response == 'Y'))
      while((reaction_term_exists != 1)
         && (reaction_term_exists != 2))
     { printf("Is the reaction term:\n");
       printf("   1) Explicit (formula is known)\n");
       printf("         or\n");
       printf("   2) Implicit (computed numerically)\n");
       printf("\n Please enter 1 or 2: ");
       scanf("%d",&reaction_term_exists);
     };

    fprintf(output,"%d, %f, %d\n",
                 left_end_exists,left_end,left_symmetric);
    fprintf(output,"%d, %f, %d\n",
                 rite_end_exists,rite_end,rite_symmetric);
    fprintf(output,"%f\n", diff_constant);
    fprintf(output,"%f\n", time_step);
    fprintf(output,"%d\n", reaction_term_exists);
```

```

     i = 1;
     location = 0.0;
     while(location > -9998)
      {
        printf("Enter location of jump %d (-9999 to exit): ",i);
        if(location > -9998)
         {
           scanf("%f",&location);
           if(location > -9998)
            { printf("How many at this location ? ");
              scanf("%d", &jumps);
              fprintf(output,"%d, %f, %f\n",jumps,location,value);
            };
         };
      };
     fprintf(output,"%d, %f, %d\n", -1,0.0,0);
    }
    else printf("\n\nGood-bye...\n\n");

}; /*** END MAIN PROGRAM ***/


/*--- get_domain_type ------------------------------------------
   This function prompts the user to enter the type of domain
   desired for this run of the program.  A finite, semi-finite
   or infinite domain may be specified.
---------------------------------------------------------------*/
int get_domain_type()
{
   int domain_type = 0;

   while((domain_type < 1) || (domain_type > 3))
   { printf("\n\n\n");
     printf("Is the domain of this set-up:\n\n");
     printf("       1> Finite \n");
     printf("       2> Semi-Infinite\n");
     printf("       3> Infinite\n\n");
     printf("  Please enter 1, 2 or 3: ");
     scanf("%d",&domain_type);
    };

   return(domain_type);
}; /* End function get_domain_type. */


/*--- get_end ----------------------------------------------------
    This routine get the location of the end and the reflection
   criteria.
----------------------------------------------------------------*/
void get_end(end, symmetric)
  int   *symmetric;
  float *end;
{
   float temp;

   printf("   Enter location: ");
   scanf("%f",&(*end));
   i = 0;
   while ((i != 1) && (i != 2))
```

```
    {
      printf("\nIs the reflection:\n");
      printf("    1> Dirichlet (symmetric)\n");
      printf("    2> Newmann (anti-symmetric)\n");
      printf("Please enter 1 or 2: ");
      scanf("%d",&i);
     };

    if (i == 1)  *symmetric = 1;
}; /* End get_end. */

```

APPENDIX C

GRADIENT RANDOM WALK SOURCE CODE

Main Program

```
/*--- Gradient Random Walk Main Program (see ./README) ---*/



/*--- Get the global data structures, variables, subroutines. */
#include "./lib/rw_header"

main()
{
/*--- Get local variable definitions. ---*/
#include "./lib/local_variables"


  /*--- Get input/output file name. ---*/
  Get_input_file_name(input_file_name);
  Get_output_file_name(output_file_name);


  /*--- Prepare the front of the glob & grid list for passing. */
  frnt_glob = Get_input(frnt_glob,back_glob,input_file_name);
  frnt_grid = (struct glob *) calloc(1,sizeof(struct glob));
  frnt_grid->prev = frnt_grid->next = NULL;


  /*--- Prompt user for time level to compute. ---*/
  printf("\n\nEnter time level to compute: ");
  scanf("%f", &max_time);


  /*--- Main Time Loop.  ---*/
  curr_time = time_step;
  while((curr_time <= max_time)&&(max_time != 0.0))
   {printf("time = %f (%d globs)\n",curr_time,number_of_globs);

    /*--- Perform the random walk step on the glob list. ---*/
    frnt_glob  = Step_the_globs(frnt_glob);
    back_glob = frnt_glob;

    /*--- This loop is executed if there is a reaction term. ---*/
    while(back_glob->next != NULL) back_glob=back_glob->next;
    if (reaction_term_exists)
     { frnt_glob = Sort_the_globs(frnt_glob);
       frnt_glob = Reaction(frnt_glob,back_glob,frnt_grid,
                                  curr_time,reaction_term_exists);
     }

    /*--- Advance time step and continue loop. ---*/
    curr_time += time_step;
   }


/*--- Now compute U(t,x) for plotting. ---*/
  printf("Sorting Complete!! Computing final pass.\n");
  Compute_Ux(frnt_grid,frnt_glob);

```

```

/*--- Write numerical results to the output file. ---*/
  curr_grid = frnt_grid;
  while(curr_grid != NULL)
   { fprintf(output,"%f, %f\n",curr_grid->location,curr_grid->U);
     curr_grid = curr_grid->next;
   }

/*--- Close output file, if it is open. ---*/
  if(output != NULL) close(output);


/*--- Print summary to the screen. ---*/
  printf("\n Initial parameters for this run: \n\n");
  printf("Input file name: %s\n",input_file_name);
  printf("    left_end = %f, left_symmetric = %d\n",
                                          left_end,left_symmetric);
  printf("    rite_end = %f, rite_symmetric = %d\n",
                                          rite_end,rite_symmetric);
  printf("    alpha = %f, time step = %f\n",alpha, time_step);
  printf("    dy = %f\n", frnt_glob->value);
  printf("    Max Time = %f\n\n", max_time);

  printf("Simulation Complete. Results written to: %s\n\n",
                                           output_file_name);
};
```

Add a glob

```
/*--- Add_a_glob   ------------------------------------------
*   This subroutine accepts a pointer to a structure in the
*  glob list then creates an identical glob, placing it in
*  the next position in the list.  The routine returns the
*  pointer to the created glob.
*
*   Add_a_glob is called from the Reaction subroutine.
*----------------------------------------------------------*/
struct glob *Add_a_glob(frnt_glob,curr_glob,back_glob)
  struct glob *frnt_glob,*curr_glob,back_glob;
{ struct glob *temp_glob=(struct glob *)
                              calloc(1,sizeof(struct glob));


  if(curr_glob->next != NULL)
   {temp_glob->next       = curr_glob->next;
    curr_glob->next->prev = temp_glob;
    temp_glob->prev       = curr_glob;
    curr_glob->next       = temp_glob;
 }else{
    curr_glob->next = temp_glob;
    temp_glob->next = NULL;
    temp_glob->prev = curr_glob;
   }

  temp_glob->location = curr_glob->location;
  temp_glob->value    = curr_glob->value;

  return(temp_glob);
};/* End Add_a_glob. */
```

Compute U

```
/*---   Compute_U ----------------------------------------
*   This subroutine accepts pointers to the front of
*  the glob list (computational elements for the method)
*  and the grid list (elements to print out).  The method
*  intgrates the glob list as per the gradient random walk
*  method, placing the results in the grid list.  The routine
*  returns a pointer at the fronjt of the grid list.
----------------------------------------------------------*/
struct glob *Compute_U(frnt_glob,frnt_grid)
 struct glob *frnt_glob, *frnt_grid;

{  float        bin_width    = 0.01,
                bin_rite_end;
   struct glob *curr_glob    = frnt_glob,
               *curr_grid    = frnt_grid;


/*---  Clear out (free) old grid.   ---*/
   while(frnt_grid->next != NULL){
      frnt_grid = frnt_grid->next;
      free(frnt_grid->prev);
    }


/*--- Set the first bin. If the domain is finite, it is
*  divided into "bins" of bin_width size.  If the domain is
*  semi-finite or infinite, start at the left end, and proceed
*  with as many bins of width bin_width as needed.---*/
```

```

   if(left_end_exists)
      bin_rite_end = left_end + bin_width;
   else{ while(curr_glob->next != NULL)
      curr_glob = curr_glob->next;

   bin_width = (curr_glob->location-frnt_glob->location)/100.;
   bin_rite_end = frnt_glob->location + bin_width;
   curr_glob    = frnt_glob;}


/*---  Prepare fresh grid ---*/
   frnt_grid->next = NULL;
   if(left_end_exists)
      frnt_grid->location=left_end + (bin_width/2.0);
   else
      frnt_grid->location=frnt_glob->location +(bin_width/2.);
   frnt_grid->U = 0.0;



/*---   Construct new grid and compute U   ---*/
   curr_grid = frnt_grid;
   while(curr_glob != NULL)
      if(curr_glob->location < bin_rite_end)
       { curr_grid->U     += curr_glob->value;
         curr_glob         = curr_glob->next;
     }else{
         curr_grid->next        = (struct glob *)
                                  calloc(1,sizeof(struct glob));
```

```
            curr_grid->next->prev = curr_grid;
            curr_grid             = curr_grid->next;
            curr_grid->next       = NULL;
            curr_grid->location   = curr_grid->prev->location
                                  + bin_width;
            curr_grid->U          = curr_grid->prev->U;
            bin_rite_end         += bin_width;
          };

      return(frnt_grid);
   };
```

E reaction

```
/*--- E_reaction  (Explicit Reaction) -------------------
*
*   This routine is called by the subroutine Reaction when the
*  reaction statistic is defined by an explicit function (one
*  known in terms of x and t).  It accepts two real valued para-
*  meters and returns the real valued value of the reaction
*  statistic. It is called when the input parameter
*  "reaction_term_exists" is set to 1.
*-----------------------------------------------------------
*
*  IMPORTANT NOTE: THIS IS NOT THE REACTION TERM !
*
*  If the original system is:
*
*        Ut = aUxx + F(t,x), etc.,
*
*  then this is G(t,x) where
*        F' = G * Ux
*
*------------------------------------------------------------*/
float E_reaction(t,x)
   float t,x;
{
   float a     = 0.1,
         theta = sqrt(2.0)*(0.5 - a),
         xi    = x + theta * t,
         z     = -xi/sqrt(2.0),
         u     = 1.0/(1.0+exp(z));

   return(-3.*pow(u,2.0) + 2.0*(a+1.0)*u - a);
}
```

Get Input

```
struct glob *Get_input(frnt_glob, back_glob, input_file_name)
   struct glob *frnt_glob, *back_glob;
   char input_file_name[80];
{
   FILE  *input=fopen(input_file_name,"r");
   int   j,
/*         number_of_jumps = 0, */
           jumps_at_this_location;
   float temp_location,
           value_at_this_location;


/*---  Read in Initial Parameters  -----------------------*/
   fscanf(input,"%d, %f, %d",
      &left_end_exists,&left_end,&left_symmetric);
   fscanf(input,"%d, %f, %d",
      &rite_end_exists,&rite_end,&rite_symmetric);
   fscanf(input,"%f",        &alpha);
   fscanf(input,"%f",        &time_step);
   fscanf(input,"%d",        &reaction_term_exists);


/*---  Set up the initial data glob:  This is a temporary
       glob which will be removed from the list after it
       is read in.                         */
   frnt_glob = (struct glob *) calloc(1,sizeof(struct glob));
   frnt_glob->next = frnt_glob->prev = NULL;
   back_glob       = frnt_glob;
/*---  Now read in the initial data.  ---------------------*/
   jumps_at_this_location = 0;
   while(jumps_at_this_location != -1)
    { fscanf(input,"%d,%f,%f",&jumps_at_this_location,
                               &temp_location,
                               &value_at_this_location);

      if(jumps_at_this_location != -1)
       { for(j=1;j<=jumps_at_this_location;++j)
          { back_glob->next = (struct glob *)
                               calloc(1,sizeof(struct glob));
            back_glob->next->prev = back_glob;
            back_glob             = back_glob->next;
            back_glob->next       = NULL;
            back_glob->location   = temp_location;
            back_glob->value      = value_at_this_location;
            ++number_of_globs;
          };
       };
    };


/*--- Advance frnt_glob 1 unit and free "dummy" glob. ---*/
   frnt_glob = frnt_glob->next;
   free(frnt_glob->prev);
   frnt_glob->prev = NULL;

   if(input != NULL) close(input);
   return(frnt_glob);
}
```

Get input file name

```
void Get_input_file_name(input_file)
  char input_file[];
{
  printf("\nEnter input file name: ");
  gets(input_file);
  if((input = fopen(input_file,"r"))==NULL){
    printf("\n ERROR: File does not exist.\n\n");
    exit(1);}
}
```

Get output file name

```
/** Get output file name. Warn if the file already exists. **/
void Get_output_file_name(output_file_name)
  char output_file_name[];
{
  char response;

  printf("\nEnter filename to OUTPUT results: ");
  gets(output_file_name);
  if((output = fopen(output_file_name,"r"))!=NULL){
    printf("WARNING: Output file exists.");
    printf(" Do you wish to overwrite [y|n]: ");
    scanf("%c",&response);
    if((response != 'Y') && (response != 'y')){
      printf("\nGood-bye\n");
      exit(1);}
    fclose(output);
   }
  output = fopen(output_file_name,"w");
};
```

Get slope

```
float Get_slope(curr_list,key)
   int key;
   struct glob *curr_list;
{
   int n = 0;

   float Sx  = 0.0,
         Sxx = 0.0,
         Sy  = 0.0,
         Sxy = 0.0;

if(key == 1)/*--- Get slope for first derivative. ---*/
 { if (curr_list->prev != NULL)
    { Sx  += curr_list->prev->location;
      Sxx += curr_list->prev->location*curr_list->prev->location;
      Sy  += curr_list->prev->U;
      Sxy += curr_list->prev->location * curr_list->prev->U;
          ++n;
    }

   Sx  += curr_list->location;
   Sxx += curr_list->location * curr_list->location;
   Sy  += curr_list->U;
   Sxy += curr_list->location * curr_list->U;
       ++n;

   if(curr_list->next != NULL)
    { Sx  += curr_list->next->location;
```

```
        Sxx += curr_list->next->location
             * curr_list->next->location;
        Sy  += curr_list->next->U;
        Sxy += curr_list->next->location
             * curr_list->next->U;
             ++n;
      }
  }

if(key == 2)/*--- Get slope for second derivative ---*/
 { if (curr_list->prev != NULL)
     { Sx  += curr_list->prev->location;
       Sxx += curr_list->prev->location
            * curr_list->prev->location;
       Sy  += curr_list->prev->Ux;
       Sxy += curr_list->prev->location
            * curr_list->prev->Ux;
            ++n;
     }

    Sx  += curr_list->location;
    Sxx += curr_list->location * curr_list->location;
    Sy  += curr_list->Ux;
    Sxy += curr_list->location * curr_list->Ux;
        ++n;

    if(curr_list->next != NULL)
     { Sx  += curr_list->next->location;
       Sxx += curr_list->next->location
            * curr_list->next->location;
      Sy  += curr_list->next->Ux;
      Sxy += curr_list->next->location
           * curr_list->next->Ux;
          ++n;
    }
 }
return((n*Sxy - Sx*Sy)/(n*Sxx - Sx * Sx));
}
```

Nuke a glob

```
struct glob *Nuke_a_glob(frnt_glob, curr_glob,back_glob)
   struct glob *frnt_glob, *curr_glob, *back_glob;
{ struct glob *temp_glob;


  if((curr_glob->prev != NULL) && (curr_glob->next != NULL)) {
    temp_glob             = curr_glob->next;
    curr_glob->prev->next = temp_glob;
    curr_glob->next->prev = curr_glob->prev;}
  else if (curr_glob->prev == NULL){
    temp_glob       = curr_glob->next;
    frnt_glob       = temp_glob;
    frnt_glob->prev = NULL;}
  else{
    temp_glob       = curr_glob->prev;
    back_glob       = temp_glob;
    back_glob->next = NULL;};


  free(curr_glob);
  return(temp_glob);
}



```

N reaction

```
float N_reaction(U,Ux,Uxx)
  float U, Ux, Uxx;

{
 return(-(U*Uxx)/Ux - Ux);
};
```

Quicksort

```
/***   QuickSort   *****************************************
*
*   This routine sorts the list of globs using the quicksort
*  method adapted for a doublely linked list.
*
*************************************************************/
struct glob *QuickSort(frnt_glob,back_glob)
 struct glob *frnt_glob,*back_glob;
{
   struct glob *split_glob;

   if(frnt_glob != back_glob)
    { split_glob = QuickSplit(frnt_glob,back_glob);
      if(split_glob != frnt_glob)
                       QuickSort(frnt_glob,split_glob->prev);
      if(split_glob != back_glob)
                       QuickSort(split_glob->next,back_glob);
    };

   return(frnt_glob);
}
```

Quicksplit

```
/***   QuickSplit   ****************************************/

struct glob *QuickSplit(frnt_glob,back_glob)
 struct glob *frnt_glob,*back_glob;
{
   struct glob *lp = frnt_glob,
               *rp = back_glob;

   while(lp != rp)
    {  while((frnt_glob->location >= lp->location)
          && (lp != rp)) lp = lp->next;
       while((frnt_glob->location <  rp->location)
          && (lp != rp)) rp = rp->prev;
       Swap(lp,rp);
    };

   if((lp == back_glob)
    &&(frnt_glob->location > back_glob->location))
    { Swap(frnt_glob,back_glob);
      rp = back_glob;
    }
   else
    { Swap(frnt_glob, lp->prev);
      rp = lp->prev;
    };

   return(rp);
};
```

Reaction

```
struct glob *Reaction(frnt_glob,back_glob,frnt_grid,
                              t,reaction_term_exists)
 int   reaction_term_exists;
 float t;
 struct glob *frnt_glob, *back_glob, *frnt_grid;
{  float critical_value;
   struct glob *curr_glob = frnt_glob;

   if(reaction_term_exists == 2)/* Numerical reaction */
     Compute_Ux(frnt_grid,frnt_glob);

   while(curr_glob != NULL){
     if(reaction_term_exists == 1)
        critical_value = time_step
                       * E_reaction(t, curr_glob->location);
     else if(reaction_term_exists == 2)
        critical_value = time_step
                       * N_reaction(curr_glob->U,
                                    curr_glob->Ux,
                                    curr_glob->Uxx);
     else
        printf("\nBogus value for reaction_term_exisits ");
        printf("in Reaction.\n\n");

    if(fabs(critical_value) > drand48())
         if(critical_value > 0.0){
            curr_glob = Add_a_glob(frnt_glob,curr_glob,
                                                  back_glob);
```

```
                ++number_of_globs;
            }else{
                curr_glob = Nuke_a_glob(frnt_glob,curr_glob,
                                                      back_glob);
                --number_of_globs;
             }

          if(curr_glob != NULL) curr_glob = curr_glob->next;
      }
    return(frnt_glob);
}
```

Reflect

```
struct glob *reflect(curr_glob)
   struct glob *curr_glob;
{
   double reflected_location = curr_glob->location;

   while( (left_end_exists)&&(reflected_location < left_end)
        ||(rite_end_exists)&&(reflected_location > rite_end))
    {
      if((left_end_exists)&&(reflected_location < left_end))
       {
         reflected_location =2.*left_end - reflected_location;
         if(left_symmetric == 0)
            curr_glob->value = -curr_glob->value;
       };
      if((rite_end_exists)&&(reflected_location > rite_end))
       {
         reflected_location =2.*rite_end - reflected_location;
         if(rite_symmetric == 0)
            curr_glob->value = -curr_glob->value;
       };
    };

   curr_glob->location = reflected_location;

   return(curr_glob);
}
```

Sort the globs

```
struct glob *Sort_the_globs(frnt_glob)
   struct glob *frnt_glob;
{
   struct glob *back_glob = frnt_glob,
               *curr_glob = frnt_glob,
               *sort_glob = frnt_glob,
               *temp_glob = frnt_glob;

   if(frnt_glob == NULL)
      printf("No globs in list.\n");
   else
    {
      temp_glob = (struct glob *) calloc(1,sizeof(struct glob));
      frnt_glob->prev = temp_glob;
      temp_glob->next = frnt_glob;
      temp_glob->prev = NULL;
      frnt_glob       = temp_glob;

      sort_glob       = back_glob->next;
      back_glob->next = NULL;

      while(sort_glob != NULL)
       {
         curr_glob = back_glob;
         while((curr_glob->prev != NULL)
             &&(curr_glob->location > sort_glob->location))
            curr_glob = curr_glob->prev;
         temp_glob = sort_glob->next;
```

```
        if(curr_glob->next == NULL)
         {
           curr_glob->next = sort_glob;
           sort_glob->next = NULL;
           sort_glob->prev = curr_glob;
           back_glob       = sort_glob;
         }
        else
         {
           curr_glob->next->prev = sort_glob;
           sort_glob->next       = curr_glob->next;
           curr_glob->next       = sort_glob;
           sort_glob->prev       = curr_glob;
         };
        sort_glob = temp_glob;
      };
   };
  temp_glob       = frnt_glob;
  frnt_glob       = frnt_glob->next;
  frnt_glob->prev = NULL;
  free(temp_glob);

  return(frnt_glob);
}

```

Step

```
/***   Step   ************************************************
*
*      This routine generates a Guassian distribution with mean
*    zero and given variance (2*D*time_step) using the Box-
*    Muller method found in "The Generation of Random Variates"
*    by Newman/Odel.
*
*************************************************************/
double Step()
{
   double rand1 = 0.0,
          rand2;

   while(rand1 == 0.0) rand1 = drand48();
   rand2 = drand48();

   return(sqrt(-4.*alpha*time_step*log(rand1))
                                          *cos(2.*pi*rand2));

}
```

Step the globs

```
struct glob *Step_the_globs(frnt_glob)
 struct glob *frnt_glob;
{
   struct glob *curr_glob = frnt_glob;
   float reflected_location = curr_glob->location;

   while(curr_glob != NULL)
    {
      /*--- Assign new position. ---*/
      curr_glob->location += Step();

/*    Reflect the glob back into the domain, if needed. ---*/
      reflected_location = curr_glob->location;
      while(((left_end_exists)
        &&(reflected_location < left_end)
        ||((rite_end_exists)&&reflected_location > rite_end)))
       { if(reflected_location < left_end)
          { reflected_location = 2.*left_end
                                    - reflected_location;
            if(left_symmetric == 0)
               curr_glob->value = -curr_glob->value;
          };
         if(reflected_location > rite_end)
          { reflected_location = 2.*rite_end
                                    - reflected_location;
            if(rite_symmetric == 0)
               curr_glob->value = -curr_glob->value;
          };
```

```
            curr_glob->location = reflected_location;
          };

        curr_glob = curr_glob->next;
      }
    return(frnt_glob);
}
```

Swap

```
void Swap(this_glob,that_glob)
  struct glob *this_glob,*that_glob;
{
   float temp;

   temp = this_glob->location;
   this_glob->location = that_glob->location;
   that_glob->location = temp;

   temp = this_glob->value;
   this_glob->value = that_glob->value;
   that_glob->value = temp;

};
```

Local variables

```
 /** This is local_variables  **/
int    i = 1;

float curr_time,U,
       max_time;

char input_file_name[80],
     output_file_name[80],
     response;

struct glob *frnt_glob = NULL,
            *curr_glob,
            *back_glob,
            *frnt_grid,
            *curr_grid,
            *back_grid;
```

Rw header

```
#include<stdio.h>
#include<math.h>
#define pi 3.14159626

/*--- Define the main data structure for the double
*  linked list. ---*/
struct glob
{
  float       location,    /* where the glob is located.  */
              value,       /* the delta jump value.       */
              U,           /* function value at the point.*/
              Ux,          /* derivative at the point.    */
              Uxx;         /* second derivative.          */
  struct glob *prev,       /* pointer to previous glob.   */
              *next;       /* pointer to next glob.       */
};

/*--- Global Variable Definitions ---*/
FILE  *input,
      *U_dat,
      *Ux_dat,
      *Uxx_dat,
      *reaction_dat,
      *output;
int   left_end_exists,
      rite_end_exists,
      left_symmetric,
      rite_symmetric,
      number_of_jumps,
      number_of_globs,
      reaction_term_exists;

float alpha,                /* Diffusivity constant.     */
      left_end,             /* Left end of the domain.   */
      rite_end,             /* Right end of the domain.  */
      time_step,            /* Delta t.                  */
      U,                    /* Function value.           */
      E_reaction(),         /* Explicit reaction.        */
      Get_slope(),          /* Computes slope.           */
      N_reaction(),         /* Numerical reaction.       */
      Step(),               /* Function for step size.   */
      exact(),
      drand48();

struct glob *Add_a_glob(),
            *Compute_U(),
            *Get_input(),
            *Mod_sort(),
            *Nuke_a_glob(),
            *QuickSplit(),
            *QuickSort(),
            *Reaction(),
            *Sort_the_globs(),
            *Step_the_globs();

void        Compute_Ux(),
            Get_input_file_name(),
            Get_output_file_name(),
            Swap();

/*--- Include the library routines (user defined). ---*/
#include "./Add_a_glob"
#include "./Compute_U"
#include "./Compute_Ux"
#include "../E_reaction"
#include "./Get_input"
#include "./Get_input_file_name"
#include "./Get_output_file_name"
#include "./Get_slope"
#include "./Mod_sort"
#include "../N_reaction"
#include "./Nuke_a_glob"
#include "./QuickSort"
#include "./QuickSplit"
#include "./Reaction"
#include "./Sort_the_globs"
#include "./Step"
#include "./Step_the_globs"
#include "./Swap"
#include "./exact"
```

VITA

Gregory Scot Lindstrom

Greg received the B.S. degree in mathematics from Harding University in 1988. He is currently teaching there in the Department of Physical Science as a physics laboratory supervisor and is the primary research associate for Harding's participation in NASA's joint venture (JOVE) program through the George C. Marshall Space Flight Center performing solar physics research.

Greg is actively involved with students requiring remedial help in mathematics through Harding University's Learning Assistance Center (LAC). He leads workshops covering all aspects of mathematical fundamentals, along with conducting sessions helping students to overcome "math anxiety," with the goal of helping everyone apply the concepts learned in the classroom to everyday life. Current plans call for his "math fundamentals" to be taped as a series of half hour programs to air on the campus/community access channel.

Greg Lindstrom's permanent address is 610 E. Woodruff Street, Searcy, AR. He resides there with his wife, Janet, and daughter, Rachel.

This thesis was compiled using LaTeX at Texas A&M University.