NORTHWESTERN UNIVERSITY

Locally Adaptive Time Stepping in Numerical Simulations for Neuroscience

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Applied Mathematics

By

Richard Alexander Kublik

EVANSTON, ILLINOIS

March 2011

ABSTRACT

Locally Adaptive Time Stepping in Numerical Simulations for Neuroscience

Richard Alexander Kublik

Ever since Hodgkin and Huxley first presented their model of neuronal activity, numerical simulations have played an important role in the field of neuroscience. Early work in the emerging field of computational neuroscience led to the development of techniques for solving the problem of action potential propagation along cables and through branched structures culminating in the widespread use of the Crank-Nicolson method and ordering scheme developed by Hines and incorporated into the NEURON simulation environment. As the available numerical packages improved, the range and scale of computational simulations continues to grow.

The work presented in this dissertation departs from the conventional methods in many respects. The Crank-Nicolson scheme is abandoned in favor of the more stable Backwards Differentiation Formula, and the computational domain is divided into distinct subdomains using a simple domain decomposition scheme. Each subdomain is then updated independently using an adaptive time stepping scheme with the local time step

determined by the level of local activity. In this locally adaptive time stepping scheme, regions experiencing high levels of activity are updated with a small time step, while regions that evolve slowly are updated using a much larger time step. Unlike other applications of adaptive time stepping, where the entire domain is updated using a globally selected time step, the locally adaptive time stepping scheme focuses computational power where it is most needed: the regions of high activity.

The locally adaptive time stepping scheme described in this dissertation is not restricted to the unique problems found in computational neuroscience, but can be easily adapted to any reaction-diffusion system, and extended to higher dimensions. While there is some computational overhead due to the domain decomposition scheme and step size selection, the focused use of computational resources provides sufficient increases in computational speed to compensate, especially for simulations on large spatial domains with highly localized pockets of activity.

Acknowledgements

This dissertation comes at the end of a difficult five years and I am grateful for the support and encouragement I received from many people.

First and foremost, I'd like to thank my family, especially my wife Catherine. I'm glad you were with me to celebrate the successes along the way, and thankful you were there to cheer me up when I was struggling. Without your love and support the struggles would have been more difficult, and the successes less joyful. You were a source of strength and encouragement, and I look forward to the successes and joys that we will share in the future.

As I move on, I feel prepared to face the challenges ahead and I would like to thank my advisor, Professor David Chopp, for his advice and guidance during my time at Northwestern. We broke new ground with this project, and by observing how you approached the issues that arose during the project I've gained a sense of how to proceed into the future. In addition to the knowledge I've gained as a student, I've learned how to approach new problems and acquired skills that will help me succeed in the future. I am also grateful for your understanding and accommodation of my personal life, allowing me to live and work from Ann Arbor so that I could be with Catherine.

I would also like to thank the professors and students in the department. Thank you to the many students I've shared an office with for the entertaining discussions over the years. Thank you Professor Riecke for your interest in my work and encouragement

along the way. Thank you Professor Bayliss for helpful discussions during the early development stages of my project and your encouragement throughout. Thank you to my committee members, Professors Kath and Spruston for your input and assistance with the neuroscience aspects of my project.

While living in Ann Arbor, I frequently returned to Evanston for research meetings and TA duties. I am grateful to Linda for welcoming me into her home on these occasions and for her continued friendship.

Table of Contents

List of Figures

CHAPTER 1

Introduction

In the early days of scientific computing, researchers were limited by the processing power of the machines available to them. As a result, new methods and algorithms were developed to most effectively use the limited computational power available, leading to higher order methods, and improvements in efficiency. More recently, advances in computer hardware have resulted in machines of ever increasing power, and it is now possible to run many simulations using a naive approach. However, the scale and complexity of the simulations that are of interest have kept pace. Thus in spite of the gains in raw computational power, there is still great value in the development of algorithms that can exploit this power to the fullest.

There are many strategies for decreasing the computational time of numerical simulations, and often the choice of method is dependent on the problem. Adaptive time stepping schemes have proven beneficial in solving ordinary differential equations and have been applied to partial differential equations as well. With the increasing prevalence of large-scale parallel computer architectures, domain decomposition schemes are utilized to spread the work over a large number of processors. There are also classes of problems where the time step is restricted in a small part of the spatial domain, when the majority of the system could be evolved with a larger step. For cases where the step size restriction remains stationary, domain decomposition schemes using different (fixed) time steps in each subdomain have increased the speed of computations without sacrificing accuracy in

the final solution. The algorithm presented here builds on these strategies and combines ideas from all of them.

In systems where activity is localized in both space and time, a small, ever changing fraction of the computational domain experiences levels of high activity while the remainder of the system evolves much more slowly. In this dissertation we present a locally adaptive time stepping algorithm that can be used in these situations. After dividing the system into small subdomains, an adaptive time stepping scheme is applied to each subdomain independently. In this way, computational resources are focused where they are most needed, and the computational cost of simulations scales with the level of activity rather than the physical size of the problem.

Although our algorithm is applicable to any system of reaction-diffusion equations, it was motivated by, and developed in the context of, simulations for computational neuroscience. In the following sections, a brief introduction to neurophysiology is presented, culminating in the mathematical model of Hodgkin and Huxley for the propagation of action potentials through nerve fibers. The remainder of this introductory chapter describes advances in simulation techniques for the Hodgkin-Huxley cable equation and further explores the computational strategies introduced above. In Chapter 2 we present our method for locally adaptive time stepping, and demonstrate its benefits in Chapter 3.

1.1. Neural Physiology

In this section, we present a simplified version of the basic ideas necessary to understand the computational model used to describe the electrical signals found in neurons.

The interested reader is referred to one of the many introductory textbooks for more details (see for example[**2, 16**]).

1.1.1. Morphology

Neurons are distributed throughout the brain and body, and though they have different functions and shapes, all neurons have the same basic structure. Figure 1.1 shows a representative example of a pyramidal cell taken from the CA1 region of rat hippocampus, stained and reconstructed on a computer. This cell, like all neurons, has a highly

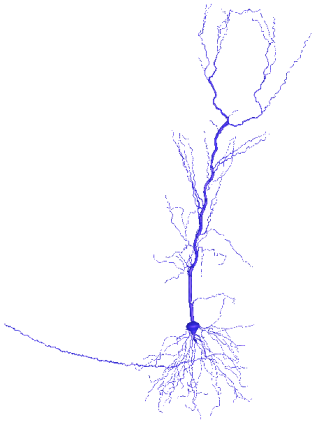


Figure 1.1. **A typical neuron.** *This pyramidal cell, from the CA1 region of rat hippocampus, shows features common to all neurons. The highly branched structure is mainly composed of dendrites, which receive inputs through synapses and produce a corresponding electrical signal. The main cell body, called the soma, integrates these inputs and when the combined input to the soma is sufficiently large, the soma generates an action potential. The action potential travels along the axon (the long branch to the left of the cell above) to relay the signal to other cells.*

branched tree-like structure, and is specialized to generate electrical signals in response to chemical inputs and transmit them to other cells [**16**]. The branching dendritic tree receives chemical signals through synaptic receptors. The chemical signal is converted to a small electrical impulse that propagates within the tree structure. The main cell body,

called the soma, integrates the impulses that reach it and when the combined input to the soma is large enough, the soma generates a large signal, called an action potential, that is transmitted along the axon. The axon is connected to neighboring cells through the synapses, and when the action potential reaches a synapse, the electrical signal is converted to a chemical signal that is received by the post-synaptic cell.

For the purposes of computational modeling, neurons are considered to be a branched structure comprised of one-dimensional cables. Typically the dendrites are long and slender. In the model cell shown in Figure 1.1, aside from the large cell body, branches range in length from 4.5 to 250 μm, with radii between 0.2 and 1.2 μm, though the actual cell may have smaller structures that are too fine to be resolved. As a result, it is reasonable to model the geometry of the cell as a network of one-dimensional branches.

1.1.2. Electrical Properties of Neurons

Neurons, and many other cells, contain large numbers of dissociated ions in varying concentrations. Some of these ions are positively charged (K^+, Na^+, Ca^{2+}), while others are negatively charged (Cl^-). These ions are found in varying concentrations within the cell, as well as in the extracellular space. Cellular membranes are composed of phospholipids, molecules composed of a hydrophilic head and a hydrophobic tail. Due to the varying affinities for water within the molecules, these phospholipids organize into bilayers, with the hydrophobic tails protected by the hydrophilic heads, as shown in Figure 1.2. Within the cell, the charged ions exert electrostatic forces on each other with like charges repelling each other. Positively charged ions build up along the inside of the cellular membrane while negatively charged ions collect along the outside of the membrane. The lipid bilayer

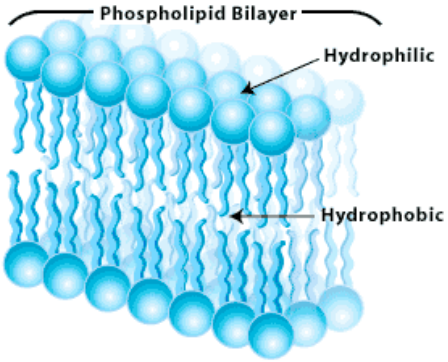


Figure 1.2. **Phospholipid Bilayer** *The hydrophilic heads of the molecules are exposed to the inside and outside of the cell, while the hydrophobic tails are kept together in the bilayer. (Image taken from http://www.bioteach.ubc.ca/Bio-industry/Inex/)*

is impermeable to ions, electrically insulating the inside of the cell from the extracellular space. This insulating layer, combined with the buildup of charge on either side of the membrane causes the cell membrane to act as an electrical capacitor, the strength of which depends on the amount of charge stored and the potential difference across the membrane according to the relationship

$$C = Q/V. \tag{1.1}$$

In this equation, C is the capacitance, or strength of the capacitor, Q is the amount of charge stored on the cellular membrane, and V is the potential difference across the membrane. By convention, the membrane potential is computed as the extracellular potential subtracted from the intracellular potential.

Because charge is defined as the time integral of the current through a capacitor, we can use (1.1) to determine the current required to charge the capacitor. Solving for Q we

obtain,

$$Q = CV, \tag{1.2}$$

and differentiation with respect to time, assuming constant capacitance gives

$$\frac{dQ}{dt} = I_C = C\frac{\partial V}{\partial t}. \tag{1.3}$$

Due to the non-uniform distribution of ions in the cell and the extracellular medium, the

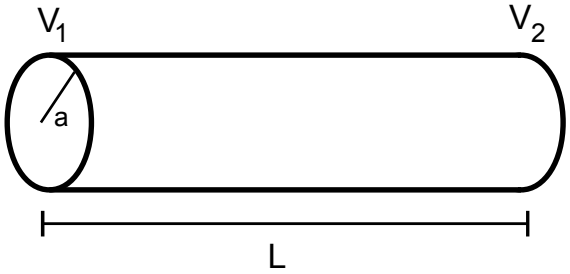


Figure 1.3. **Current in a uniform cable.** *Unequal voltages at each end of the cable (V_1 and V_2) cause a current through the cable, as given in* (1.4)*. The current is proportional to the difference in voltage at each end and the cross-sectional area, with uniform radius a, and inversely proportional to the length of the cable, L, and the specific axial resistance, R_a.*

potential difference across the membrane is not the same everywhere in the cell. These spatial differences cause ions to travel within the cell in order to equalize the potentials. The current created by the movement of these ions within a small section of the cell is proportional to the voltage difference at the ends of the section and inversely proportional to the resistivity of the intracellular fluid, giving $I_L = (V_2 - V_1)/R_L$, where R_L, the total resistance, is proportional to the length of the section, L, and the specific axial resistance, R_a, and inversely proportional to the cross-sectional area of the section with uniform radius a. Thus, the current created by the flow of ions along the uniform cylindrical cable

shown in Figure 1.3 is

$$I_L = -\frac{\pi a^2}{R_a L}(V_2 - V_1), \tag{1.4}$$

where by convention, positive current flows in the direction of increasing x. In the limit where L approaches zero, the longitudinal current at a point in the cable is given by:

$$I_L = -\frac{\pi a^2}{R_a}\frac{\partial V}{\partial x}. \tag{1.5}$$

In order to know how the membrane potential at a given point changes over time, we must consider all the currents that contribute. Current can flow longitudinally into the section from neighboring sections, and can also flow out of the cell through the membrane. These membrane currents could be due to an external stimulating electrode, or the ionic channels that will be discussed shortly. For the moment, we treat the sum of these membrane currents as a single current, I_m, expressed as a current per unit surface area of membrane. Figure 1.4 shows a small section of cable, of length Δx, along with the relevant currents.

The cable equation is derived from the conservation of charge flux, or current. In the cable section shown in Figure 1.4 , the radius $a = a(x)$ is variable and we define a_l and a_r to be the radius at the left and right ends of the section. Within this section of cable, the net flux of charge must balance, with the sum of the currents into the section of cable equal to the outward capacitive current required to charge the membrane. This balance of current gives the equation:

$$2\pi a \Delta x C \frac{\partial V}{\partial t} = -\left(\frac{\pi a_l^2}{R_a}\frac{\partial V}{\partial x}\right)\Bigg|_{\text{left}} + \left(\frac{\pi a_r^2}{R_a}\frac{\partial V}{\partial x}\right)\Bigg|_{\text{right}} - \pi(a_l + a_r)\Delta x I_m, \tag{1.6}$$

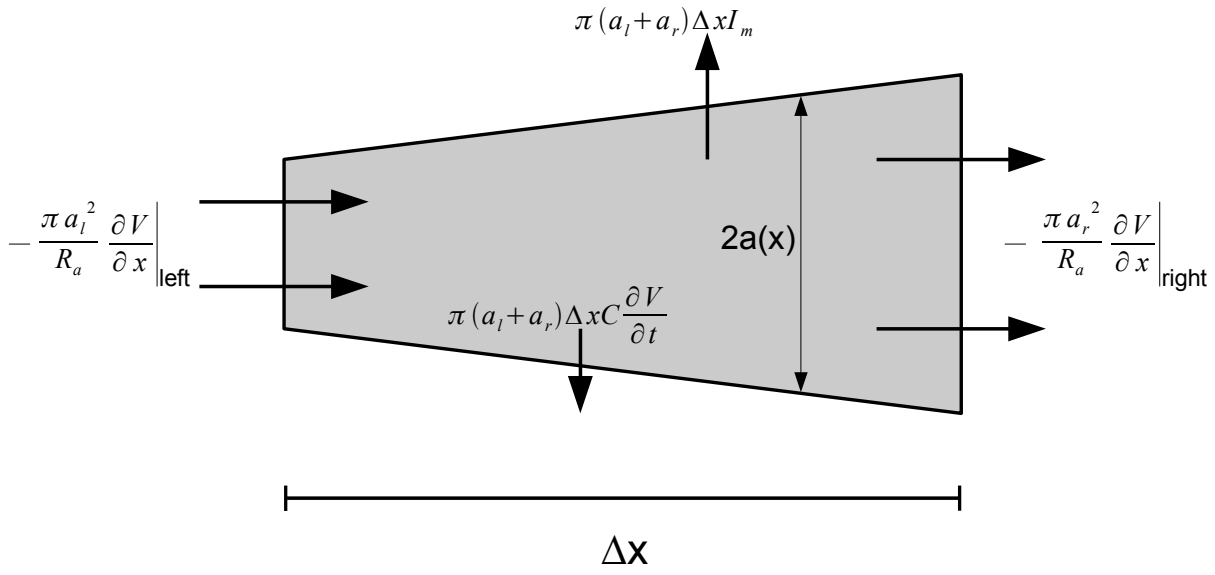


Figure 1.4. **Section of cable used to derive the Cable Equation.** *The various currents that contribute to the membrane potential are depicted, with the arrows showing the direction of positive current. The membrane currents are expressed as current per unit surface area of a cylinder of radius a.(Adapted from Dayan & Abbot Figure 6.6* [**16**]*).*

where the membrane current, I_m has been scaled by the surface area of the section of cable. Dividing both sides by $\pi(a_l + a_r)\Delta x$ and taking the limit as $\Delta x \to 0$, we obtain

$$C\frac{\partial V}{\partial t} = \frac{1}{2\pi a}\frac{\partial}{\partial x}\left(\frac{\pi a^2}{R_a}\frac{\partial V}{\partial x}\right) - I_m(V). \tag{1.7}$$

This is the cable equation, and much work was done by Rall [**50, 51, 52**] to apply it to the electrical propagation within neurons.

While the lipid bilayer is impermeable to ions, the cell membrane contains many protein structures, including gated pores or channels that allow specific ions to pass through. Under normal circumstances these channels tend to be closed, preventing the flow of ions. However, conformational changes to the protein structure as a result of voltage changes or the binding of some ligand to the channel cause them to open. The

current resulting from the flow of ions through these channels are included in the cable equation as part of the last term, $I_m(V)$, in (1.7).

When channels are open, the cell membrane is permeable to specific ions, and these ions are pushed through the channel by competing forces. Negative membrane potentials attract positively charged ions into the cell, while positive membrane potentials push them out. Additionally, the neuron is equipped with ion pumps that create differences in concentration inside and outside the cell, keeping the concentrations of Na^+ and Ca^{2+} higher outside the cell, and the concentration of K^+ higher inside, resulting in flux of ions due to the concentration differences. The net flow of ions is determined by both the concentration and electrical gradients. The membrane voltage where these effects are equal resulting in no net flow of ions is called the equilibrium, or reversal, potential. The current into the cell through a given ion channel is modeled by the equation

$$I_{\text{ion}} = g(V,t)(V - E_{\text{ion}}), \tag{1.8}$$

where E_{ion} is the reversal potential, and $g(V,t)$ is the conductance through the channel.

In a series of papers, Hodgkin and Huxley[**28, 29, 30, 31, 32**] characterized the Na^+ and K^+ channels found in the giant axon of the squid. Using the voltage clamp technique they were able to hold the membrane voltage fixed, and by measuring the current required to maintain a constant voltage, the corresponding ionic currents could be inferred. Hodgkin and Huxley eliminated any spatial movement of ions through use of the space clamp, which kept the entire axon isopotential, and adjusted the ionic concentrations in the bath solution in order to isolate specific currents.

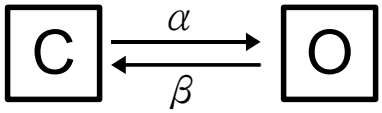


Figure 1.5. **Two State Channel Model.** *Each gate moves from the closed to open state with rate α and from open to closed with rate β.*

To determine the precise form of the voltage-dependent conductances in (1.8), Hodgkin and Huxley fit their mathematical model to the observed evolution in the ionic conductances following changes in membrane voltage. In the case of the potassium channel, the observational data was best fit by the current model

$$I_K = \overline{g_K} n^4 (V - E_K), \tag{1.9}$$

where $\overline{g_K}$ is the maximal conductance observed, and n takes values between 0 and 1. Hodgkin and Huxley interpreted the form of the potassium conductance in a physiological way, suggesting that "potassium ions can only cross the membrane when four similar particles occupy a certain region of the membrane [**29**]." This has since been reinterpreted as a set of conformational changes within the proteins that make up the potassium channel. The potassium channel is considered to have four "gates." Each gate has an "open" and "closed" configuration, and all four gates must be in the open state to allow ions to flow through. In this modern interpretation, equation (1.9) represents the cumulative effect of many channels.

In the voltage-gated potassium channel, each gate switches from the closed to open state with a rate $\alpha(V)$, and from the open to closed state with a rate $\beta(V)$, as shown in Figure 1.5. Again if we consider the macroscopic picture, the fraction of gates in the open

state evolves according to the ODE

$$\frac{dn}{dt} = \alpha(1-n) - \beta n. \tag{1.10}$$

The particular form of α and β were determined during the fitting of the model to observations, and are detailed in the next section.

The sodium channel is modeled in the same manner, and also contains four gates, however they do not all behave in the same way. The sodium channel contains three activation gates that are in the closed configuration at low voltage, and open as the membrane potential rises. The fourth gate is open for low voltages and closes as the voltage increases. This is called inactivation, and is slower than activation. The fraction of open gates for each type is given by m and h respectively, and are modeled in a similar fashion to the n gate in (1.10). The current through the sodium channel is given by

$$I_{Na} = \overline{g_{Na}} m^3 h (V - E_{Na}). \tag{1.11}$$

Hodgkin and Huxley also included a non-specific ionic current, commonly referred to as the "leak" current. This current,

$$I_{\text{leak}} = g_{\text{leak}}(V - E_{\text{leak}}), \tag{1.12}$$

with a constant conductance, is meant to represent the net effect of all the remaining channels that are not explicitly modeled.

During experimental work, it is common to inject current into a cell. This current injection may be used to mimic synaptic input or to stimulate the cell to produce an action

potential. This current injection, located at the point x_0 is an inward current included in the total membrane current I_m given in (1.6) as $-I_e\delta(x-x_0)$.

1.2. The Hodgkin-Huxley Cable Equation

Having described all its components, we can now present the Hodgkin-Huxley cable equation:

$$C_m\frac{\partial V}{\partial t}=\frac{1}{2\pi a}\frac{\partial}{\partial x}\left(\frac{\pi a^2}{R_a}\frac{\partial V}{\partial x}\right)-I_m(V) \tag{1.13a}$$

where

$$I_m(V)=\overline{g_{\mathrm{Na}}}m^3h(V-E_{\mathrm{Na}})+\overline{g_{\mathrm{K}}}n^4(V-E_{\mathrm{K}})+g_{\mathrm{leak}}(V-E_{\mathrm{leak}})-\frac{I_e}{2\pi a}\delta(x-x_0) \tag{1.13b}$$

$$\begin{aligned}\frac{dm}{dt} &= \alpha_m(V)(1-m)-\beta_m(V)m\\ \frac{dn}{dt} &= \alpha_n(V)(1-n)-\beta_n(V)n\\ \frac{dh}{dt} &= \alpha_h(V)(1-h)-\beta_h(V)h\end{aligned} \tag{1.13c}$$

$$\begin{aligned}\alpha_m(V)&=\frac{0.1(25-V)}{\mathrm{e}^{(25-\mathrm{V})/10}-1} & \beta_m(V)&=4\mathrm{e}^{-\mathrm{V}/18}\\ \alpha_n(V)&=\frac{0.1(10-V)}{\mathrm{e}^{(10-\mathrm{V})/10}-1} & \beta_n(V)&=0.125\mathrm{e}^{-\mathrm{V}/80}\\ \alpha_h(V)&=0.07\mathrm{e}^{-\mathrm{V}/20} & \beta_h(V)&=\frac{1}{\mathrm{e}^{(30-\mathrm{V})/10}+1}\end{aligned} \tag{1.13d}$$

In these equations, C_m is the membrane capacitance, R_a is the axial resistivity, a the radius, $\overline{g_{\mathrm{Na}}}$ and $\overline{g_{\mathrm{K}}}$ are the maximal conductances for the sodium and potassium channels and g_{leak} is the conductance of the non-specific "leak" channel. E_{Na}, E_{K}, E_{leak} are the ionic reversal potentials associated with each channel type. The final term in (1.13b) represents the current (I_e) due to an external stimulating electrode at location x_0. Here the electrode current has been scaled by the membrane surface area. The variables m, n, and h are dimensionless and evolve according to (1.13c). Finally, (1.13d) gives the form of $\alpha(V)$, and $\beta(V)$ used in (1.13c). $I_m(V)$ is a current and each of the terms in (1.13b) model the flow of current through the membrane due to the movement of ions through the membrane ion channels. The Hodgkin-Huxley model presented here provides a framework that has been extended to model many different channel types, leading to additional equations similar to those given in (1.13c & 1.13d) and extra membrane currents included into $I_m(V)$ [**17, 37, 42, 49**].

1.3. Numerical Methods for Neuroscience

When Hodgkin and Huxley developed their model, they relied on hand-crank calculators to solve the equations numerically. Since that time, advances in computer technology and numerical methods, together with the development of simulation packages like NEURON [**8, 26, 27**] and GENESIS [**5**], have allowed for simulations of increasingly complex networks of cells. Many of the early numerical studies employed explicit methods, Forward Euler [**18, 22**], Improved Euler [**24, 23**] or Runge Kutta schemes [**33, 35**], but these methods become unstable for large time steps. To guarantee a numerically stable solution, the step size is limited by the Courant-Friedrichs-Lewy (CFL) condition [**12**]. The CFL

condition, combined with the limited computational power available, meant that the sort of simulations frequently done today were impossible at the time. Indeed, Fitzhugh [**18**] performed simulations on a single axon 26.75 mm in length using a spatial grid of 0.25 mm (resulting in 107 computational grid points) and a time step of 7.5×10^{-4} ms. The computers used for the study required 1.5 hours of computational time to evolve the system for 1.392 ms, not even enough time to form a complete action potential curve.

To avoid the stability restrictions of explicit methods, implicit methods have been developed. These methods are much more stable, and in many cases provide computationally stable solutions for any sized time step. It should be noted that even though implicit schemes are stable for large time steps, accuracy requirements limit the step size that can be taken. Cooley and Dodge [**11**], used the second order Crank-Nicolson method [**48**] for the diffusion term and combined it with an iterative procedure for the non-linear reaction terms. Other groups followed this approach, including Moore *et al.* [**34, 44, 45, 46, 57, 53**] who extended the work of Cooley and Dodge to cables with non-uniform shapes. Considering the cell axon as a variable radius cable, their simulations assumed piecewise constant radius. This separated the cable into sections of uniform cable, coupled through boundary conditions. The sections were then updated sequentially, using boundary data approximated through an explicit method. The explicit update led to instabilities at the boundaries where sections connected, although the instabilities could be significantly reduced by alternating the order of update for the connected sections throughout the simulation. This approach can also be applied to junctions of multiple branches and facilitates simulations of systems with arbitrary branching patterns.

Hines [**25**] presented three innovations that dramatically improved the efficiency of simulations, especially on branched structures. In Hines' method, the entire cell is treated as a single large system of equations that is solved simultaneously at each step in the evolution of the system. Hines employs a variant of the Crank-Nicolson scheme, where the nonlinear reaction terms are updated at the midpoint of the time step. This is a three step process: the gating variables in (1.13c) are updated using the trapezoidal rule to time $t^{n+1/2}$, followed by an update of the voltage to time $t^{n+1/2}$ using the Backward Euler method. Finally the voltage is extrapolated to time t^{n+1}. By alternating the update of the reaction and diffusion terms in this way, Hines maintains the second order accuracy of the Crank-Nicolson method without the need for an expensive iterative procedure. Additionally, Hines showed that by numbering the branches and constituent compartments of the structure in a specific way, the resulting linear system for the voltage in the entire branched cell can easily be made almost tri-diagonal and solved as efficiently as in the case of an unbranched cable. Since the rate constants, the α's and β's in (1.13d), are only functions of the membrane voltage, they can be computed prior to the simulation. Hines demonstrated that it is more efficient to pre-compute these values for a range of voltages and interpolate needed values from those saved than to compute them at each update of the simulation. These innovations became the foundation of the NEURON simulation environment, a package that has become one of the most popular simulation tools in the field of computational neuroscience.

1.4. Adaptive Time Stepping

In many systems, periods of high activity alternate with intervals of low activity and in cases such as these, adaptive time stepping schemes have been effectively utilized to control the simulations. With each step, all numerical methods introduce errors into the solution. The relative size of these errors is determined by the order of the method, with higher order methods contributing smaller errors. The total accumulated error in the final solution is approximately equal to the average error at each step times the number of integration steps. In adaptive simulations, the user defines an acceptable level of error in the final solution which is then translated into an error tolerance for each step. In each update step, the local error in the computation is estimated and compared to this prescribed error tolerance. If the error is below the tolerance, the step size is increased for future updates, but if the error is above the tolerance, the step size is decreased and the most recent update is recomputed to obtain a solution with the required accuracy. Estimates of the error can be obtained in a number of ways: one method is to compare the result of two methods of different order, as in the Matlab® [**41**] function ODE45 which compares the 4th and 5th order Runge-Kutte methods, or by comparing the result with different sized steps. Other commonly used methods compare the result of one step of size Δt with the solution obtained through two steps of size $\Delta t/2$, or compare the result of an explicit method with an implicit method [**56**].

Using one of the error estimators, adaptive time stepping schemes adjust the step size used to maintain a constant level of error. During periods of low activity, when the solution is changing slowly, a larger step can be taken than when the solution changes rapidly, as shown in Figure 1.6. When using a fixed step integration method, one considers

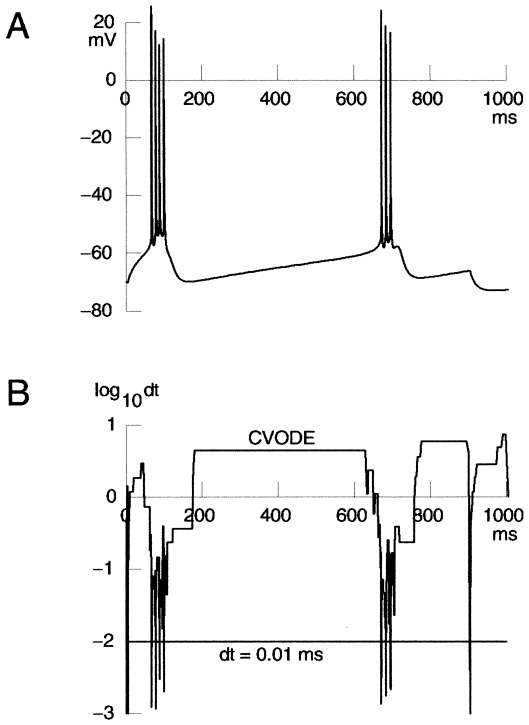


Figure 1.6. **Adaptive Time Stepping.** *The plot in (A) shows the membrane voltage in a bursting cell, from a simulation done in NEURON. The plot in (B) shows the time steps used for simulations with CVODE, an adaptive time stepping scheme, and Hines' fixed step variant of the Crank-Nicolson method. Notice the large steps taken by CVODE in the period between bursts. (Plots taken from Hines and Carnevale 2001* [**27**]*, Figure 9).*

the trade-off between speed of computation and accuracy of the solution. To accurately resolve the fast evolution of a system, the entire simulation must be computed using a time step that is sufficiently small to capture the fastest evolution. To obtain comparable accuracy as an adaptive method, the fixed step scheme must be run using the smallest time step taken by the adaptive method, making the adaptive stepping scheme more efficient by comparison.

The NEURON simulation environment includes an adaptive time stepping scheme, based on the CVODE algorithm that will be described shortly. This was included to

provide error control over the solution in the most efficient way. However, in many cases, the level of accuracy provided by CVODE is not necessary, and simulations can be run faster using one of the fixed-step methods.

A natural extension of variable step methods are the variable step variable order methods. The widely used package by Gear [**19, 20**] is an example of such a method. This package provides two integrators, one based on the Adams-Bashforth-Molton methods, and the second based on the Backwards Differentiation Formula (BDF) methods for stiff systems. The basic algorithm remains the same in both cases: when using the method of order p, the solution is integrated with a fixed time step for $p+1$ steps, after which a change of step size and order are contemplated. Based on the local error in the solution, the order and step size are chosen to maximize the step size used. Gear's algorithm is based on the families of linear multistep methods, and consequently requires solution values at a number of past time points. When the step size and order are changed, the history data for the new step size is obtained by interpolation of the known history data and the new order selected is restricted to one of $p-1,\ p$ and $p+1$.

Gear's method allows for variation in the order and step size, but it is not a variable step method in the strictest sense, as fixed step methods are used and require equally spaced integration points. Additionally, the repeated interpolations when the step size and order are changed lead to instabilities in some cases. These instabilities lead to the development of the CVODE package [**6, 7, 10**]. As in Gear's method, CVODE uses either the Adams-Bashfoth-Molton or BDF methods, though the fixed step variants have been replaced by the variable coefficient schemes that allow for arbitrarily spaced solution points, eliminating the instabilities that can occur with Gear's method. In CVODE, step

sizes are adjusted at each time point to provide the optimal time step for the given error tolerance. As in Gear's method, changes in order are considered following $p + 1$ steps using the order p method, and the order can be changed by at most 1. Both methods use a predictor-corrector approach: the solution at the new time is predicted using an explicit method, either Adams-Bashforth, or through extrapolation when using the BDF methods, and the final solution is found through iteration of the implicit scheme, either Adams-Molton or BDF. Gear's method and CVODE implement a different method for error estimation from those mentioned previously. The error estimate is obtained from a comparison of the explicit predictor and the final iteration of the implicit scheme.

When using a fixed-step method, there is no guaranteed level of accuracy in the final solution, and the step size is chosen to provide a reasonable solution while avoiding excessive computational cost. In developing the locally adaptive time stepping algorithm presented in this dissertation, we chose to provide a similarly reasonable solution with the lowest computational cost possible. In our method we do not control the error in each step of the computation, but rather adjust the time step size in response to how quickly the solution is evolving in time.

1.4.1. Application to Partial Differential Equations

Many methods for solving partial differential equations are based on ODE methods applied to the spatial domain through the method of lines. In this method, the spatial domain is discretized first, resulting in a set of coupled ODEs, one equation for each spatial grid point. Adaptive methods can be used for solving PDEs, and the simplest application relies on a globally chosen time step. In these cases, the time step for the entire system

is chosen to be sufficiently small to accurately resolve the solution where it is changing most rapidly. For these methods, the globally selected time step means that activity in a small region of space will cause the entire system to be evolved with a small time step, even though most of the system could be accurately resolved with a much larger step. In the next section, we present algorithms for dealing with localized restrictions on step size.

1.5. Domain Decomposition Schemes

In some cases, domain decomposition schemes have been effective in the solution of PDE problems. The spatial domain is divided into smaller subdomains which are updated independently and coupled through boundary conditions. As the subdomains are independent, it is also possible for them to be updated in parallel, and many of these schemes have been advanced to take advantage of parallel computer architecture.

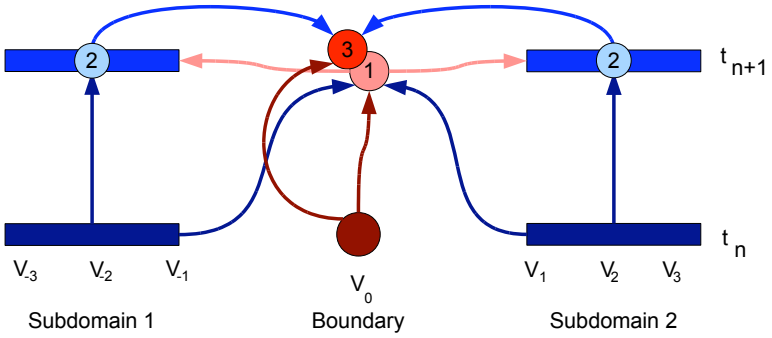


Figure 1.7. **(Corrected) Explicit-Implicit Domain Decomposition.** *A predicted value (1) for the boundary at time t_{n+1} is found using an explicit method. Following this, the subdomains are updated using an implicit method and the predicted value of the boundary to obtain the solution at t_{n+1} (2). In the CEIDD schemes the boundary values are corrected using an implicit scheme and the newly computed solutions in the subdomains (3).*

A commonly used domain decomposition method is the Explicit-Implicit Domain Decomposition (EIDD) scheme [**1, 15, 36**]. The spatial domain is split into non-overlapping

subdomains together with the dividing interface, as shown in Figure 1.7 for a one dimensional domain. The interface is updated with an explicit method, followed by an update of the interior of the subdomains using an implicit method and the just computed interface values as boundary data. It was found that the stability of the EIDD schemes could be improved by following the implicit update of the subdomains with an implicit correction update of the interface values, leading to the so-called Corrected Explicit-Implicit (CEIDD) methods [**58, 63, 64**].

Rempe and Chopp [**54**] combined a CEIDD scheme with Hines' variant of the Crank-Nicolson method to solve the Hodgkin-Huxley cable equations represented here by the simplified reaction-diffusion system

$$\frac{\partial V}{\partial t} = A\frac{\partial^2 V}{\partial x^2} - \Gamma V \tag{1.14a}$$

$$\frac{d\Gamma}{dt} = \alpha(V)(1-\Gamma) - \beta(V)\Gamma. \tag{1.14b}$$

Discretization in space gives the coupled set of ordinary differential equations

$$\frac{dV_i}{dt} = \lambda V_{i-1} - 2\lambda V_i + \lambda V_{i+1} - \Gamma_i V_i, \tag{1.15a}$$

$$\frac{d\Gamma_i}{dt} = \alpha(V_i)(1-\Gamma_i) - \beta(V_i)\Gamma_i, \tag{1.15b}$$

where $\lambda = A/\Delta x^2$.

The update of the system is done through the following steps. First, the reaction, Γ, is updated at each grid point to time $t^n + \Delta t/2$ using the trapezoidal rule:

$$\frac{\Gamma^{t^n+\Delta t/2} - \Gamma^{t^n-\Delta t/2}}{\Delta t} = \alpha(V(t^n)) - (\alpha(V(t^n)) + \beta(V(t^n)))\,\frac{\Gamma^{t^n+\Delta t/2} + \Gamma^{t^n-\Delta t/2}}{2} \tag{1.16}$$

Next, the boundary is updated to time $t^n + \Delta t/2$ using the explicit Forward Euler method:

$$V_0^* = V_0^{t^n} + \frac{\Delta t}{2}\left(\lambda V_{-1}^{t^n} - 2\lambda V_0^{t^n} + \lambda V_1^{t^n} - \Gamma_0^{t^n+\Delta t/2} V_0^{t^n}\right), \tag{1.17}$$

followed by a Backward Euler update in each subdomain using the updates:

$$\lambda V_{i-1}^{t^n+\Delta t/2} - \left(2\lambda + \frac{2}{\Delta t} + \Gamma^{t^n+\Delta t/2}\right) V_i^{t^n+\Delta t/2} + \lambda V_{i+1}^{t^n+\Delta t/2} = \frac{2}{\Delta t} V_i^{t^n+\Delta t/2},$$

in the interior of each subdomain, and

$$\begin{aligned}
\lambda V_{-2}^{t^n+\Delta t/2} - \left(2\lambda + \frac{2}{\Delta t} + \Gamma^{t^n+\Delta t/2}\right) V_{-1}^{t^n+\Delta t/2} &= -\lambda V_0^* + \frac{2}{\Delta t} V_{-1}^{t^n+\Delta t/2}, \\
-\left(2\lambda + \frac{\Delta t}{2} + \Gamma^{t^n+\Delta t/2}\right) V_1^{t^n+\Delta t/2} + \lambda V_2^{t^n+\Delta t/2} &= -\lambda V_0^* + \frac{2}{\Delta t} V_1^{t^n+\Delta t/2}.
\end{aligned}$$

for the endpoints of each subdomain. In these equations, the subscripts on the voltage correspond to the subscripts found in the compartmentalization found in Figure 1.7. The solution at the boundary is then corrected at time $t^n + \Delta t/2$, using the Backward Euler method

$$V_0^{t^n+\Delta t/2} = \frac{\frac{2}{\Delta t} V_0^{t^n} + \left(\lambda V_{-1}^{t^n+\Delta t/2} + \lambda V_1^{t^n+\Delta t/2}\right)}{\frac{2}{\Delta t} + 2\lambda + \Gamma^{t^n+\Delta t/2}}. \tag{1.18}$$

Finally, the solution at all grid points is advanced to time $t^n + \Delta t$ through extrapolation:

$$V_i^{t^n+\Delta t} = 2V_i^{t^n+\Delta t/2} - V_i^{t^n}. \tag{1.19}$$

Other domain decomposition schemes have also been applied to the branched structures found in neuroscience problems. Mascagni [**39, 40**] utilized the Backward Euler method for time integration and developed an exact domain decomposition scheme for passive (linear) cables. In his method, the tree structure is separated into smaller computational units: sections of unbranched cable and the nodal points where the sections connect. In each of the cable sections, the system of equations is tri-diagonal and is uncoupled from all the other sections, obtaining boundary data from the neighboring nodal points. In Mascagni's algorithm, two tri-diagonal systems are solved for each section: one with the original right hand side with the boundary condition that the voltage at the nodal point (V_N) is zero, and the second with the right hand side set to zero and the boundary condition $V_N = 1$. Following this, the solution in the section is a linear combination of the two computed solutions involving the yet unknown value at the node. The nodal voltage is obtained as part of the solution to a dense system of equations involving all nodal points which couples the system together. Unlike most domain decomposition schemes that use approximate solutions on sections of the domain to build an approximate solution on the whole domain, this algorithm uses the linear superposition of solutions to linear systems to construct the exact solution in the full domain from the exact solution in the subdomains.

1.5.1. Subcycling Algorithms

The domain decomposition techniques described above allow the computational cost of solving PDEs to be split among many processors. However, all subdomains are still updated with the same time step size, leading to unnecessary computations in regions

with little activity. In many problems found in structural engineering, the computational domain is composed of structural components in a fluid media. When the systems are studied using a finite element method and an explicit time stepping scheme, the stability limit in the structure mesh is much smaller than that for the media mesh [**4**]. In cases such as these, mixed methods of integration, where different regions of the domain are updated with different time stepping schemes, subcycling methods utilizing different step sizes in different subdomains have been proposed. Many of the subcycling methods require the ratio of step sizes between subdomains to be integers, although non-integer ratio methods have also been proposed [**59**]. Belytschko *et al.* [**47, 3, 62**] use explicit time stepping schemes and update the regions with largest step size first. As the regions with smaller time steps are updated, boundary conditions are obtained from the larger step regions through linear interpolation. This method is shown to be stable for reasonable ratios in step sizes between the neighboring regions of the system.

In the algorithm presented by Daniel [**13**], two regions S and L with stepsizes Δt and $r\Delta t$ respectively are updated from time T to $T + r\Delta t$ in three distinct phases. First the region S is updated $r/2$ steps using as boundary data the computed values in region L at time T. Next, region L is updated with a single step to time $T + r\Delta t$ using the last computed values in S for boundary data. Finally the region S is updated another $r/2$ steps to time $T + r\Delta t$ using the solution in L at the final time for boundary data.

Smolinski and Wu present explicit [**61**] and implicit [**60**] subcycling algorithms. In the explicit method, the regions with the smallest step size are updated first. Contrary to other methods, where boundary conditions are assumed piecewise-constant or are obtained through interpolation, the boundary points are updated to the intermediate times

required. In the implicit case we again have two regions S and L as described earlier with an integer ratio of step sizes r. An update from time T to time $T + 2r\Delta t$ is considered a complete cycle. The region S is updated first, a total of $2r$ steps to the end of the cycle, followed by 2 update steps of the region L. In both cases, boundary data is assumed to be piecewise constant, and is taken to be the most recently computed value available. This method, using the trapezoidal rule in each region is unconditionally stable, and provides reasonable accuracy provided the ratio of step sizes and the total cycle time are not too large.

1.5.2. Spatial Adaptation

In neural simulations, it is common for large regions of the network to be at or near equilibrium with only a small region expressing any activity. Only the regions experiencing significant activity need to be updated with a small time step, while the remainder of the cell could potentially be updated with a larger step. However, unlike structural mechanics problems, the subdomain requiring small time steps is constantly changing.

For their spatially adaptive algorithm, Rempe and Chopp [**54, 55**] combined a CEIDD scheme with Hines' variant of the Crank-Nicolson method as described previously. Unlike the exact domain decomposition used by Mascagni, the CEIDD scheme Rempe and Chopp employ decouples the sections of the cell, with the solutions at the nodes, or branch points, only depending on the solutions in the sections adjoining the node. With the cell uncoupled, spatial adaptivity is relatively straight-forward: sections experiencing significant deviation from equilibrium are updated with the underlying fixed time step method,

either Backward Euler or Hines' variant of Crank-Nicolson. As the sections return to rest they are inactivated and no longer updated.

In this algorithm, the nodal grid points take on a controlling role, monitoring each of the connected sections for activity and in turn activating the remaining sections in time to ensure an accurate solution. When a section, k, is activated, any inactive node connected to it is placed into a "listening" state during which the voltage in the node is not updated in time, and boundary data for the connected sections is computed to maintain conservation of current at the node. While in this listening state, the node monitors the activity in the section,

$$A_k = max\left\{\left|\frac{\partial V^*}{\partial t}\right|, \left|\frac{dm_g}{dt}\right|\right\}, \tag{1.20}$$

where V^* is the voltage in the section, scaled to be between 0 and 1, and m_g are the gating variables present in the model. When the activity in the section is above a user specified tolerance, τ_A, the node enters the active state and the algorithm begins computing updates of the solution. When the node is activated, all the connected sections are activated along with it, and the nodes attached to these sections are placed into the listening state and the process repeats.

During the update step, the activity in each section is monitored to determine when the section should be taken out of the active state. A section is considered inactive when two conditions are met. The current into the section from either end must be below a specified tolerance, τ_I, and the membrane potential everywhere in the section must be within the tolerance τ_V of the resting membrane potential. If both of these conditions are satisfied, the section is inactivated: the voltage and all gating variables are set to

their resting values, and the algorithm no longer computes updates in the section. If an active node senses that a connected section has been inactivated, it also leaves the active state, returning to the listening mode. Finally, when all connected sections have become inactive, the node also deactivates.

In this algorithm for spatial adaptivity, there is a trade-off between computational speed and accuracy of the solution. A faster solution can be obtained by having the sections activate later and deactivate sooner, corresponding to higher tolerances, while a more accurate solution is obtained using smaller tolerances at the expense of increased computational time. In each simulation, the tolerances τ_A, τ_I and τ_V must be adjusted by the user until the desired balance between speed and accuracy is achieved.

As we have seen, the action potential is characterized by a large fast deviation from the resting voltage followed by a slower return to rest. Depending on the type of cell and the electrophysiological properties of the membrane, the return to rest can be of varying duration. In some cases, when the recovery is very long, the spatial adaptivity of Rempe and Chopp can be improved on. Following an action potential, large regions of the cell will be evolving very slowly, but will be too far from rest to be ignored, and therefore cannot be turned off. This particular situation motivated the development of a method for locally adaptive time stepping, leading to a further gain in efficiency in neural simulations. However, the method itself is not dependent on the unique behavior of the action potential, and is applicable to other classes of problems and adaptable to higher dimensions as well.

CHAPTER 2

Localized Adaptive Time Stepping

Most commonly, numerical solutions to (1.13a) are found using an implicit method, either Backward Euler or Crank-Nicolson for the linear diffusion term. The nonlinear reaction terms are then updated using an iterative procedure [**11, 39**] or with the trapezoidal rule and a staggered time step [**25**]. These methods are appealing since they are simple, stable and efficient, while maintaining the full accuracy of the underlying numerical method. However, the Backward Euler method is only first order, and is highly diffusive. The Crank-Nicolson method has dispersive error, leading to oscillations in the solution when a large time step is used. Under normal circumstances these oscillations decay in time, thus the method remains unconditionally stable. However, when combined with our domain decomposition scheme and locally adaptive time stepping these oscillations create instabilities at the branching points. We have therefore chosen to base our scheme on the second order Backwards Differentiation Formula (BDF) method, utilizing a predictor-corrector approach that is no more expensive than those proposed for either the Crank-Nicolson or Backward Euler schemes, while maintaining the superior stability properties of the BDF methods.

2.1. Second Order Backward Differentiation Formula

The backward differentiation formulas are a family of linear multistep methods for solving ordinary differential equations of the form $y'(t) = f(y, t)$. Due to their stability

properties, these methods are well suited to stiff systems, and have been incorporated into the CVODE [**10**] solver package. The pth order BDF scheme is derived from the Lagrange interpolating polynomial of order p to $y(t)$. In the case of the second order method, the solution $y(t)$ is approximated by the quadratic polynomial that passes through the points $\{(t^{q-1}, y^{q-1}), (t^q, yt^q), (t^{q+1}, y^{q+1})\}$ and has a slope at (t^{q+1}, yt^{q+1}) equal to $y'(t^{q+1})$. The second order BDF scheme is derived as follows. Using the Lagrange interpolating polynomial through the data points $\{(t^{q-1}, y^{q-1}), (t^q, yt^q), (t^{q+1}, y^{q+1})\}$, the interpolated value $y(t)$ is

$$y(t) = \sum_{j=n-1}^{n+1} y^j \ell_j, \tag{2.1}$$

where

$$\ell_j = \prod_{\substack{m=q-1 \\ m \neq j}}^{q+1} \left(\frac{t - t^m}{t^j - t^m} \right). \tag{2.2}$$

Substituting (2.2) into (2.1) and expanding, we obtain

$$\begin{aligned} y(t) &= y^{q-1} \left(\frac{t - t^q}{t^{q-1} - t^q} \right) \left(\frac{t - t^{q+1}}{t^{q-1} - t^{q+1}} \right) + y^q \left(\frac{t - t^{q-1}}{t^q - t^{q-1}} \right) \left(\frac{t - t^{q+1}}{t^q - t^{q+1}} \right) \\ &+ y^{q+1} \left(\frac{t - t^{q-1}}{t^{q+1} - t^{q-1}} \right) \left(\frac{t - t^q}{t^{q+1} - t^q} \right). \end{aligned}$$

We now differentiate with respect to t and evaluate at t^{q+1} to obtain:

$$y'(t^{q+1}) = \frac{y^{q-1}}{2\Delta t} - \frac{2y^q}{\Delta t} + \frac{3y^{q+1}}{2\Delta t} \tag{2.3}$$

Finally, substituting $y'(t) = f(y, t)$ and solving for y^{q+1} gives the update scheme:

$$y^{q+1} = \frac{4}{3}y^q - \frac{1}{3}y^{q-1} + \frac{2}{3}\Delta t f(y^{q+1}, t^{q+1}). \tag{2.4}$$

The adaptive time stepping variant of the BDF methods are again derived from the interpolating polynomial, using unequally spaced data points. This leads to a scheme where the coefficients are non-constant and depend on the ratio of step sizes. If we define $\Delta t^{q+1} = t^{q+1} - t^q$ and $\rho = \Delta t^{q+1}/\Delta t^q$, then the adaptive second order BDF scheme is:

$$(2\rho + 1)y^{q+1} = (\rho + 1)^2 y^q - \rho^2 y^{q-1} + (\rho + 1)\Delta t^{q+1} f(y^{q+1}, t^{q+1}). \tag{2.5}$$

From this point on, we will discuss only the adaptive scheme, as the corresponding fixed step scheme can be obtained by setting the ratio of step sizes (ρ) to 1. The Backward Differentiation Formulas are a family of methods developed for the solution of ordinary differential equations (ODEs), but can be easily applied to partial differential equations through the method of lines. In the next sections, we present our algorithm in the context of the generic one dimensional reaction-diffusion equation

$$\frac{\partial V}{\partial t} = \xi \frac{\partial^2 V}{\partial x^2} - R(V). \tag{2.6}$$

2.2. Predictor-Corrector Update in a One Dimensional Domain

Typically, when solving nonlinear partial differential equations, an iterative procedure (e.g. Newton Iteration) is employed to handle the nonlinearities. It can be shown that a single predictor-corrector update results in a method with the same order of accuracy as the base method. Iteration to completion is done to retain the stability properties of the

fully implicit scheme rather than increase the accuracy [**56**]. In the case of the Hodgkin-Huxley equations, the predictor-corrector scheme we use here retains sufficient stability for our purposes, as will be shown in section 2.7.2. For a one dimensional problem, the computational domain is discretized in space and divided into a number of subdomains that we refer to as sections, as shown in Figure 2.1, each of which is updated sequentially.

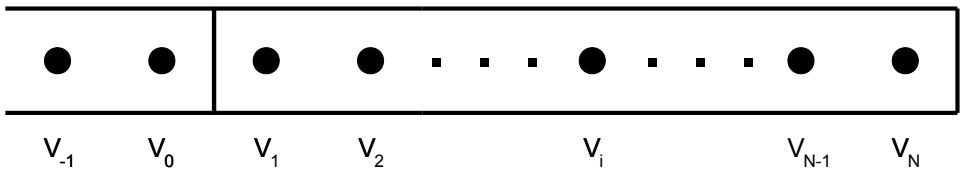


Figure 2.1. **One Dimensional Domain.** *Schematic representation of a one dimensional domain, showing the interface between neighboring sections, and the location of computational grid points relative to the interface.*

Spatial discretization using a central difference approximation leads to the coupled ODE system

$$\frac{dV_i}{dt} = \lambda V_{i-1} - 2\lambda V_i + \lambda V_{i+1} - R(V_i), \tag{2.7}$$

where $\lambda = \dfrac{\xi}{\Delta x^2}$.

We now describe the algorithm for updating a section of unbranched cable from time t^q to t^{q+1}. As the BDF methods are multi-step methods, a starting procedure is required. We have chosen to compute the first three update steps using the Backward Euler method, and focus our attention on later updates. Utilizing the adaptive second order BDF scheme presented in (2.5), and having solution history data at times t^q, t^{q-1} and t^{q-2} with $\Delta t^q = t^q - t^{q-1}$ we define $\rho = \Delta t^{q+1}/\Delta t^q$ and $\eta = \Delta t^q/\Delta t^{q-1}$. The first step is to predict the voltage at time t^{q+1} using the second order Lagrange interpolating polynomial of the

solution, to obtain

$$V_i^* = \left(\frac{\rho+1}{\eta+1}\right)(\rho\eta+\eta+1)V_i^q - \rho(\rho\eta+\eta+1)V_i^{q-1} + \rho\eta^2\left(\frac{\rho+1}{\eta+1}\right)V_i^{q-2}. \tag{2.8}$$

The second step is to obtain the value of the reaction term at t^{q+1}, using the predicted value V^*, giving $R^* = R(V^*, t^{q+1})$.

Finally, the voltage is corrected at time t^{q+1} using R^*. In the interior of a section, the update is given by:

$$-\lambda\tau V_{i-1}^{q+1} + \left[(2\rho+1) + 2\lambda\tau\right]V_i^{q+1} - \lambda\tau V_{i+1}^{q+1} = (\rho+1)^2 V_i^q - \rho^2 V_i^{q-1} - \tau R^*, \tag{2.9}$$

where we have defined $\tau = (\rho+1)\Delta t^{q+1}$.

At the ends of sections, boundary data for the update is obtained from the solution in the neighboring section. In the case of V_1 in Figure 2.1, we have the corrector update:

$$\left[(2\rho+1) + 2\lambda\tau\right]V_1^{q+1} - \lambda\tau V_2^{q+1} = (\rho+1)^2 V_1^q - \rho^2 V_1^{q-1} + \lambda\tau\tilde{V}_0 - \tau R^*, \tag{2.10}$$

where $\tilde{V}_0$ is either the computed value of V_0^{q+1}, when the branch containing V_0 has already been updated to time t^{q+1}, or else an approximation obtained through linear extrapolation in time is used.

There are a few details regarding the order in which sections are updated, but these will be dealt with in the next section, when we present the localized time stepping scheme.

2.3. Localized Adaptive Time Stepping

One of the main benefits of the domain decomposition scheme employed here is that the resultant method is well suited for locally adaptive time stepping. Since each section is updated independently of the others, regions experiencing high levels of activity can be updated using a small Δt while the remainder of the system takes larger time steps, thus concentrating computational power where it is most needed, as determined by local activity. As our primary goal is to reduce the computational cost while retaining a reasonable solution for comparison to empirical observations, we have chosen to eliminate the additional computations required to obtain an estimate of the local error. Instead we utilize an inexpensive measure of local activity in each section:

$$\epsilon = \left\| \mathbf{V}^{q+1} - \mathbf{V}^{q} \right\|_2, \tag{2.11}$$

where $\|\mathbf{V}\|_2$ is the L_2 norm of the vector of values $\mathbf{V}$ over the entire length of the section. The activity level is then compared to the specified tolerance, τ, and as in CVODE [**6, 7, 10**], the section chooses the step size for the next step to be

$$\Delta t^* = 0.8 \left(\frac{\tau}{\epsilon} \right)^{\frac{1}{3}} \Delta t^{q+1}. \tag{2.12}$$

In the case $\epsilon > \tau$, the update is discarded, and the section is updated again from t^q using a step of Δt^*. Following each successful update, in which $\epsilon < \tau$, the size of the next step is also taken to be Δt^* and the section is placed in a queue to ensure that the next update occurs at the correct time relative to the other sections of the cell. Our algorithm then proceeds in a time-marching manner. When the simulation reaches time

t_j^{q+1}, corresponding to the end of an update interval for section j, the update for section j from t_j^q to t_j^{q+1} is performed as outlined in section 2.2. We update sections at the end of the update step rather than the beginning to ensure that the most accurate boundary data possible is available. In cases where multiple branches have a common scheduled time of update, they are updated sequentially in order of decreasing levels of activity. That is, sections with larger $\frac{\partial V}{\partial t}$ are updated first.

2.4. Summary of Algorithm

We have presented the details for the update of a single section of the computational domain. The time evolution of the entire system is obtained by computing individual updates for each section, ordered in an update queue by time of update and level of local activity. The algorithm steps through the queue, performing the following steps.

(1) **Select section from update queue** with the earliest next update time.

(2) **Obtain predictor value** V^*, from the Lagrange interpolating polynomial using (2.8).

(3) **Update reactions** using the predicted solution V^* at time t^{q+1}.

(4) **Compute the corrector value** using the updated reaction term as in (2.9) for interior points. Grid points at the ends of the section are updated as in (2.10) using an approximation for the boundary data at time t^{q+1}, either a computed solution at the neighboring grid point, or an approximation obtained through extrapolation.

(5) **Determine new time step** as in (2.12) using the activity measure ϵ given in (2.11), and the user specified activity tolerance τ.

(6) **Schedule next update.** If the local activity in the completed step is below the tolerance, determine the section's place in the update queue and add it. In the case the activity is too large, immediately recompute the update using the newly determined (smaller) time step.

At the successful completion of the update for a single section, the above steps are repeated for the next section in the update queue, until all sections have been updated to the terminal time of the simulation.

2.5. Extension to Higher Dimensions

We have presented our algorithm for locally adaptive time stepping to reaction-diffusion problems in one dimension.. It should be straightforward, however, to adapt our method to other problems in higher dimensions.

Consider the two dimensional reaction-diffusion equation

$$\frac{du}{dt} = \nabla^2 u + R(u),$$

where $u = u(x, y, t)$ and $\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ is the Laplacian operator. In this case, the spatial derivative is computed using a 5 point stencil:

$$\nabla^2 u = \frac{1}{h^2}(u(x_{i-1}, y) + u(x_{i+1}, y) + y(x, y_{i-1}) + y(x, y_{i+1}) - 4u(x, y)),$$

where h is the uniform grid spacing in both the x and y directions. If the spatial domain is divided into rectangular subdomains, these cells can be treated similarly to the sections of unbranched cable presented here. Each cell is then updated using an independently selected time step appropriate for the level of local activity. The cells are updated at the

end of their update step, and ordered based on the level of local activity. Boundary data is easily obtained from the computed solution in neighboring cells or from extrapolation of previously computed solutions in cells that have not yet been updated.

2.6. Application of Generic Algorithm to Neural Simulations

In this section, we apply our time stepping algorithm to the highly branched structures found in neuroscience simulations, beginning with a discussion of the spatial discretization of the branched neural structure.

2.6.1. Spatial Discretization and Branching

We follow the discretization method used by many of the available neural simulation packages and approximate the cable equation (1.13a) as a series of compartments connected by resistors, as shown in Figure 2.2 for a section of unbranched cable. The size of

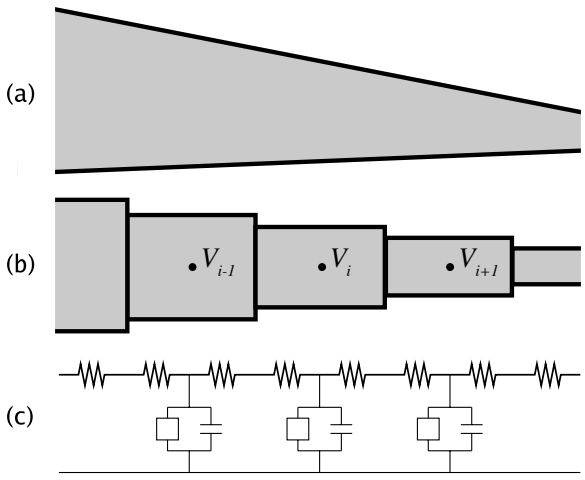


Figure 2.2. **Compartmentalization** *The top figure (a) represents a section of an unbranched dendrite. In our discretization, this is split into a number of compartments, assumed to be of constant radius as shown in the middle panel (b). The bottom panel (c) shows the equivalent circuit diagram for the system. Similar to Figure 3.3 in* [**26**].

each compartment is chosen to be small enough that the physical properties of the cell can reasonably be considered uniform within the compartment, with the computational grid point located in the center. The flux of voltage from one compartment to another is then given by the difference in voltage divided by the total resistance between the compartments. If we let

$$R_i = \frac{R_a \Delta x_i}{\pi a_i^2} \tag{2.13}$$

be the total resistance within each compartment shown in Figure 2.2(b), the total resistance between the centers of compartments i and k (for $k = i \pm 1$) is

$$R_{i,k} = \frac{1}{2}(R_i + R_k) = \frac{1}{2}\left(\frac{R_a \Delta x_i}{\pi a_i^2} + \frac{R_a \Delta x_k}{\pi a_k^2}\right) = \frac{R_a(a_i^2 \Delta x_k + a_k^2 \Delta x_i)}{2\pi a_i^2 a_k^2} \tag{2.14}$$

since resistances sum linearly.

Following this discretization, the Hodgkin-Huxley cable equation (1.13a) is converted to a system of ordinary differential equations of the form:

$$C_m \frac{dV_i}{dt} = \frac{1}{A}\left[G_{i,i-1}(V_{i-1} - V_i) + G_{i,i+1}(V_{i+1} - V_i)\right] - \Gamma V_i + \gamma, \tag{2.15a}$$

where

$$G_{i,k} = \frac{1}{R_{i,k}} = \frac{2\pi a_i^2 a_k^2}{R_a(a_i^2 \Delta x_k + a_k^2 \Delta x_i)} \tag{2.15b}$$

is the harmonic average of the conductances in compartments i and k, $A = 2\pi a_i \Delta x_i$ is the surface area of compartment i, and for notational simplicity we have defined:

$$\begin{aligned} \Gamma &= \overline{g_{\mathrm{Na}}} m^3 h + \overline{g_{\mathrm{K}}} n^4 + g_{\mathrm{leak}}, \\ \gamma &= \overline{g_{\mathrm{Na}}} m^3 h E_{\mathrm{Na}} + \overline{g_{\mathrm{K}}} n^4 E_{\mathrm{K}} + g_{\mathrm{leak}} E_{\mathrm{leak}} + \frac{I_e}{2\pi a} \delta(x - x_0). \end{aligned}$$

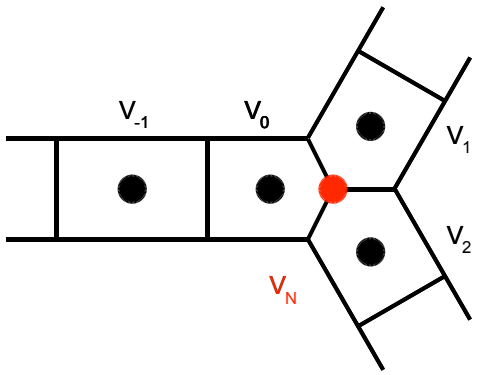


Figure 2.3. **Three-branch structure** *An example of a branching pattern in the neural tree structure, showing the computational points and corresponding compartments near the branching nodal point.*

At the nodal points where branches connect, an additional condition is imposed to ensure conservation of flux:

$$\sum_{b=0}^{n} \frac{2\pi a_b^2}{R_a \Delta x_b} (V_b - V_N) = 0 \Rightarrow V_N = \frac{\sum_{b=0}^{n} \frac{a_b^2}{\Delta x_b} V_b}{\sum_{b=0}^{n} \frac{a_b^2}{\Delta x_b}} \tag{2.16}$$

where V_N is the voltage at the node, and the sum is over all branches connected at the node. An example of a branch point with three connected branches is given in Figure 2.3. In this case, the voltage V_0 evolves according to the equation:

$$C_m \frac{dV_0}{dt} = \frac{1}{A} G_{0,-1}(V_{-1} - V_0) + \frac{1}{A} G_0 (V_N - V_0) - \Gamma V_0 + \gamma, \tag{2.17}$$

where

$$G_0 = \frac{2\pi a_0^2}{R_a \Delta x_0}$$

is the conductance from the center of compartment 0 to the branching nodal point. Note that the formulation in (2.17) is valid for an arbitrary number of connected branches, with V_N defined as in (2.16), and that in the case of only 2 branches connected at the node, $G_0(V_N - V_0) = G_{0,1}(V_1 - V_0)$. At unconnected ends of branches, V_N is equal to the voltage in the end compartment of the branch, corresponding to a no-flux boundary condition.

As mentioned previously, most of the available numerical packages for neural simulations treat the entire cell network as a single large system. As our goal is localized time step adaptivity, we reduce the large system into smaller systems corresponding to sections of unbranched cable, coupled through boundary conditions at the branching nodal points V_N as described above. Each section of cable is then updated sequentially using the predictor-corrector approach described in Section 2.6.2.

2.6.2. Predictor-Corrector Update

We now present the details of the update scheme presented in Section 2.2, for the full Hodgkin-Huxley cable equation. As in Section 2.2, we define $\Delta t^q = t^q - t^{q-1}$, $\rho = \Delta t^{q+1}/\Delta t^q$ and $\eta = \Delta t^q/\Delta t^{q-1}$ and assume that the solution is known at the previous times t^q, t^{q-1}, and t^{q-2}. The first step is to predict the voltage at time t^{q+1} using the second order Lagrange interpolating polynomial of the solution, to obtain

$$V_i^* = \left(\frac{\rho+1}{\eta+1}\right)(\rho\eta+\eta+1)V_i^q - \rho(\rho\eta+\eta+1)V_i^{q-1} + \rho\eta^2\left(\frac{\rho+1}{\eta+1}\right)V_i^{q-2}. \tag{2.18}$$

The second step is to solve for the rate constants $\alpha(V^*)$ and $\beta(V^*)$ as given in (1.13d), and update the gating variables (1.13c) using the second order BDF method. For example, the update for the variable m is:

$$(2\rho+1)m^{q+1} = (\rho+1)^2 m^q - \rho^2 m^{q-1} + (\rho+1)\Delta t^{q+1}\left(\alpha_m(V^*)(1-m^{q+1}) - \beta_m(V^*)m^{q+1}\right).$$

Solving for m^{q+1} yields:

$$m^{q+1} = \frac{(\rho+1)^2 m^q - \rho^2 m^{q-1} + (\rho+1)\Delta t^{q+1}\alpha_m(V^*)}{(2\rho+1) - (\rho+1)\Delta t^{q+1}(\alpha_m(V^*)+\beta_m(V^*))}. \tag{2.19}$$

The variables h and n are updated in a similar manner, and the resulting values are used to determine Γ^{q+1} and γ^{q+1}.

Finally, the voltage is corrected at time t^{q+1} using the computed values of the gating variables at the new time point. In the interior of a section, the update is given by:

$$\begin{aligned} &-(\rho+1)\frac{\Delta t^{q+1}}{C_m}\frac{1}{A}G_{i,i-1}V_{i-1}^{q+1} \\ &\quad + \left[(2\rho+1) + (\rho+1)\frac{\Delta t^{q+1}}{C_m}\left(\frac{1}{A}(G_{i,i-1}+G_{i,i+1}) + \Gamma^{q+1}\right)\right]V_i^{q+1} \\ &\quad - (\rho+1)\frac{\Delta t^{q+1}}{C_m}\frac{1}{A}G_{i,i+1}V_{i+1}^{q+1} \\ &\quad = (\rho+1)^2 V_i^q - \rho^2 V_i^{q-1} + (\rho+1)\frac{\Delta t^{q+1}}{C_m}\gamma^{q+1}. \end{aligned} \tag{2.20}$$

At the ends of sections, the boundary data V_N is determined using (2.16) to obtain

$$V_N = \frac{\sum_{b=0}^{n} \frac{a_b^2}{\Delta x_b} \tilde{V}_b}{\sum_{b=0}^{n} \frac{a_b^2}{\Delta x_b}}, \tag{2.21}$$

where $\tilde{V}_b$ is an approximation to the voltage in the neighboring compartments at time t^{q+1}. As the sections are updated sequentially, this approximation will either be the computed value for sections that have already been updated or a linear extrapolation from the previously computed solutions. If we consider the update for V_0 in the three branch example shown in Figure 2.3, we obtain:

$$V_N = \frac{\frac{a_0^2}{\Delta x_0} V_0^{q+1}}{\sum_{b=0}^{2} \frac{a_b^2}{\Delta x_b}} + \frac{\sum_{b=1}^{2} \frac{a_b^2}{\Delta x_b} \tilde{V}_b}{\sum_{b=0}^{2} \frac{a_b^2}{\Delta x_b}}, \tag{2.22}$$

and the voltage update equation becomes:

$$\begin{aligned} &- (\rho+1)\frac{\Delta t^{q+1}}{C_m}\frac{1}{A} G_{0,-1} V_{-1}^{q+1} \\ &+ \left[(2\rho+1) + (\rho+1)\frac{\Delta t^{q+1}}{C_m} \left(\frac{1}{A} \left[G_{0,-1} + G_0 \left(1 - \frac{\frac{a_0^2}{\Delta x_0}}{\sum_{b=0}^{2} \frac{a_b^2}{\Delta x_b}} \right) \right] + \Gamma^{q+1} \right) \right] V_0^{q+1} \\ &= (\rho+1)^2 V_0^q - \rho^2 V_0^{q-1} + (\rho+1)\frac{\Delta t^{q+1}}{C_m} \left(\frac{G_0}{A} \frac{\sum_{b=1}^{2} \frac{a_b^2}{\Delta x_b} \tilde{V}_b}{\sum_{b=0}^{2} \frac{a_b^2}{\Delta x_b}} + \gamma^{q+1} \right). \end{aligned} \tag{2.23}$$

In the case where V_0 is the voltage at the end of an unconnected segment, we obtain $V_N = V_0$ and the update:

$$
\begin{aligned}
(2.24) \quad & -(\rho+1)\frac{\Delta t^{q+1}}{C_m}\frac{1}{A}G_{0,-1}V_{-1}^{q+1} \\
& + \left[(2\rho+1) + (\rho+1)\frac{\Delta t^{q+1}}{C_m}\left(\frac{G_{0,-1}}{A} + \Gamma^{q+1}\right)\right] V_0^{q+1} \\
& = (\rho+1)^2 V_0^q - \rho^2 V_0^{q-1} + (\rho+1)\frac{\Delta t^{q+1}}{C_m}\gamma^{q+1}.
\end{aligned}
$$

As in the generic case, sections are updated at the end of their update step, and updates are scheduled by update time and ordered by activity level, as described in Section 2.3.

2.6.3. Localized Adaptive Stepping

For neural simulations, locally adaptive time stepping is implemented as described in Section 2.3, with a few adjustments. To measure the local activity in section j, changes in the gating variables are considered in addition to the voltage, to provide the estimate of activity:

$$
(2.25) \qquad \epsilon_j = \max\left\{\frac{\left\|\mathbf{V}^{q+1} - \mathbf{V}^q\right\|_2}{V_{\max} - V_{\min}}, \left\|\mathbf{m}^{q+1} - \mathbf{m}^q\right\|_2, \left\|\mathbf{h}^{q+1} - \mathbf{h}^q\right\|_2, \left\|\mathbf{n}^{q+1} - \mathbf{n}^q\right\|_2\right\},
$$

where the voltage has been scaled to be between 0 and 1, to match the range of values found in the gating variables. The step size for the next step is chosen as before to be

$$
(2.26) \qquad \Delta t^* = 0.8\left(\frac{\tau}{\epsilon}\right)^{\frac{1}{3}}\Delta t^{q+1}.
$$

In neural simulations, large portions of the cell are at or near rest for long periods of time. During these rest periods it is possible for the step size chosen by the branch to exceed the duration of the action potentials that propagate through the cell. Due to the short duration and steep initial rise of the action potential combined with large time steps in the resting sections it is possible for sections to remain at rest and miss the action potential that should have propagated through. To compensate for this, we have implemented a means for long periods between successive updates to be interrupted and the sections forced to update with a sufficiently small time step that accurately resolves the action potential. This forced update is based on the spatial adaptivity activation of Rempe and Chopp [**54**] and occurs when the following conditions are met. Section j is forced to update at time T^{q+1} if all three conditions are met:

(1) $\frac{\partial V}{\partial t} > \tau_{\mathrm{F}}$ in one of the sections, k, connected to section j for some user defined threshold τ_{F}.
(2) $\Delta t_j^{q+1} > \Delta t_{\mathrm{F}}$ for a user defined Δt_{F}. We typically choose $\Delta t_{\mathrm{F}} = 0.01$ms
(3) the next scheduled update for section j, $T_j^{q+1} > T_k^{q+1}$.

If all the above conditions are met, section j is updated from t_j^q to $t_j^{q+1} = T^{q+1}$ as in a regular update, with the additional restriction that $\Delta t^* \leq \Delta t_k^{q+1}$. This limitation on Δt^* is maintained for 1ms to ensure that the action potential reaches section j before it is free to choose its own Δt based on local error. We have also found it beneficial to include an additional restriction on Δt^* following a successful step. By considering activity levels in the neighboring branches when selecting the next step size, the section is less prone to drastic fluctuations in dt. Thus, the local activity computed in (2.25) is modified such

that:

$$\tilde{\epsilon}_j = \max_k \left\{ \epsilon_j, \frac{\epsilon_k}{\Delta t_k^q} \right\}, \tag{2.27}$$

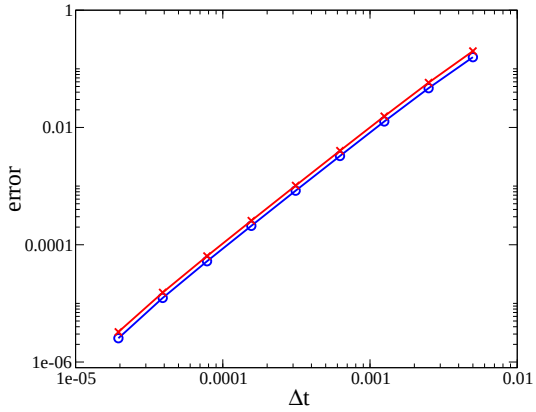
over all connected sections k.

2.7. Analysis of the Method

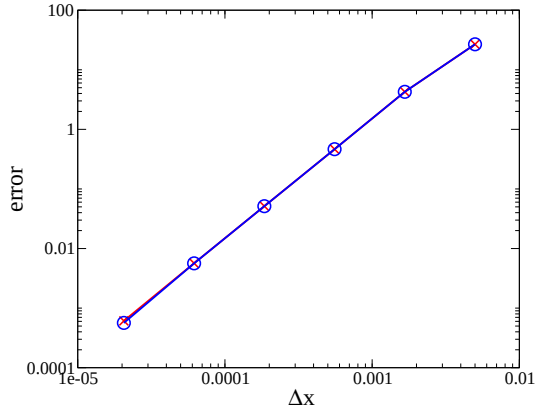
In this section, we verify the convergence, stability and accuracy of our predictor-corrector and domain decomposition schemes.

2.7.1. Convergence

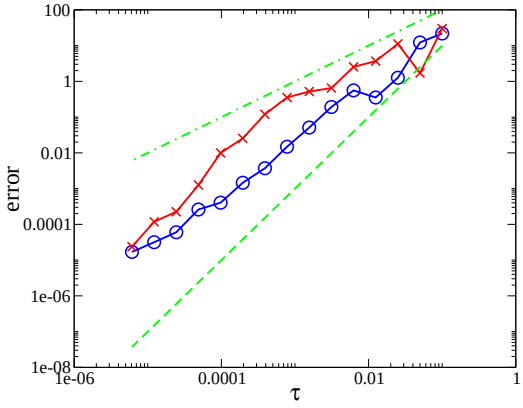
To test the convergence of our method we looked at the propagation of an action potential along a single unbranched cable. To test the predictor-corrector scheme introduced above, we used a cable consisting of a single section. Another test, in which the cable is made up of five sections allows us to test the effect of our domain decomposition scheme. In all tests, the cable was stimulated at one end to initiate an action potential that was then allowed to propagate along the cable. Since the Hodgkin-Huxley cable equation does not have an exact solution, a sequence of simulations with successively reduced discretizations (in either time or space) were performed. The solution obtained with the most refined grid and time step size was then taken to be the exact result and errors in the less refined simulations were determined by comparison to the finest grid solution using an L_2 norm. Figure 2.4(a) shows the temporal convergence of our method for a fixed value of $\Delta x = 0.001$cm. In both the single and five section tests, we see that a decrease in Δt by a factor of 2 results in a four-fold reduction in the error, indicating that our method



(a) Temporal Convergence



(b) Spatial Convergence



(c) Adaptive Temporal Convergence

Figure 2.4. **Convergence in Time and Space** *An action potential was propagated down a straight cable consisting of either a single section (open symbol) or five sections* (x)*. In the non-adaptive cases ((a) & (b)), an* $O(10^{-1})$ *reduction in grid size reduces the error by* $O(10^{-2})$ *indicating second order convergence in both time and space. In the adaptive case (c), the activity tolerance* τ *was reduced, keeping the spatial grid uniform. As the activity tolerance is reduced, we obtain a convergence rate between first and second order, as represented by the dashed lines.*

is indeed second order in time. Similarly, Figure 2.4(b) confirms that we have retained second order convergence in space for a fixed value of $\Delta t = 0.01\left(\frac{1}{2}\right)^{10} \approx 9.8 \times 10^{-6}$ms. To test the convergence of the adaptive method, rather than decreasing the size of the temporal grid, we decreased the tolerance for local activity that is used to determine the

appropriate step size. Errors in the solution were computed by comparison to the solution obtained with the smallest tolerance, using a spline interpolation of the time course to obtain solution values at the non-uniform time points. Figure 2.4(c) shows that we have a nearly second order relationship between the activity tolerance, τ, and the error in the resulting solution, for a fixed spatial grid with $\Delta x = 1.25 \times 10^{-4}$cm.

2.7.2. Stability

The BDF methods are known to have excellent stability properties, however, it is not obvious that our domain decomposition scheme combined with a single predictor-corrector iteration retains the required level of stability. In our local adaptive time stepping scheme, we require the method to be stable for large step sizes, as step sizes as large as 100 ms are possible in simulations run for long times with little activity. The local adaptive time stepping also results in large ratios of step sizes between neighboring sections. Figure 2.5 shows a typical simulation of an action potential propagating along an unbranched cable, and the largest ratio in step sizes is 6000, between sections 16 and 17.

To verify the stability of our scheme, we performed two tests on a section of unbranched cable containing only passive membrane properties (i.e., only the leak current). The cable was given initial conditions far from the resting voltage and was allowed to evolve toward equilibrium. The leak conductance was chosen so that the return to rest was very slow, so that even after 400 ms of evolution the cable had not reached equilibrium. To determine the stability of our method, this evolution was simulated with increasingly larger time steps Δt, up to Δt =100 ms. These large time steps are much larger than would be chosen in practice, but are used for the sole purpose of demonstrating the stability of the

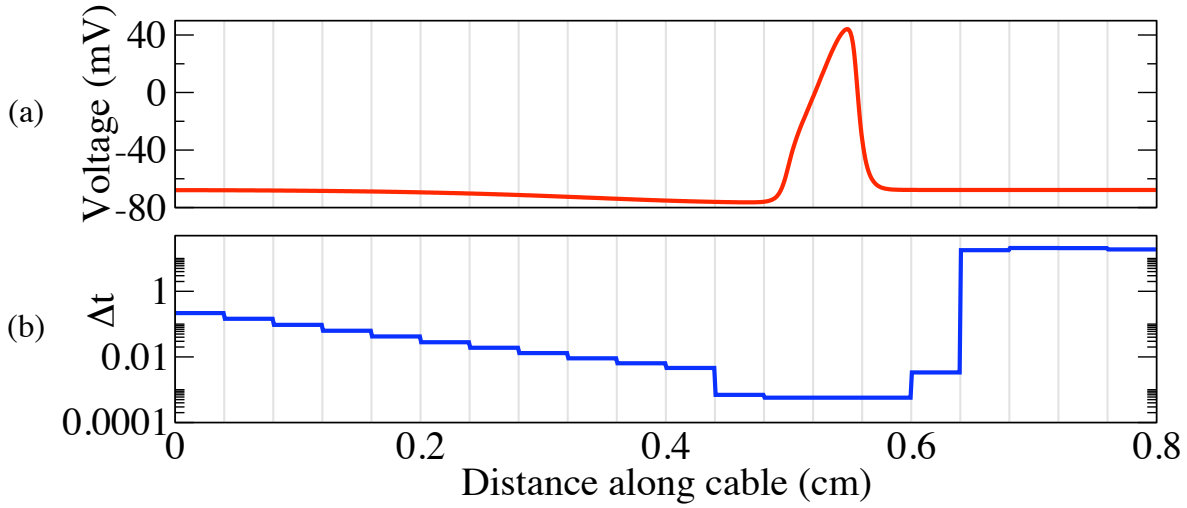


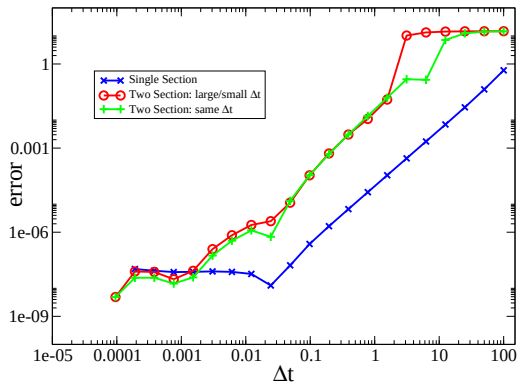
Figure 2.5. **Localized Adaptive Time Stepping on a straight cable.** *An action potential was initiated at the left end of a cable, and was allowed to propagate to the right. The cable was divided into 20 sections and the solution in space at time $T = 25.5ms$ is shown in (a), with the step size taken in each section shown in (b).*

algorithm. We also considered three related problems: we considered the cable as a single section, to verify that a single predictor-corrector iteration provided adequate stability; in the second test, we split the cable into two sections that were evolved with the same size Δt; finally we evolved the two sections with different size Δt. In locally adaptive time stepping, each section evolves with an independently chosen time step and consequently neighboring sections may have different step sizes. This last test determines how the ratio of step sizes between neighboring sections effects stability of the overall scheme.

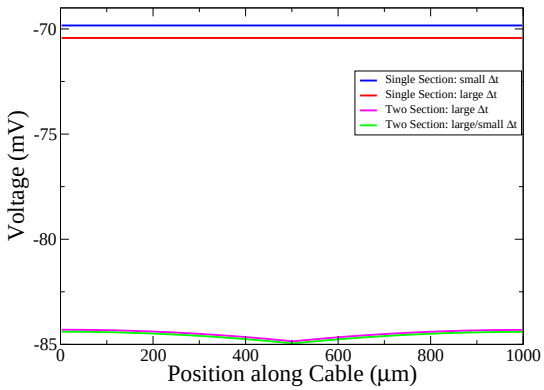
Figure 2.6(a) shows the relationship between increasing Δt and error in the final solution. The errors in all cases grow in a consistent manner, suggesting that the growth in error is due to accuracy concerns rather than stability. This suspicion is confirmed in Figure 2.6(b), showing the membrane voltage along the cable at the final time of T=400 ms, comparing a solution computed on the cable comprised of a single section with

$\Delta t \approx 1 \times 10^{-4}$ms to solutions obtained on a single section with Δt =100 ms, two sections using Δt =100 ms, and two sections with one section using the small $\Delta t \approx 1 \times 10^{-4}$ms and the other section evolving with Δt =100 ms. In the solutions involving two sections, we see that the solution is still stable, but highly inaccurate, with the greatest errors occurring at the midpoint of the cable where the two sections are connected, as expected. These errors at the junction nodes connecting sections result from a time-lag in the solution due to the sequential updates of the sections and the excessively large time step size. If the simulation is started with an initial condition above the resting voltage, the solution is most elevated at the point where the branches connect, as shown in Figure 2.6(c).

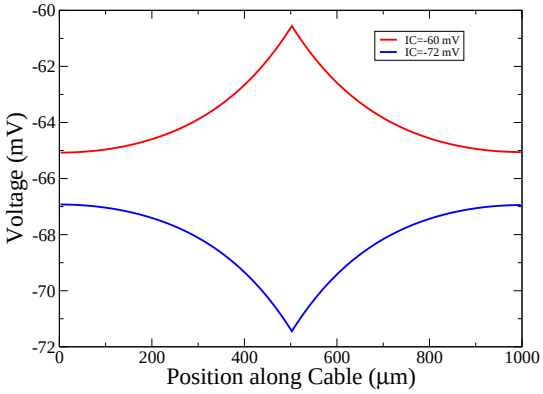
As a second test of stability, the same cable system, including active membrane properties, was given a noisy initial condition near the resting voltage, as shown in Figure 2.6(d). We then evolved this system using the same four protocols described above for Figure 2.6(b). Again we see that our method is stable for large time steps and large ratios of time steps in neighboring sections. The largest errors again occur in the two section cases, due to errors at the interface, and at such large time steps, the solution in time displays oscillations. However these oscillations decay as the system evolves, indicating that the overall method is stable. Furthermore, the action potential that is of most interest in neural systems has a height of roughly 120 mV, making the errors of $O(10^{-2})$ negligible in comparison. Additionally, in practice, our algorithm would only employ such large step sizes in an adaptive case when the solution is near equilibrium and evolving slowly. As evidenced in Figure 2.4(c), the user selected tolerance can be tuned to make the solution as accurate as desired. Included in Figure 2.6(d) is a solution obtained using the locally adaptive stepping scheme and a tolerance $\tau = 0.01$.



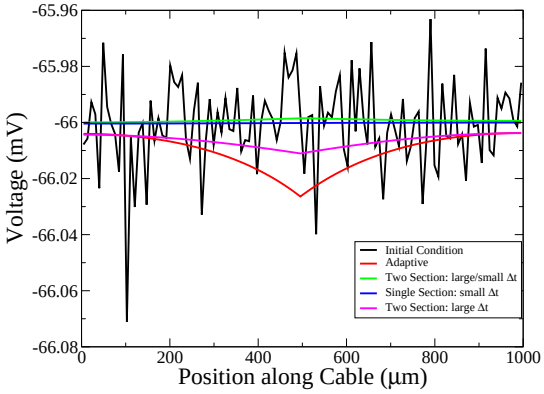
(a) **Error vs step size.** *In the two branch different Δt case, one branch was evolved with Δt given, while the second branch used a Δt of approximately 1×10^{-4}. In all other cases, the entire cable was evolved with the step size given on the horizontal axis.*



(b) **Final state at T = 400**. *This plot shows the voltage along the cable for each of the cases studied with $\Delta t = 100$. We see that all the solutions are stable, but the cases with multiple branches are not as accurate for large time steps.*



(c) **Largest errors at nodes where sections connect.** *For initial conditions above and below the resting voltage of -66 mV, the largest errors occur where the sections connect, due to a time lag in the update.*



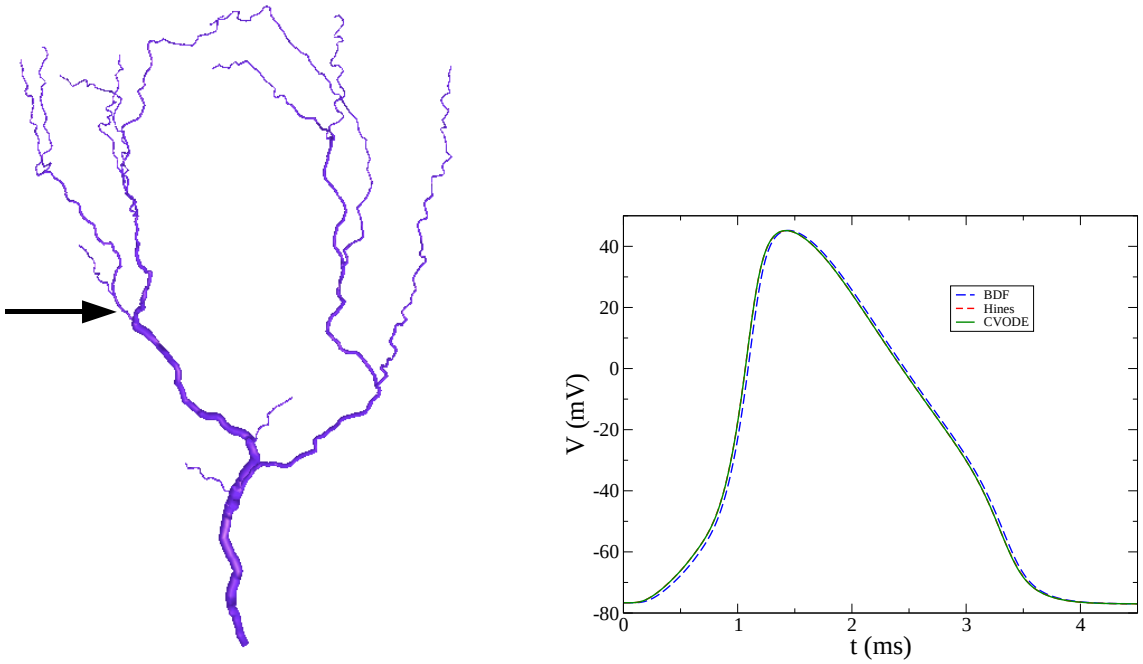
(d) **Final state at T = 400ms, from noisy initial conditions** *This plot shows the noisy initial condition, and the solutions obtained at $T = 400ms$*

Figure 2.6. **Stability results on an unbranched cable.** *A cable with only passive (leak) properties was simulated during a slow evolution towards rest. The plot in 2.6(a) shows the error in the voltage along the cable compared to the evolution of a cable made up of a single section, and evolved with $\Delta t \approx 1 \times 10^{-4}$. The plot in 2.6(b) shows the final voltages for the simulations using the largest Δt and 2.6(c) show evolution from initial conditions above and below rest for the two section case with Δt =100. 2.6(d) shows the evolution of noisy initial conditions with the largest Δt.*

2.7.3. Accuracy

To test the accuracy of our predictor-corrector scheme, we compared it to the main methods employed by NEURON: Hines' variant of the Crank-Nicolson method and CVODE, an adaptive time stepping scheme. In both cases, NEURON utilizes a single large matrix for the entire tree structure. The branched structure used was the dendritic tree shown in Figure 2.7(a). This is the distal portion of the dendritic arbor of a CA1 pyramidal cell from rat hippocampus generated in Professor Nelson Spruston's lab. This reconstruction was previously used in a mathematical modeling study that was compared to experimental results [**21**]. For the current study, Hodgkin-Huxley type channels were distributed throughout the structure, making the entire tree excitable. A stimulating current of 1.2 nA was applied to the base of the tree for 0.5 ms, leading to the formation of an action potential that then propagated through the tree. Figure 2.7(b) shows the time course for the membrane voltage recorded mid-tree at the location marked with the arrow in Figure 2.7(a). Three separate time courses are plotted in Figure 2.7(b), one for our predictor-corrector scheme combined with the domain decomposition described in section 2.6.1, along with traces of the solution obtained using Hines' implementation of the Crank-Nicolson method and CVODE. As visible in the plot, all three methods are in good agreement, indicating that the domain decomposition has not lead to a loss of accuracy.

To verify the accuracy of our spatial discretization, we considered action potential propagation through an entire cell, and compared the results using our method to the result obtained using the NEURON simulation environment. Using the full cell from the previous test, current was injected into the soma to initiate an action potential that



(a) (b)

Figure 2.7. **Accuracy comparison of predictor-corrector method** *The dendritic tree shown in (a) was stimulated at the base of its trunk with a 1.2 nA stimulus lasting for 0.5 ms. The plot in (b) shows time courses of the voltage recorded at the location marked with the arrow in (a). Our implementation of the second order BDF method with the domain decomposition scheme described in section 2.6.1 (long dashed) is compared to Hines' implementation of the Crank-Nicolson method (short dashed) and CVODE (solid trace). There is good agreement between all three methods. Spatial discretization was done according to the* d_λ *rule described by Carnevale and Hines* [**8**]. *CVODE was run using relative tolerance of* 1×10^{-4}, *and absolute tolerance of* 1×10^{-2}, *Hines method and our predictor-corrector scheme used* Δt *= 0.01 ms.*

propagated through the tree structure. The voltage was recorded in the soma and a distal branch, as shown in Figure 2.8. Again we have good agreement in the solutions obtained, demonstrating that our spatial discretization is consistent with that used by NEURON.

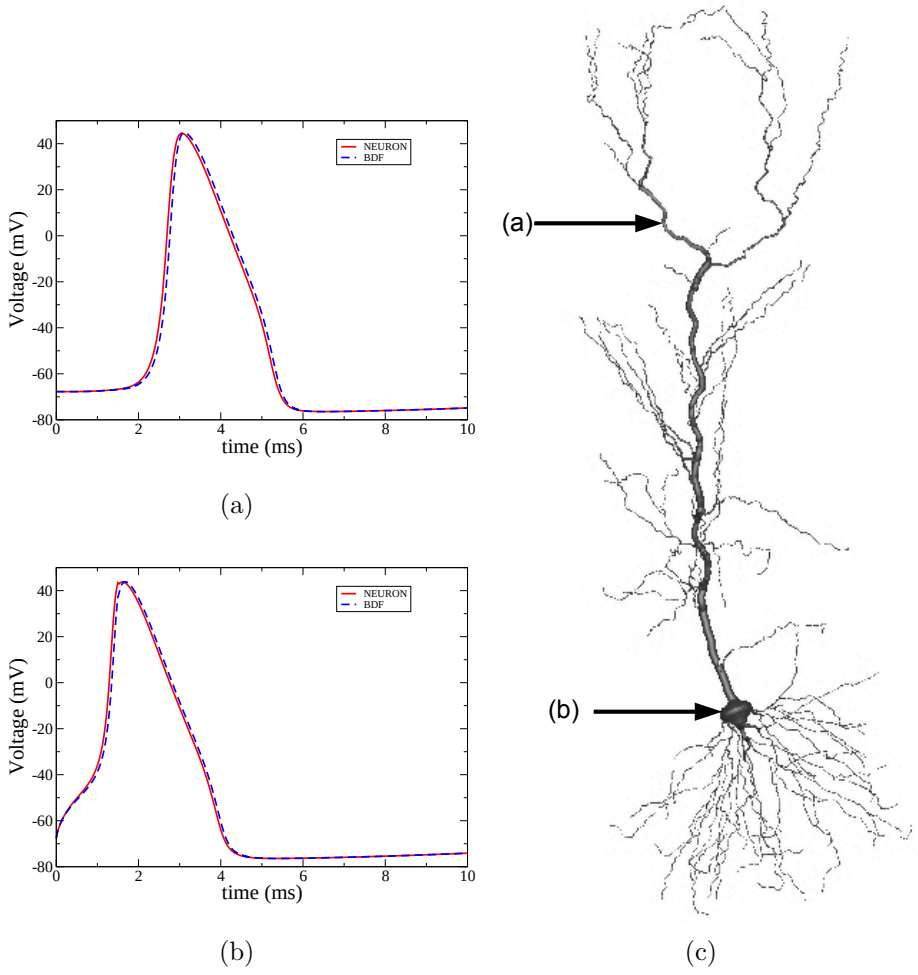


Figure 2.8. **Accuracy Comparison to NEURON.** *The cell in (c) was stimulated at the cell body at the point marked with the arrow (b) to initiate an action potential that propagated through the cell. The time courses presented in (a) and (b) show the voltage evolution at the points in the tree marked with arrows in (c).*

CHAPTER 3

Application of the Method

In this section, we compare our locally adaptive time stepping scheme to other methods currently used to solve problems in computational neuroscience.

3.1. Localized Adaptive Time Stepping on an Unbranched Cable

Figure 3.1 demonstrates the locally adaptive time stepping in practice. A straight cable 8000 μm in length was divided into 20 sections and stimulated at the left end. The resulting action potential was allowed to propagate through the cable. The top two panels show a snapshot in time of the simulation. The top panel shows the membrane potential at each point along the cable at simulation time $T = 25.5$ ms. The middle panel shows the corresponding time step size used by each section at that point in time. The bottom panel displays the update history for the entire simulation: the vertical axis is the simulation time and for each section numbered from 1 to 20, there is a horizontal bar at each time the section was updated. Because each section uses an independently chosen step size, it is possible for the sections at resting voltage or in the recovery phase to use a significantly larger time step than the sections containing the steep part of the action potential.

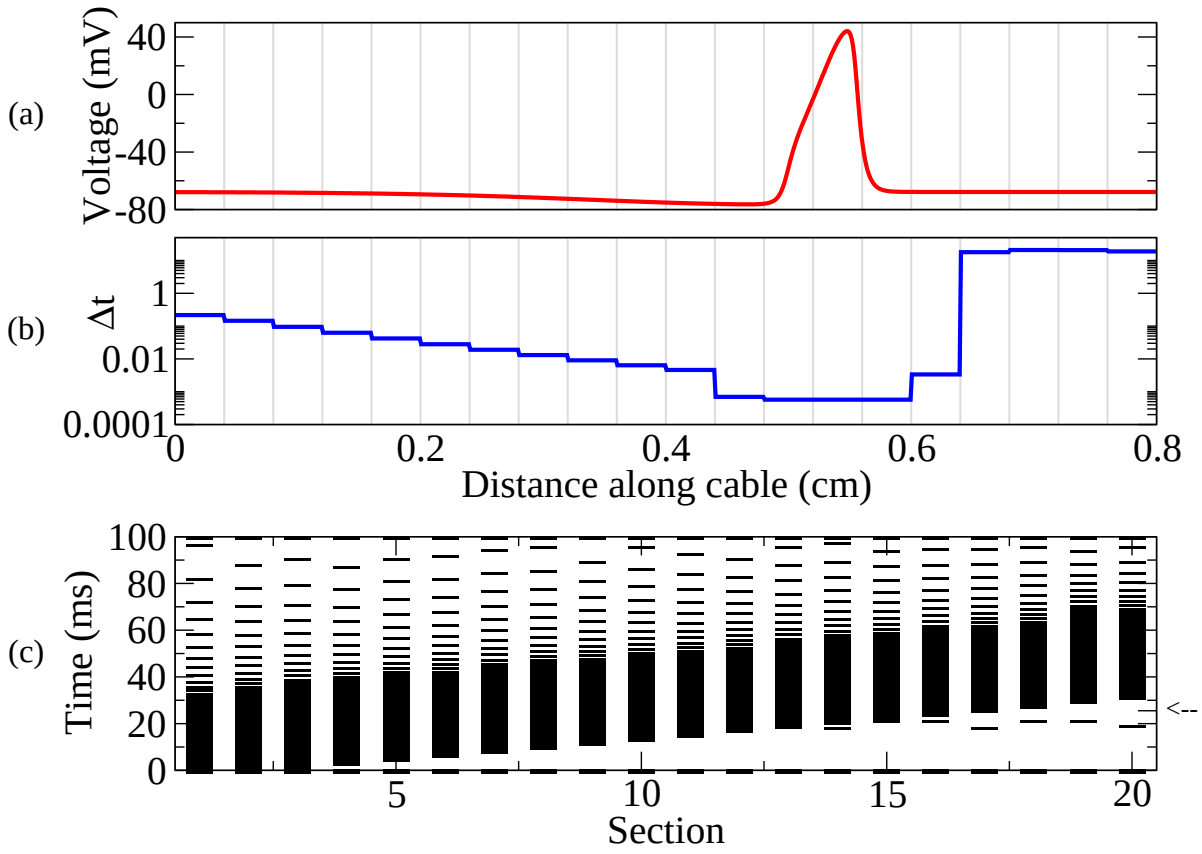


Figure 3.1. **Localized time adaptivity on an unbranched cable.** *The cable has been divided into 20 segments, each of which update independently. In the bottom panel, each horizontal bar shows the time at which the corresponding segment was updated during the simulation. The top two panels show a snapshot in time, at T=25.5 ms. The solution in each segment is given in the top panel, and the step size used is given in the middle panel.*

3.2. Comparison of Numerical Methods: Action Potential Propagation along Unbranched Cables

The numerical method we have presented here allows for locally adaptive time stepping, but at the expense of introducing extra steps into the update algorithm. To determine the additional computational cost of our domain decomposition method and the

relative savings provided by locally adaptive time stepping we compared our method to existing schemes for neural simulations. We compared five methods: The method used by Hines [**25**], CVODE [**10**], our predictor-corrector scheme without adaptivity, the locally adaptive time stepping scheme presented here, and a variant of the spatial adaptivity of Rempe and Chopp [**54**] modified to work with our spatial discretization. For each method, we simulated the propagation of an action potential along a series of unbranched cables with increasing lengths. Each cable consisted of individual sections 100μm in length connected as described in Section 2.6.1. Figure 3.2 shows the computational time for each method plotted against the total length of the cable. In each case, our predictor-corrector method without adaptivity is more costly than the corresponding simulation done with Hines' method. This is expected due to the additional calculations that are required at the points where sections connect. Both of these methods scale linearly with the cable length. CVODE is known to be much slower than Hines' method, and the simulation time grows at a much greater rate, as shown in Figure 3.2(b). In contrast, the computational time for the adaptive methods is not significantly affected by the problem size, since both focus computational resources on the few sections that contain the action potential. CVODE is very expensive, especially for large problems because of the nonlinear, iterative solve that must be done at each update as well as the need to decrease time step sizes as the problem size increases to maintain constant global error. Figure 3.2(c) shows the final spatial solution of our test problem computed with each of the methods tested, verifying that all methods provide solutions with comparable accuracy.

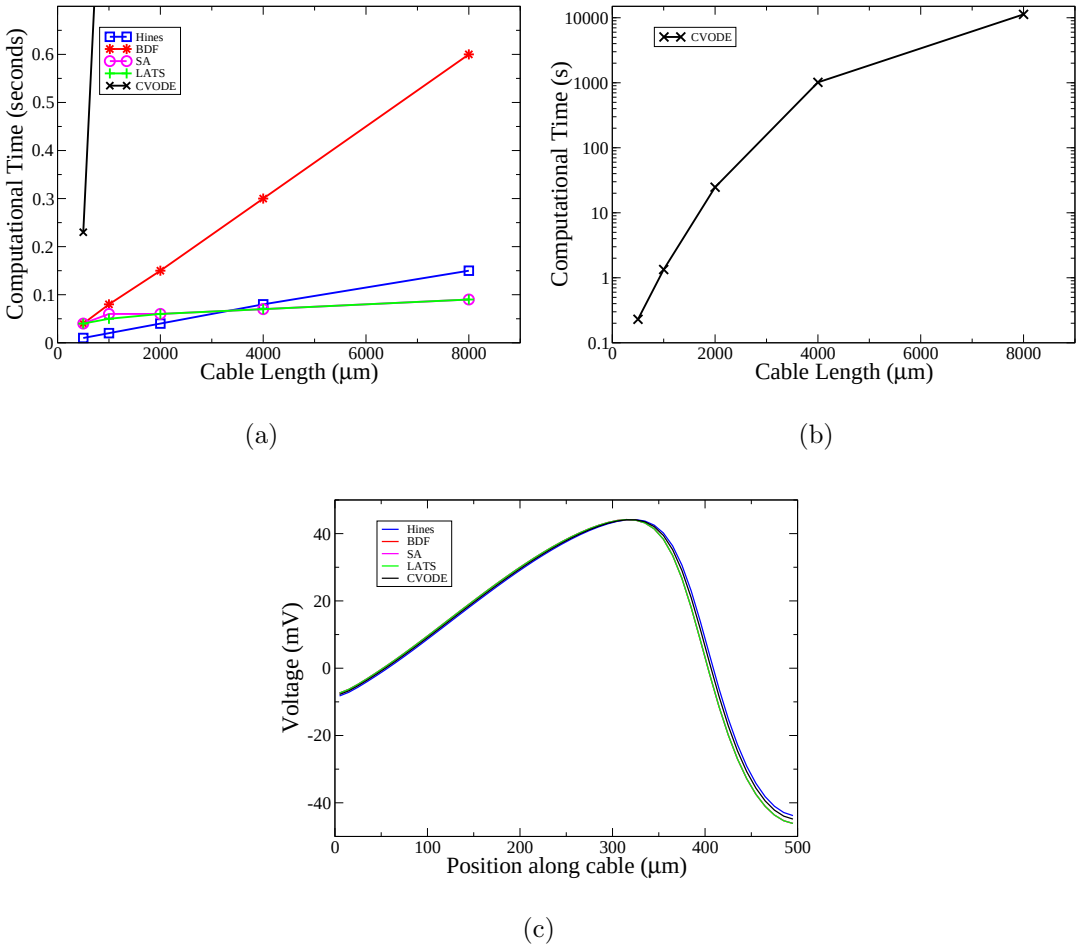


Figure 3.2. **Timing comparison of numerical methods on various length cables.** *Shown in 3.2(a) and 3.2(b) is the computing time in seconds to simulate the propagation of an action potential along cables of various lengths. Each cable is made up of several sections that are each 100 µm long and connected as described in Section 2.6.1. Here we compare the simulation times of two classes of schemes. In the case of Hines' method (H), CVODE (C), and our predictor-corrector scheme (NA), the entire cable is updated using a globally chosen time step, which is adaptive for CVODE. The second class of methods exploit the local activity levels in the cell: our locally adaptive time stepping scheme (LATS), and an adaptation for our spatial discretization of the spatial adaptivity of Rempe and Chopp* [**54**] *(SA). In the non-adaptive methods the computational time grows linearly with the length of the cable simulated, whereas it remains nearly independent of the cable length in the spatially adaptive and locally adaptive time stepping methods. The adaptive methods concentrate computational resources in the regions of highest activity and therefore the computational cost scales with the amount of activity in the system rather than the physical size. Shown in 3.2(c) is the spatial solution computed with all methods, showing comparable accuracy achieved with each method.*

The cable simulated in this test is artificial, in that realistic neurons do not have such long individual processes. However, the total number of sections and grid points is comparable to that in typical cells. From Figure 3.2(a), we see that the benefits of adaptivity (spatial and locally adaptive stepping) appear as the physical size of the problem grows. For simulations of a single cell, there is no reason to include the added computational overhead of our method, as Hines' method provides a more efficient solution. There is however a threshold problem size, after which the efficiency gains of adaptivity outweigh any extra overhead. As seen in Figure 3.2(a), the computational cost of the adaptive methods remains constant as the size of the problem grows, while the cost of Hines' method and CVODE continues to grow. Additionally, even though our locally adaptive time stepping method does not provide the exact control of errors that CVODE does, a careful choice of the tolerance τ will produce a solution of comparable accuracy to one obtained by CVODE at a fraction of the cost. Indeed, true error control could be easily added to the algorithm but as our main goal was increased speed of simulation we have chosen to avoid the extra computations required.

3.3. Optimal Section Size

The locally adaptive time stepping scheme is effective due to spatial localization of activity within the physical domain, suggesting that the efficiency of the method will be optimal when the size of the sections corresponds to some physical length scale present in the solution. To explore this relationship, we simulated action potential propagation along a 51200μm long cable. A series of computations were conducted with the cable divided into an increasing number of sections. The simulation was run long enough for the action

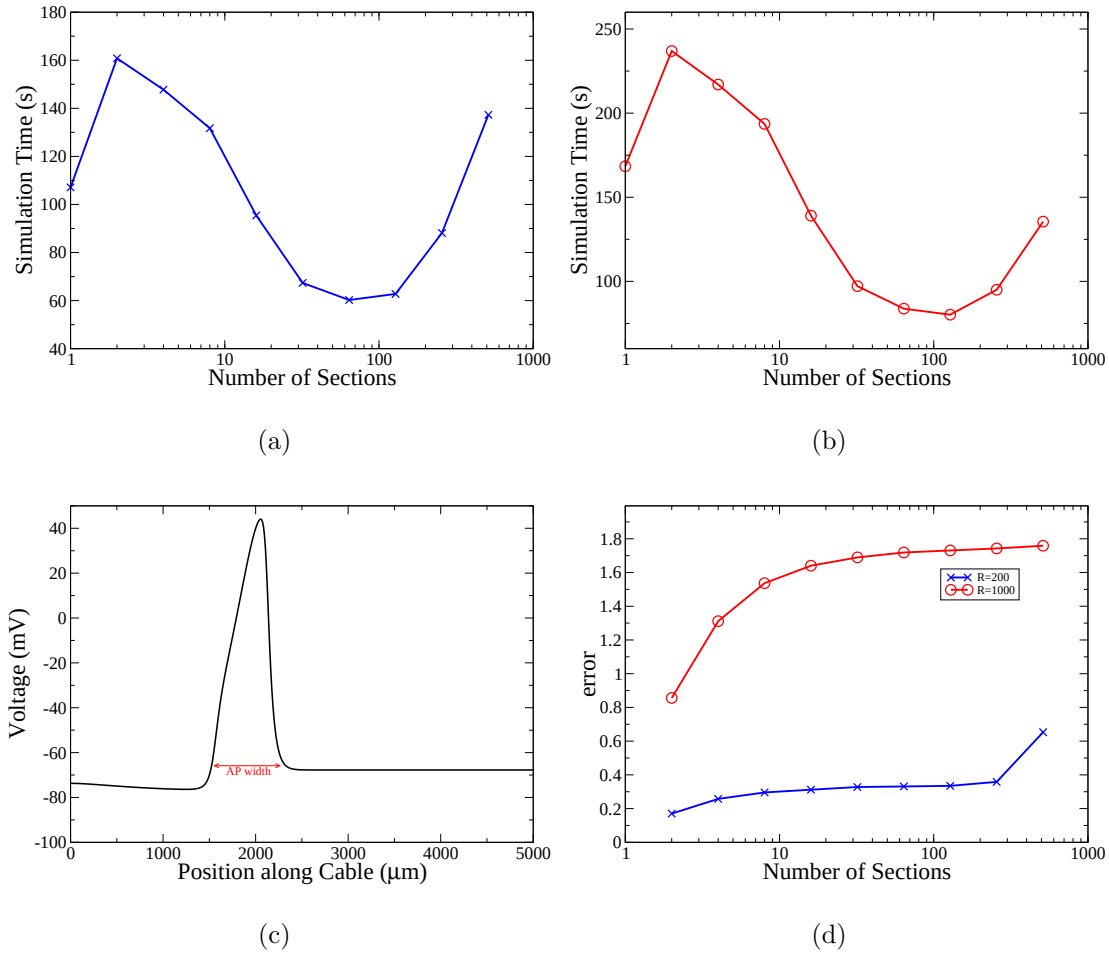


Figure 3.3. **Effect of Section Size on Efficiency of LATS method.** *Figures (a) and (b) show the computational time for simulations of an action potential through a 51200 µm long cable for an increasing number of compartments. The axial resistivity (R_a) was altered to generate action potentials with different spatial width, measured as shown in (c). For R_a=200, the action potential is 960µm wide, and the fastest simulation occurs when the cable is divided into 64 sections of length is 800µm (a). When R_a is increased to 1000, giving an action potential width of 420µm, the optimal section length is 400µm, occuring when the cable is divided into 128 sections (b). The phase errors, determined by the time of the peak of the action potential at the end of the cable are not significantly affected by the number of sections (d).*

potential to reach the end of the cable, and phase errors in the solutions were determined from the time of the peak of the action potential in the last compartment of the cable. As

shown in Figure 3.3(a), the simulation time for this problem was smallest when the cable was divided into 64 sections, each 800μm in length. In this problem, the initial rise and fall of the propagating action potential, measured as shown in Figure 3.3(c), is 960μm. To further demonstrate that the optimal efficiency found in the first test corresponds to the physical properties of the action potential, and not the ratio of spatial grid points and sections, a second test was performed. By increasing the axial resistiviy, R_a, from 200Ωcm to 1000Ωcm, the width of the action potential was decreased from 960μm to 420μm. The series of simulations was repeated, with only the total simulation time increased, to allow the now more slowly propagating action potential to reach the end of the cable. In this case, the narrower action potential resulted in the optimal section size of 400μm, obtained when the cable was divided into 128 sections. Since the total number of grid points remained the same, this test strongly suggests that the optimal section size is dependent on the width of the action potential.

3.4. Improvement over Spatial Adaptivity

Looking at the results in Figure 3.2, one may question the benefit of using locally adaptive time stepping compared to the simpler spatial adaptivity. For the test shown in Figure 3.2, the action potential was only allowed to propagate a short distance along the cable. The individual sections of the cable were either at the resting potential, or experiencing the peak of the action potential. This means that in the locally adaptive time stepping method, sections used either a small Δt comparable to that used by the non-adaptive methods, or took very large steps. This is essentially the same as the results of the spatially adaptive method, where the first few sections, containing the action potential,

were updated while the remaining were kept at rest and were not evolved in time. The benefits of locally adaptive time stepping are most pronounced during the recovery phase, when the voltage changes slowly but is far enough from rest that the spatially adaptive scheme must still update the section. At these times, the local time step adaptivity allows the sections in recovery to evolve with much larger step sizes, allowing for more efficient solution of the problem as will be demonstrated in Section 3.5.

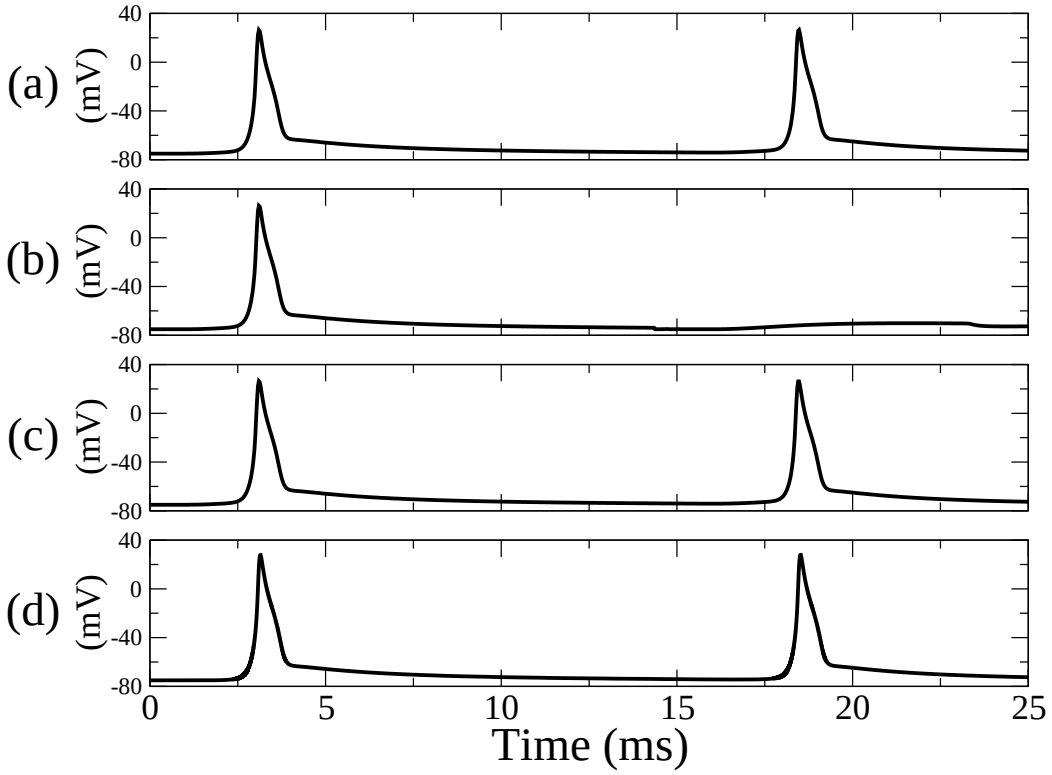


Figure 3.4. **Result of two stimuli on a unbranched cable.** *A stimulus of 0.19 nA at time T=0 initiates an action potential, and a second stimulus at T=15 of 0.18 nA, initiates an action potential when the membrane voltage is above rest. Shown are the numerical results for the non-adaptive BDF scheme (a), the spatially adaptive method with "off" tolerance 0.005 (b) and 0.003 (c), and the locally adaptive time stepping scheme (d). With the more aggressive tolerance, the spatially adaptive method fails to capture the second action potential.*

An additional benefit of locally adaptive time stepping, is that it is more robust than the spatial adaptivity of Rempe and Chopp. In neural systems, the kinetics of individual types of ion channels can occur on a variety of time scales, and may vary between problems. As a result, the tolerances associated with the spatial adaptivity need to be tuned for each problem, as described by Rempe et al. [**55**]. We now describe an instance where the choice of tolerance qualitatively changes the results obtained. A cell consisting of sections forming an unbranched cable, 500 μm in length was stimulated for 1 ms with a 0.19 nA current, triggering an action potential. A second stimulus of 0.18 nA was applied at time 15 ms. Under normal conditions a stimulus of 0.18nA is insufficient to trigger an action potential. However, due to the initial stimulus, the stimulated section of the cell has a slightly elevated voltage, and the second, smaller stimulus produces a second action potential. Figure 3.4(a) shows the two action potentials in a time course recorded from the far end of the cable. The spatial adaptivity algorithm relies on 3 tolerances. Sections of the cell that are at, or near, the resting voltage are not updated until activity in a neighboring section increases above the "on" tolerance at which point the section is "activated". Active sections become inactivated when activity in the section falls below the "off" tolerance, provided the membrane voltage is within the voltage tolerance, V_{tol}, of the resting voltage. When a section is inactivated, all state variables (V, m, h, n in the Hodgkin-Huxley equations (1.13)) are returned to their resting values. Figures 3.4(a) and (b) show the resulting solutions of our test problem utilizing "off" tolerances of 0.005 and 0.003 respectively. When choosing the tolerances, one must balance the desired speed of computation with accuracy, as described by Rempe et al. [**55**]. Typically, the more aggressive tolerances will result in a faster simulation, but,

as shown in Figure 3.4, may result in an incorrect solution. Figure 3.4(d) shows the same simulation computed with the locally adaptive time stepping scheme, using an activity tolerance $\tau = 0.0075$, capturing both action potentials.

3.5. Comparison of Numerical Methods: Chains of Cells Connected by Synapses

A branch of current research in neuroscience involves simulation of large scale networks of cells, evidenced by the development of the Blue Brain Project [**38**], and extensions to the NEURON simulation environment to facilitate large networks of morphologically realistic cells [**43**]. In the final test of our algorithm, we considered chains of morphologically realistic cells connected by synapses, representative of large-scale problems. To make our chain of cells, we took the digital representation of the neuron presented in Figure 1.1, and copied the morphology as many times as needed to produce the system in question. The individual cells were connected together via a single synaptic connection linking the end compartment of the axon to the cell body of the next cell in the chain, as shown in Figure 3.5(a). When an action potential arrives at the presynaptic terminal, at the end of the axon, the synaptic activity is approximated by a current injection into the post-synaptic compartment that mimics the excitatory post synaptic potential (EPSP) observed experimentally. For the purposes of our demonstration, the size of the synaptic conductance was increased to the point where a single synaptic event was sufficient to trigger an action potential in the post-synaptic cell.

Activity in our system was initiated by a synaptic-like current injection into the cell body of the initial cell in the chain, and simulations were run for 7 ms per cell, a sufficiently

long time to allow the action potential to propagate through the entire system and reach the end of the axon in the final cell. Because the length of the simulation increases with the number of cells, the computational cost of the BDF method without adaptivity grows quadratically, as shown in Figure 3.5(b). In the spatially adaptive case, only a fraction of the sections in the computational domain are in the "active" state, providing improved efficiency over the non-adaptive method. However, the number of active sections increases with the problem size, as shown in Figure 3.5(c), causing the computational time to grow quadratically in this case as well, though more slowly than in the non-adaptive method.

For small problems, with fewer than 15 cells in the chain, the locally adaptive time stepping scheme is actually more expensive than the spatially adaptive method. In addition to the extra computational overhead involved in adaptive time stepping, the step size selection method often chooses a much smaller time step during the peak of the action potential than is used by the fixed step methods. In simulations of short duration this results in the adaptive time stepping method actually computing the solution at a greater number of (unequally spaced) time points than the fixed step methods. As the length of the simulation increases, the ratio between the number of small and large steps taken by the locally adaptive time stepping scheme becomes more favorable, and in total the method performs fewer computations than the fixed step methods. Additionally, the locally adaptive time stepping method focuses computational resources more narrowly than the spatially adaptive method, leading to a linear increase in the computational time relative to the number of cells in the chain.

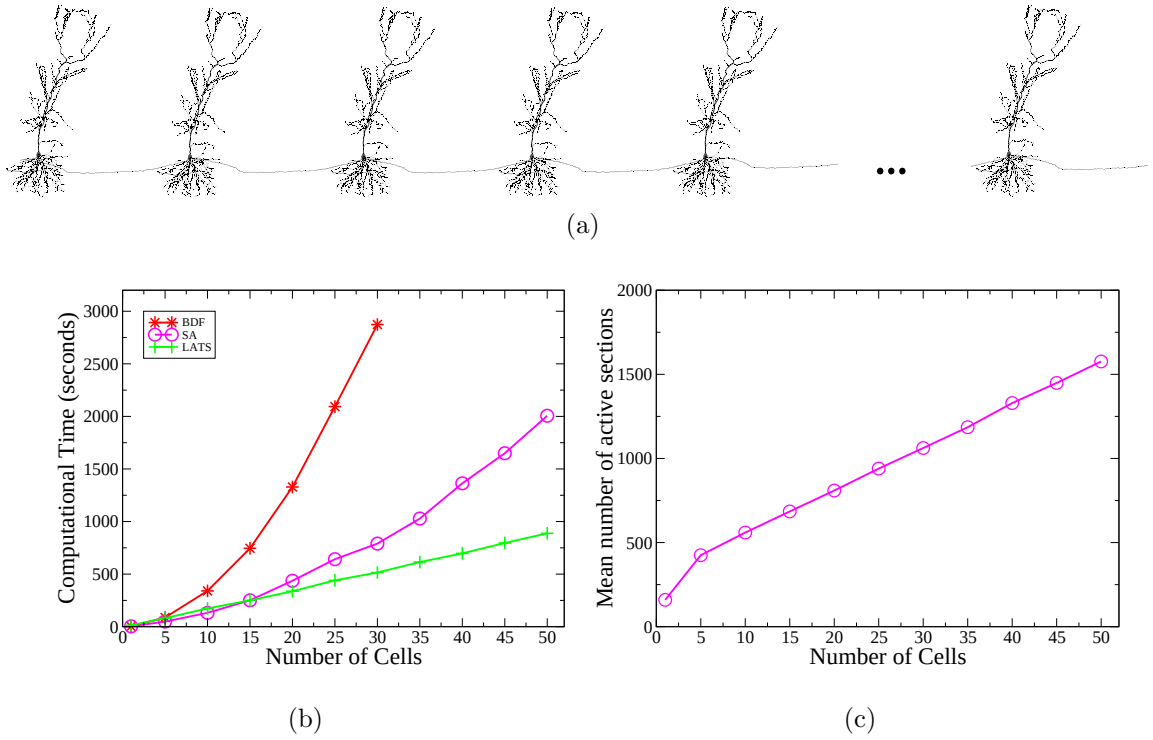


Figure 3.5. **Timing comparison of numerical methods for chains of cells connected by synapses.** *An increasing number of cells were connected via synapses to form chains (a). Activity in the system was initiated through a synaptic-like current injection into the cell body of the initial cell. Simulations were run for long enough for the action potential to propagate through the entire system and reach the end of the axon in the terminal cell. The computational time of three methods is presented in (b): The BDF method with no adaptivity (BDF), the spatially adaptive method of Rempe and Chopp (SA), and the locally adaptive time stepping scheme (LATS). Shown in (c) is the average number of active sections during each of the simulations for the spatially adaptive method. Since the number of active sections grows linearly with the size of the problem, and the length of time increases as well, this leads to the computational time growing quadratically. In the same way, the non adaptive BDF scheme demonstrates quadratic growth in computational time. In contrast, the locally adaptive time stepping scheme focuses computational power much more narrowly and consequently the computational time grows linearly with the length of the simulation.*

3.6. Closed Loop

In some cases, cells in a network system can be joined through direct electrical connections, called gap-junctions. When the gap junction connections form a closed loop in the system, the standard computational methods lose their efficiency. For solvers that treat the entire network as a single system, the matrix inversion required can no longer be completed with the same level of efficiency and the computational cost grows. For a system without closed loops, Hines' ordering of the sections and compartments the matrix inversion can be completed with O(N) operations, however when closed loops are present the computational cost of the matrix inversion can grow to $O(N^3)$ [**25**].

Because our domain decomposition scheme divides the problem into smaller, uncoupled problems, our method is able to handle cases containing closed loops with the same efficiency observed in other cases. To demonstrate this, we created a cable and connected the ends together to form a torus, and divided it into 10 sections. The simulation time for action potential propagation through this loop was compared to a straight cable of the same length. The cable was stimulated in the center and the simulation was run for 2ms, allowing the two resulting action potentials to propagate away from the point of stimulation. Figure 3.6 shows the simulation time for the straight cable, and the closed loop examples. In both cases the same simulation was computed over the same system with an increasingly refined spatial grid. As the total number of grid points in the system grows, the computational cost grows linearly, as expected, and the total simulation time is the same for both the straight cable and closed loop cases.

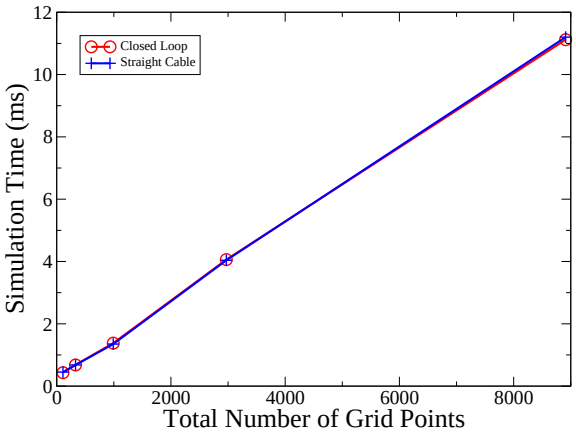


Figure 3.6. **Computational Efficiency on Closed Loops.** *As the size of the problem grows, the simulation time grows linearly. The domain decomposition scheme employed for the locally adaptive time stepping algorithm is as efficient in cases containing a closed loop as in those without.*

3.7. Concluding Remarks

We have presented an algorithm for domain decomposition and locally adaptive time stepping, and applied it to the branched domain present in neural simulations. For large scale problems, this method provides improved efficiency over the spatially adaptive method of Rempe and Chopp [**54**], which in turn provided increased efficiency over the standard methods incorporated into the NEURON simulation package.

As in the case of Rempe and Chopp's spatially adaptive method, we employed a domain decomposition scheme and reduced the large computational problem into smaller, tri-diagonal systems. Our method provides many of the same benefits of the spatially adaptive scheme including the ability to handle systems containing closed loops, and suitability for parallel implementation. As in the case of the commonly used methods, our algorithm requires O(N) updates for each section, where N is the number of grid points, but is stable for a much larger range of step sizes, and provides more robust behavior

through the use of an error-like activity tolerance. Our method also represents a refinement of the binary "on"-"off" behavior, allocating computational resources according to the location and intensity of the activity within the system. While we have presented our algorithm in the context of the neural systems that motivated its development, it can be applied to more general classes of problems and will be beneficial in any system that displays localized activity.

CHAPTER 4

Conclusions and Future Work

Many fields of scientific study rely on computer simulations to test and inform theoretical and experimental work. As computational results have gained greater acceptance, the scale and complexity of simulations have grown and parallel avenues of computational research have emerged. One branch of research applies numerical tools to study the physical system, while another path leads to the development of new computational tools. The locally adaptive time stepping (LATS) algorithm presented in this dissertation represents the latest in a series of algorithms designed to efficiently utilize available computational power.

Early advances in numerical methods were in the development of higher order schemes that allowed larger step sizes while maintaining a constant level of accuracy, and implicit methods that remained stable for large step sizes. Depending on the application, these methods were a considerable improvement over Euler's explicit, first order method. However, higher order explicit schemes are still subject to a limiting Courant-Friedrichs-Lewy (CFL) condition, and the stability of implicit methods is not a guarantee of accuracy. Adaptive time stepping schemes adjust the step size to maintain a consistent level of accuracy throughout the simulation, and can provide considerable savings in cases where short bursts of activity alternate with long periods of slow evolution. In simulations of partial differential equations, activity in a small region of the domain causes a time step restriction on the entire system.

The advancement of domain decomposition schemes, where the spatial domain is split into smaller subdomains, has led to several ways of increasing the speed of numerical simulations. The simplest application of a domain decomposition scheme is a brute-force application of parallel computing. Each subdomain is assigned to a separate processor, and all are updated simultaneously. Spreading the work in this manner speeds the simulation, however, the time step over the whole domain can still be restricted by the level of activity in a small region of space. Subcycling algorithms have been developed to allow different subdomains to evolve with different step sizes, but these are of limited use in practice as the step sizes remain fixed throughout the simulation, and must be determined in advance. For simulations in which the solution is either at rest or changing very rapidly, the algorithm for spatial adaptivity of Rempe and Chopp is attractive. The solution in regions of the domain at rest is not computed, and the active regions are updated with a fixed time step appropriate for the highest level of activity.

While each of the schemes mentioned above are well suited to specific classes of problems, the locally adaptive time stepping algorithm presented in this dissertation is much more generally applicable, and through the use of a single error-like tolerance selects the optimal time step for each subdomain independent of the activity in other regions of the domain. The locally adaptive time stepping scheme incorporates features found in many other algorithms, but is unique in its combination of these features. Domain decomposition combined with adaptive time stepping allows each subdomain to select the most appropriate step size for the local activity level. As in the subcycling algorithms, approximations for boundary data are required when the solution in neighboring subdomains are not available at the required time point and a modification of the spatial adaptivity

scheme is employed to ensure that events of short duration are not ignored by sections taking large steps.

In the context of numerical simulations for neuroscience the locally adaptive time stepping algorithm is novel for a number of reasons. In the commonly used simulation packages and recently developed schemes, the solution over the entire tree structure is updated simultaneously and simulations are done using the Backward Euler or Crank-Nicolson methods, as these are simple to implement, stable and efficient. The simultaneous update is done either as one large system of equations, or as the result of a domain decomposition scheme where the update of the branches is alternated with an update of the branching nodal points. In the algorithm presented in this dissertation, the individual branches of the tree are updated sequentially. The sequential update is not new and a similar approach was presented in 1978 by Joyner *et al.* [**34**], who combined an explicit approximation of the boundary data with the Crank-Nicolson scheme in the interior of each branch. However, in the current work, the choice of the second order Backward Differentiation Formula method and the stability it provides eliminates the instabilities that develop at the branching points with the Crank-Nicolson scheme. As demonstrated in Chapter 3, locally adaptive time stepping dramatically reduces the computational expense of large scale simulations compared to a fixed step method and even the spatially adaptive scheme of Rempe and Chopp.

Our locally adaptive time stepping algorithm was developed and tested in the context of numerical simulations for neuroscience, however it can be applied to many different classes of problems. The predictor-corrector implementation of the second order BDF

scheme results in a method that is as efficient as either Backward Euler or Hines implementation of the Crank-Nicolson scheme, with the improved stability of the BDF methods at little additional cost in storage for recent solution history data. As the BDF schemes are implicit, the scheduling of updates plays an important role in the accuracy of the computed solution. By updating each subdomain at the end of its update step, and ordering the updates so that the fastest evolving sections are updated first we ensure that the most accurate boundary data possible is available. The highly stable nature of the second order BDF method is essential to the locally adaptive time stepping algorithm. Unlike the subcycling techniques, where the ratio of step sizes between subdomains is restricted for stability reasons and frequent synchronization of the solution over the whole domain is required, our scheme requires no explicit synchronization of the solution, and remains stable for very large step size ratios. Indeed, step size ratios of 10^6 remained stable, as shown in Figure 2.6. This combination of features found in a variety of algorithms, results in an algorithm that is not limited by a global time step restriction. Our scheme automatically focuses computational power where it is needed, resulting in faster simulations without requiring the user to have detailed knowledge of the structure of the solution.

4.1. Future Directions

The locally adaptive time stepping algorithm presented in this dissertation has been applied to problems in neuroscience, resulting in faster simulations and the ability to tackle larger, more complex networks. The algorithm, and accompanying computer code are easily adapted to future neuroscience problems, and will be of particular benefit for large-scale simulations.

The algorithm is not restricted to the unique problem of simulation of the Hodgkin-Huxley equations on branching neural networks. Extensions to higher dimensional spatial domains is relatively straightforward, and many of the key design features will be beneficial in any system containing localized activity. This localization of activity can be seen in the retina model of Chang *et al.* [**9**] in which individual cells and inter-cellular connectivity in the retina are modelled as activity on a two dimensional conductive sheet. Localized activity also exists in pattern formation in reactive media and in the synchronization of oscillators. Danino *et al.* [**14**] present a one dimensional model of the two dimensional patterns observed experimentally, and our locally adaptive time stepping algorithm combined with an extension of their model to higher dimensions may provide further insight into the observed phenomena.

References

[1] Dganit Amitai, Amir Averbuch, Moshe Israeli, and Samuel Itzikowitz. Implicit-explicit parallel asynchronous solver of parabolic PDEs. *SIAM Journal on Scientific Computing*, 19:1366, 1998.

[2] Mark F Bear, Barry W Connors, and Michael A Paradiso. *Neuroscience: Exploring the Brain*. Lippincott Williams & Wilkins, Baltimore, Maryland, 1996.

[3] Ted Belytschko and Noreen D Gilbertsen. Implementation of mixed time integration techniques on a vectorized computer with shared memory. *Int. J. Numer. Meth. Engng.*, 35(9):1803–1828, 1992.

[4] Ted Belytschko, HJ Yen, and R Mullen. Mixed methods for time integration. *Computer Methods in Applied Mechanics and Engineering*, 17:259–275, 1979.

[5] J.M. Bower and D. Beeman. *The Book of Genesis: Exploring Realistic Neural Models with the GEneral NEural SImulation System*. Telos, Santa Clara, 1998.

[6] Peter N Brown, George D Byrne, and Alan C Hindmarsh. VODE: a variable-coefficient ODE solver. *SIAM Journal on Scientific and Statistical Computing*, 10(5):1038–1051, 1989.

[7] George D Byrne and Alan C Hindmarsh. A polyalgorithm for the numerical solution of ordinary differential equations. *ACM Transactions on Mathematical Software*, 1(1):71–96, 1975.

[8] Nicholas T Carnevale and Michael L Hines. *The NEURON Book*. Cambridge University Press, New York, NY, USA, 2006.

[9] S Chang, S Baer, S Crook, and C Gardner. Modeling the gaba and ephaptic feedback mechanisms in cat outer retina. *BMC Neuroscience*, Dec 2008.

[10] Scott D Cohen and Alan C Hindmarsh. CVODE, a stiff/nonstiff ODE solver in C. *Computers in Physics*, 10(2):138–143, 1996.

[11] JW Cooley and FA Dodge. Digital computer solutions for excitation and propagation of the nerve impulse. *Biophys. J.*, 6(5):583–599, 1966.

[12] R Courant, K Friedrichs, and H Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, Jan 1928.

[13] WJT Daniel. Subcycling first- and second-order generalizations of the trapezoidal rule. *Int. J. Numer. Meth. Engng.*, 42(6):1091–1119, Jan 1998.

[14] T Danino, O Mondragón-Palomino, L Tsimring, and J Hasty. A synchronized quorum of genetic clocks. *Nature*, Jan 2010.

[15] Clint Dawson, Qiang Du, and Todd Dupont. A finite difference domain decomposition algorithm for numerical solution of the heat equation. *Mathematics of Computation*, 57(195):63–71, Jul 1991.

[16] Peter Dayan and L F Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, Cambridge, Massachusetts, 2001.

[17] FA Dodge and B Frankenhaeuser. Membrane currents in isolated frog nerve fibre under voltage clamp conditions. *J Physiol-London*, 143(1):76–90, Jan 1958.

[18] Richard Fitzhugh. Computation of impulse initiation and saltatory conduction in a myelinated nerve fiber. *Biophys. J.*, 2(1):11–&, Jan 1962.

[19] CW Gear. Algorithm 407: DIFSUB for solution of ordinary differential equations [D2]. *Communications of the ACM*, 14(3):190, 1971.

[20] CW Gear. The automatic integration of ordinary differential equations. *Communications of the ACM*, 14(3):179, 1971.

[21] Nace L Golding, William L Kath, and Nelson Spruston. Dichotomy of action-potential backpropagation in CA1 pyramidal neuron dendrites. *Journal of neurophysiology*, 86(6):2998–3010, 2001.

[22] L Goldman and JS Albus. Computation of impulse conduction in myelinated fibers - theoretical basis of velocity-diameter relation. *Biophys. J.*, 8(5):596–&, Jan 1968.

[23] WL Hardy. Propagation speed in myelinated nerve:: I. experimental dependence on external Na+ and on temperature. *Biophys. J.*, 13(10):1054–1070, Jan 1973.

[24] WL Hardy. Propagation speed in myelinated nerve:: II. theoretical dependence on external Na+ and on temperature. *Biophys. J.*, Jan 1973.

[25] Michael L Hines. Efficient computation of branched nerve equations. *International Journal of Bio-Medical Computing*, 15(1):69–76, 1984.

[26] Michael L Hines and Nicholas T Carnevale. The neuron simulation environment. *Neural Computation*, 9(6):1179–1209, 1997.

[27] Michael L Hines and Nicholas T Carnevale. Neuron: a tool for neuroscientists. *The Neuroscientist*, Jan 2001.

[28] Alan Lloyd Hodgkin and Andrew Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo. *J Physiol-London*, 116(4):449–472, Jan 1952.

[29] Alan Lloyd Hodgkin and Andrew Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, 117:500 – 544, 1952.

[30] Alan Lloyd Hodgkin and Andrew Huxley. The components of membrane conductance in the giant axon of loligo. *J Physiol-London*, 116(4):473–496, Jan 1952.

[31] Alan Lloyd Hodgkin and Andrew Huxley. The dual effect of membrane potential on sodium conductance in the giant axon of loligo. *J Physiol-London*, 116(4):497–506, Jan 1952.

[32] Alan Lloyd Hodgkin, Andrew Huxley, and B Katz. Measurement of current-voltage relations in the membrane of the giant axon of loligo. *J Physiol-London*, 116(4):424–448, Jan 1952.

[33] NA Hutchins, ZJ Koles, and RS Smith. Conduction velocity in myelinated nerve fibres of xenopus-laevis. *J Physiol-London*, 208(2):279–&, Jan 1970.

[34] R W Joyner, M Westerfield, J W Moore, and N Stockbridge. A numerical method to model excitable cells. *Biophys. J.*, 22(2):155–170, 1978.

[35] ZJ Koles. A computer simulation of conduction in demyelinated nerve fibres. *The Journal of Physiology*, Jan 1972.

[36] Yu A Kuznetsov. New algorithms for approximate realization of implicit difference schemes. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 3(2):99–114, Dec 1988. doi: 10.1515/rnam.1988.3.2.99.

[37] Jack Lee, Bruce Smaill, and Nicolas Smith. Hodgkin-Huxley type ion channel characterization: An improved method of voltage clamp experiment parameter estimation. *Journal of theoretical Biology*, Jan 2006.

[38] H Markram. The blue brain project. *Nat Rev Neurosci*, 7(2):153–160, Jan 2006.

[39] Michael Mascagni. The Backward Euler method for numerical solution of the Hodgkin–Huxley equations of nerve conduction. *SIAM Journal on Numerical Analysis*, 27(4):941–962, 1990.

[40] Michael Mascagni. A parallelizing algorithm for computing solutions to arbitrarily branched cable neuron models. *Journal of Neuroscience Methods*, 36(1):105–114, 1991.

[41] The MathWorks Inc., Natick Massachusetts. *MATLAB version 7.5.0.338*, 2007.

[42] Claude Meunier and Idan Segev. Playing the Devil's advocate: is the Hodgkin-Huxley model useful? *Trends in Neurosciences*, Jan 2002.

[43] M Migliore, C Cannia, William W Lytton, Henry Markram, and Michael L Hines. Parallel network simulations with neuron. *J Comput Neurosci*, 21(2):119–129, Jan 2006.

[44] J W Moore, R W Joyner, M H Brill, S D Waxman, and M Najar-Joa. Simulations of conduction in uniform myelinated fibers. relative sensitivity to changes in nodal and internodal parameters. *Biophys. J.*, 21(2):147–160, 1978.

[45] J W Moore and F Ramon. On numerical integration of the hodgkin and huxley equations for a membrane action potential. *Journal of theoretical Biology*, 45(1):249, 1974.

[46] J W Moore, F Ramon, and R W Joyner. Axon voltage-clamp simulations. i. methods and tests. *Biophys. J.*, 15(1):11–24, 1975.

[47] MO Neal and Ted Belytschko. Explicit explicit subcycling with non-integer time step ratios for structural dynamic-systems. *Comput Struct*, 31(6):871–880, Jan 1989.

[48] Phyllis Nicolson and John Crank. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *P Camb Philos Soc*, 43(1):50–67, Dec 1947.

[49] D Noble. Applications of Hodgkin-Huxley equations to excitable tissue. *Physiol Rev*, Jan 1966.

[50] W Rall. Membrane time constant of motoneurons. *Science*, 126(3271):454–454, Jan 1957.

[51] W Rall. Branching dendritic trees and motoneuron membrane resistivity. *Exp Neurol*, 1(5):491–527, Jan 1959.

[52] W Rall. Membrane potential transients and membrane time constant of motoneurons. *Exp Neurol*, 2(5):503–532, Jan 1960.

[53] F Ramon, R W Joyner, and J W Moore. Propagation of action potentials in inhomogeneous axon regions. *Fed Proc*, 34(5):1357–1363, Jan 1975.

[54] Michael J Rempe and David L Chopp. A predictor-corrector algorithm for reaction-diffusion equations associated with neural activity on branched structures. *SIAM Journal on Scientific Computing*, 28(6):2139–2161, 2006.

[55] Michael J Rempe, Nelson Spruston, William L Kath, and David L Chopp. Compartmental neural simulations with spatial adaptivity. *J Comput Neurosci*, 25(3):465–480, Dec 2008.

[56] Lawrence F. Shampine. *Numerical Solution of Ordinary Differential Equations*. Chapman & Hall, New York, NY, 1997.

[57] G H Sharp and R W Joyner. Simulated propagation of cardiac action potentials. *Biophys. J.*, 31(3):403–423, 1980.

[58] Han-Sheng Shi and Hong-Lin Liao. Unconditional stability of corrected explicitimplicit domain decomposition algorithms for parallel approximation of heat equations. *SIAM Journal on Numerical Analysis*, 44:1584, 2006.

[59] P Smolinski. Subcycling integration with non-integer time steps for structural dynamics problems. *Comput Struct*, 59(2):273–281, Jan 1996.

[60] P Smolinski and YS Wu. An implicit multi-time step integration method for structural dynamics problems. *Computational Mechanics*, 22(4):337–343, 1998.

[61] P Smolinski and YS Wu. Stability of explicit subcycling time integration with linear interpolation for first-order finite element semidiscretizations. *Computer Methods in Applied Mechanics and Engineering*, 151(3-4):311–324, Jan 1998.

[62] SP Xiao and Ted Belytschko. A bridging domain method for coupling continua with molecular dynamics. *Computer Methods in Applied Mechanics and Engineering*, 193(17-20):1645–1669, 2004.

[63] Yu Zhuang and Xian-He Sun. Stable, globally non-iterative, non-overlapping domain decomposition parallel solvers for parabolic problems. *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, page 19, 2001.

[64] Yu Zhuang and Xian-He Sun. Stabilized explicit-implicit domain decomposition methods for the numerical solution of parabolic equations. *SIAM Journal on Scientific Computing*, 24(1):335–358, 2002.