# The Application of Adaptive Step-Size Control in the Numerical Simulation of Calcium Release in Vascular Smooth Muscle

Michael John Sydor

Thesis submitted to the Faculty of the Graduate School
of the New Jersey Institute of Technology
in partial fulfillment of the requirements for the degree of

Master of Science in Biomedical Engineering

May 1991

# Approval Sheet

Title of Thesis : The Application of Adaptive Step-Size Control in the Numerical Simulation of Calcium Release in Vascular Smooth Muscle.

Name of Candidate : Michael J. Sydor
Master of Science, 1991

Thesis and Abstract Approved :

Dr. F. P. J. Dierke — Date
Chairman
Department of Physiology
University of Medicine and Dentristy of New Jersey

Signatures of other members of the thesis committee.

Dr. E. V. Hersh — Date
Associate Professor
Director of Pharmacology & Clinical Therapeutics
University of Pennsylavania

Dr. D. Kristol — Date
Chairman
Biomedical Engineering Program

Dr. L. Michaelson — Date
Director
Research Computing Services
University of Medicine and Dentristy of New Jersey

# VITA

Name : Michael John Sydor

Permanent Address : 68 Spring Street, Metuchen, New Jersey, 08840

Degree and date to be conferred : M. S. Biomedical Engineering, June 1991

Secondary Education : Metuchen High School, 1976

| Collegiate institutions attended | Dates | Degree | Date of Degree |
|---|---|---|---|
| New Jersey Institute of Technology | 9/86- 6/87 | M.S.Biomed.Eng. | 6/91 |
| New Jersey Institute of Technology | 9/80-12/83 | B.S.Eng.Sci | 6/84 |

Major : BioMedical Engineering

Publications :

Bone Remodeling Under Stress, Biomedical Research at the New Jersey Institute of Technology, June 1984 (Abstract)

3-D Reconstruction of Small Blood Vessels, 27th Am. Meeting ASCB, St. Louis, Missouri, Nov 1987 (Poster & Abstract)

Computerized 3-D Reconstruction of Small Blood Vessels from High Voltage Electromicrographs of Thick Serial Cross Sections, Satellite Symp. IV World Cong. Microcir. Aug 1987, "Vascular Endothelium in Health and Disease" (Abstract)

Morphological Analysis of Blood Vessels Using High Voltage Electron Microscopy and 3-D Computerized Reconstruction, Proceedings of the Ninth Annual Conference of the IEEE Engineering in Medicine and Biology Society, Ch. 2513-0/87 Vol. 3; 1682-1684 (1987)

Computerized 3-D Reconstruction of Small Mesenteric Arteries from Normotensive and 1-Clip-2 Kidney Goldblatt Chronic Hypertensive Dogs Using High Voltage Electromicrographs of Thick Serial Cross Sections, Adv. Exp. Med. Biol. 242:35-42 (1988)

Positions Held :

Applications Engineer : Biosym Technologies, San Diego, CA.
Senior Systems Engineer : Concurrent Computer Corporation, Tinton Falls, NJ.
Consultant: Software Applications and Engineering, South Orange, NJ .
Teaching Assistant : Department of Physics, NJIT, Newark, NJ.
Engineer : George Shultz Laboratory for Orthopaedic Research, UMDNJ, Newark, NJ.
Development Manager, LabSoft, New York, NY.
Research Technician : General Foods Corporation, Cranbury, NJ.
Chemist : Pilot Chemical Company, Avenel, NJ.

# Abstract

**Title of Thesis :** The Application of Adaptive Step-Size Control in the Numerical Simulation of Calcium Release in Vascular Smooth Muscle.

Michael J. Sydor, Master of Science in Biomedical Engineering, 1991

**Thesis directed by :** Dr. F. P. J. Diecke, Chairman of Physiology

An algorithm for the adaptive control of numerical integration step-size is developed and implemented for the simulation of a three compartment model for Vascular Smooth Muscle. The three compartment model accounts for the simultaneous diffusion of Ca, $^{45}$Ca, EGTA, Ca-EGTA, and $^{45}$Ca-EGTA, and is an extension of a two compartment model by Diecke for the simultaneous diffusion of Ca, EGTA, and Ca-EGTA. The addition of the third compartment is to account for the presence of the Sarcoplasmic Reticulum which stores the calcium needed for contraction and is the primary regulator of calcium in the VSM cell. The SR has been implicated as the slow component in calcium release, as measured by Stout and Diecke in saponin skinned VSM.

The compartmental model is developed from mass-balance equations and is solved numerically with a Runge-Kutta-Gill algorithm. Step-size is controlled with an adaptive algorithm which adjusts the integration interval (step-size) for the transient and steady-state phases of the simulation. The implementation of the various programs is designed to accommodate automated execution and analysis of a high volume of simulation runs. Some introduction into the methodology of modeling and simulation, as well as the complex physiology of the SR is discussed and the complete process of modeling, simulation, and analysis is illustrated for a simple model of a two-compartment leaky tank. A more comprehensive introduction into numerical integration is included to provide sufficient background for the development of the adaptive algorithm.

The adaptive algorithm allows a complex simulation to be executed in one-fifth the time required for constant step (non-adaptive) numerical integration without incurring significant error. This reduction in the amount of computer time required permits more aggressive protocols for the determination of parameters and model responses by allowing more simulation runs to be processed in the course of a study. The simulation of calcium release from VSM revealed that the response of the system is not a multiple of the increase in rate but is instead related via a linear function representative of the buffering capacity of the model. Results from a similar two-compartment model suggest that the EGTA buffer system has a significant impact of the perceived rates of release.

This Thesis is dedicated to the memory of my friend Richard "Butter" Waterworth.

## Acknowledgments

The completion of this thesis would not have been possible without the measured patience and timely guidance of Dr. Diecke.

I am also indebted to Dr. Lief Horn for his suggestions and guidance, particularly in his reading of the more detailed drafts and his thoughtful prodding.

Dr. Michaelson has often been the last check point, in this and a number of other projects, when I wandered out of bounds in the application of computers. Whatever success I obtain by in this field will be in part attributable to his clarity of thought and perspective in problem solving.

The fundamental shift in my facility for technical writing can be attributed to Dr. Hersh, both in his suggestions, and his requirements as a potential user of computer modeling techniques.

For Dr. Kristol I can only hope that this dividend is sufficient, for having guided my efforts over the years and enduring my particular insensitivity to the passage of time.

This project was made more interesting with the use of Dr. Art Ritter's computer and his library, where I located the discussion on the initial form of the algorithm -- the root of my thesis.

# Table of Contents

## Conclusions



## Appendix

# List of Figures

# Introduction

The application of computer technology is an important tool for the illumination of Biological systems. In particular, the numerical solution of mathematical models for physiological systems and processes has enabled the elucidation of many complex biological systems. These systems are often difficult to manage experimentally and the use of a computer model, or *simulation*, often provides the only reliable platform on which the underlying physical processes may be explored.

While the development and parameterization of a physiological model is frequently the more demanding process, the actual simulation of the model has become a significant problem. As the growth of experience and acceptance in the area of computer modeling has established a rational framework for model design, it has become reliable to construct larger systems composed of simple, well-known components. These simple components, such as 1$^{st}$ order decay processes, kinetic processes, and simple diffusion, have been long established experimentally. Assemblies of these simple components, in the construction of sophisticated and detailed models (Kootsey), increase the computational demands of the ensuing simulations due to the range and complexity of model system responses. The question is: *can the computational efficiency of a simulation be significantly increased without compromising the physiological accuracy?*

The traditional response to the increasing computational demands of contemporary simulation has been to seek faster and more capable computer systems. Despite the additional costs (both in dollars and the personnel time needed to exploit newer technology) the growth in the technical level simulation has not been impeded. The arguable point is whether the growth in the number of practitioners even remotely parallels that of the technical achievements.

This thesis serves to illustrate the simulation task of the modeling process by taking a two-compartment physiological model and extending it one conceptual increment, to three-compartments. This extension introduces a variety of pitfalls computationally which are addressed by improving the efficiency of the simulation engine. Achieving this efficiency entails the development and implementation of an adaptive step-size algorithm, which is the major contribution of this thesis. A number of other enhancements for the generation and analysis of simulations are introduced so as to provide a robust platform on which to further extend the modeling of vascular smooth muscle and other related compartment models.

The content of the thesis can be summarized as follows:

The Introduction includes some discussion of Vascular Smooth Muscle to provide background for the variety of experimental problems that are encountered. Following this is a review of the two-compartment model on which this project is based. Finally, the objectives of this project are summarized.

The Material of the thesis is the compartmental model and its extension to three-compartments. The Methods for the thesis are the programs, software methods and techniques for the analysis of simulation results.

The Discussion provides the evidence as to the success of the adaptive step-size algorithm and introduces the experiments which validate the software implementation, improve the physiologic reality of the model, and explore the effect of the third compartment on the release of calcium in vascular smooth muscle.

The Conclusion summarizes the contribution of the methods developed for the thesis and indicates potential areas for extension of the three-compartment model and towards simulation in general.

## Background

In the case of smooth muscle used in the human vascular system, defects in function have chronic effects. In the design of strategies and drugs to alleviate these conditions, the major focus is on the regulation of calcium used by smooth muscle. A major regulator of calcium, used in muscle contraction, is the Sarcoplasmic Reticulum (SR). All of the three types of muscle -- skeletal, cardiac and smooth -- are dependent on the SR to provide the calcium necessary for contraction.

The mechanism by which smooth muscle affects the vascular system is contraction. Smooth muscle has other properties, such as its ability to synthesize products, and its manner of growth and development, but these are not as significant. Any potential regulation of the SR needs to be very precise because the the two sides of the vasculature, arterial and venous, have contradicting effects for the same mode of operation. For example, contraction for the arterial side causes an increase in blood pressure which decreases cardiac output, while contraction on the venous side causes an increase in cardiac output. These differences are largely attributable to the architecture of the respective vessels but illustrate that a "simple fix", such as relaxing the vasculature to correct a hypertensive state, potentially causes more problems by decreasing the cardiac output and increasing the strain on the heart. The underlying regulatory process and the environment need to be understood.

For smooth muscle, this is a considerable challenge for a number of reasons. The first is the overall duration of contraction. Smooth muscle is not under

general active control as is skeletal muscle and instead responds to a variety of nervous and hormonal factors. Its physical size and distribution make it difficult to isolate and its internal cellular structures differ significantly from skeletal muscle, which has been more extensively characterized. The second is the selection of a suitable animal model for experimental use. Each component of the vasculature, from arteries through capillaries through veins, has different characteristics, in addition to the overall differences between the two sides of the vasculature. There are also differences between the same type of tissue taken from different animals. The overall result is that literature findings are not transferrable and progress in understanding the regulatory mechanisms has been slow.

The physical attributes of smooth muscle are well known and because the various elements have been characterized in other tissues, particularly skeletal muscle (Peachey), it is possible to construct a mathematical model which relates to each of the elements. Before conclusions can be drawn about the accuracy of a given model, it is necessary to validate its prediction with physical experiment; this remains an ongoing problem. Although it is straightforward to construct models of great complexity, it it unlikely that they can ever be validated experimentally. Modeling best proceeds by "leap-frogging" established experimental results in order to provide the direction for a new suite of experiments (Kootsey). And as each model has been validated, another hypothesis is added to continue the process of modeling and experimentation.

While the generation of the model, which expresses the process, is often straightforward, the mathematical simulation of the resulting model may not be as simple. This project is an outgrowth of an modeling increment which encounters a computational and process bottleneck, indicating that even a conservative increment in the model can result in a radical increase in challenging problems.

## Previous work

The present simulation model and program (or simulation engine) began as an extension to a local implementation by Roeseler. The nomenclature for the calcium exchange was implemented by Hausser. Both the original program and the model nomenclature have been substantially modified in order to streamline and automate the simulation process.

In previous work by Diecke and Hausser, where the two-compartment model was implemented and its parameters for diffusion validated, it was not possible to directly extend the model to three-compartments because the additional equations extended the simulation duration beyond a reasonable limit. At that time, the simulations were run on an IBM PC and took about 12 to 20 hours to complete. As the run times increased into the order of days,

the reliability of the machines became an issue and many unforeseen problems arose.

## Particular Areas to be Addressed

The following Figure 1 summarizes the extension to the existing two-compartment model as suggested by Diecke. The horizontal arrows indicate diffusion between compartments while the vertical arrows indicate a reaction within a compartment. The relative volumes are indicated above each compartment. The compartments are in series indicating that the only path to the SR, for example, is through the other two-compartments. The reservoir is considered to be a homogeneous pool (the concentrations do not vary with distance) of the components at some concentration and is referred to as a *forcing function* when considered in the mass balance equations.

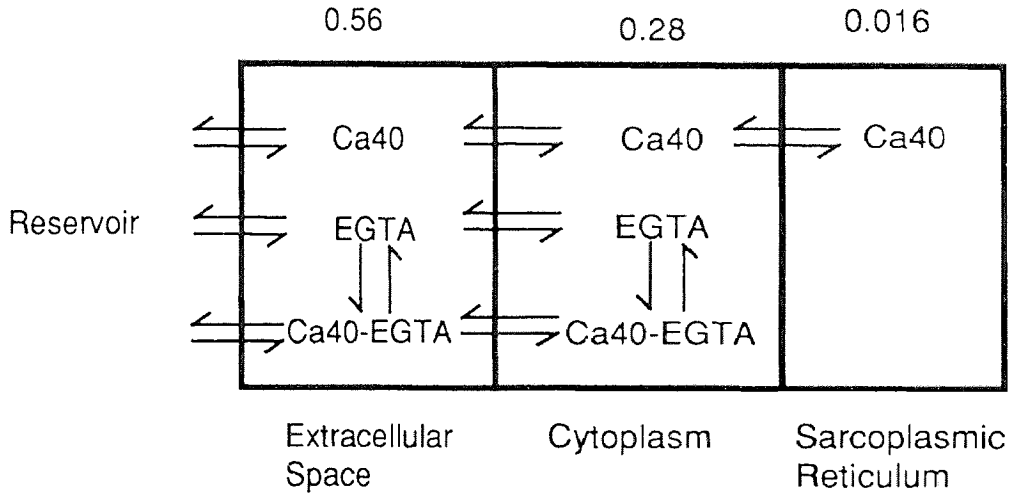


Figure 1 - Proposed Three-compartment Model

Along with the addition of a third compartment, it was desired as well to use a model which better represented the actual experimental conditions, so far as the species that were tracked by the model. The addition of equations to model $^{45}Ca$ effectively doubled the number of equations to be evaluated. This increased run times into the order of weeks and the results were completely unreliable.

Clearly, the limitation on the number of equations was unacceptable since it did not allow an *in vitro* characterization of the SR, which would be necessary to compare with the ongoing animal experiments. In order to achieve this three-compartment model, it was necessary to explore alternate paths unrelated to the physiologic issues. These are 1) a faster computer or platform to run the program; 2) varying the step size in order to control the round-off error and run-time; 3) optimizing the model so that overall run times would be decreased. Of these paths, the second is the most reasonable. The first requires a capital investment or some other serendipity. The third has the potential for complicating and impairing the transferability of the project. It is often straightforward to use lumping of parameters to reduce the number of computations. This results in a model which is difficult and confusing to maintain and less likely to have general application.

Stout and Diecke have shown that the S.R. can be destroyed with Triton X-100 without impairing the contractile functionality of the smooth muscle cell. This yields a two-compartment model. Characterizing this two-compartment model and accurately reflecting the experimental conditions still involves number of assumptions. The following Figure 2 introduces some of these assumptions.

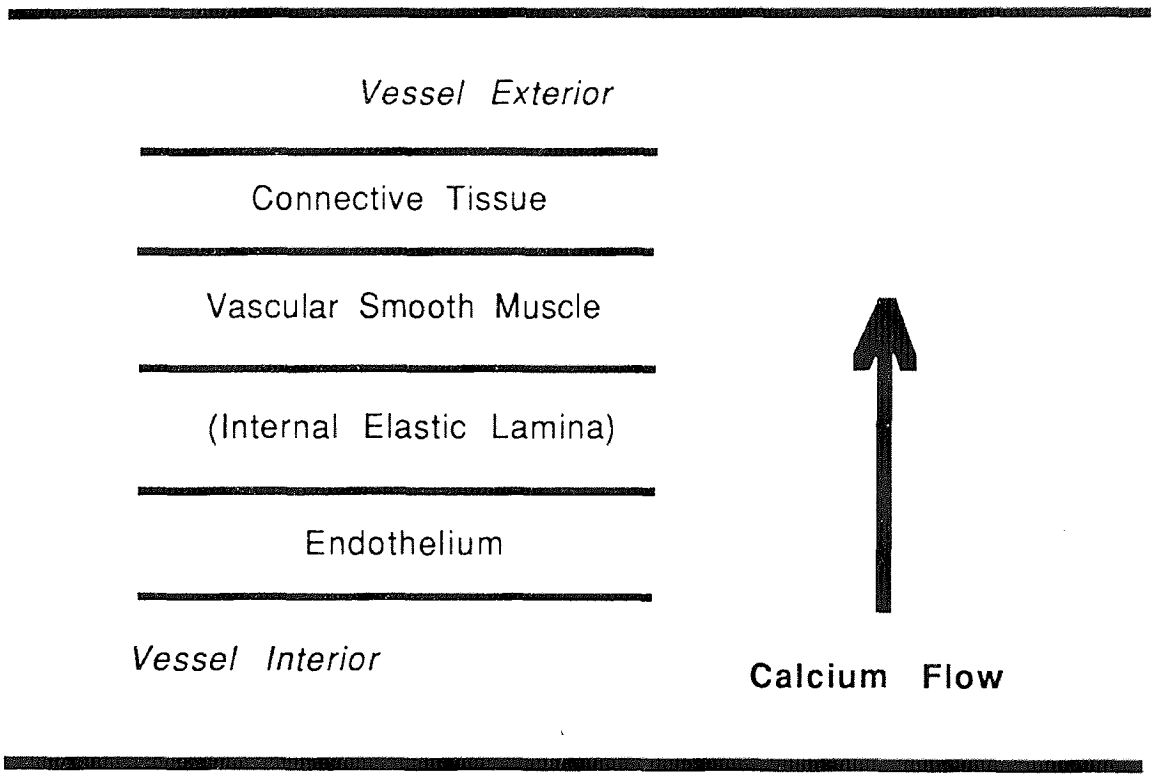


Figure 2 - Vessel Cross Section

This is a diagrammatic representation of the pharmacological model, as would be present *in vivo*. The experimental model (tissue preparation or

prep) uses helically cut strips of the vessel, so that both the inner and outer wall are exposed to the test solutions. The spindle-shaped smooth muscle cells are distributed helically about the vessel wall, so the prep results in a sample with the cells parallel to the long axis. The connective tissue and endothelium/internal elastic lamina are lumped together and form the first compartment or extracellular space. This compartment introduces considerable variability in the experimental prep and has been suggested to account for the difference between predicted and observed rate of diffusion (Stout).

Modeling the actual contributions of the SR to calcium release is in itself a major undertaking. The SR has a number of established diffusion and activation pathways as well as internal calcium storage and active transport mechanisms. The contribution of each of these physical components (cell architecture) during the release of calcium has yet to be established. The performance of some of these components, when isolated from their respective membrane structures, has been characterized chemically, although this information is not directly transferable to the physiologic model. That is, chemical characteristics obtained under non-physiologic conditions are not transferable to a physiologic model. The absence of verified parameters is a significant impediment to the modeling process.

A number of other investigators have attempted to study the SR by isolating it from the smooth muscle cell (Hasselbach, MacLannan, Miyamoto, Ostwald, Stewart). These studies have enumerated the components of the S.R. but have not illuminated the *in-vivo* operation of the aggregate components. They do provide insight into the functional relationships and capacities of the components.

Given the complex role that the SR plays in the regulation of intercellular calcium, it is not now possible to analytically characterize the whole expression of its function with one distinct model. The presence of three-compartments combined with the active transport and sequestering reaction (see Tanford) is not convenient to solve analytically (see Jacobs for a typical result). In addition, correlation of the model with experimental data is more complex because the *in-vitro* experiments are performed with a radioactively labeled species of calcium, designated $^{45}Ca$, with the additional presence of EGTA, which acts as a calcium buffer. This increases both the number of variables to be tracked by the model, and the models' complexity. The addition of a buffer system adds a reaction phenomenon to the existing simple diffusion. The absence of verified parameters for the components of the SR will limit the functional representation to that of calcium release and ignore the sequestering reaction and capacities, as well as the active transport mechanism by which the SR removes calcium from the cytoplasm.

Until the development of experimental procedures to characterize the unknown parameters of the SR, it would be possible to obtain a likely range for the value of some physiologic parameter through a technique known as a parameter study or *sensitivity analysis*. In this method, the parameters for the model are varied to determine what type of effect they have on the output. This approach is used frequently with *discrete* models (queueing theory, etc.) in order to characterize the dependence of a model on some subset of parameters, if any, which in turn further refines the model (Law, Shannon). The drawback in this method, for *continuous* models (differential equations), is that hundreds to thousands of runs are required. This is not a problem for *discrete* models because they are inherently easier to evaluate (calculate), in comparison with *continuous* models. Unfortunately, the software and analysis tools initially were not amenable to such a high volume of simulation runs and subsequent analysis.

This approach, which I will term as a rational parameter analysis, will be considered a *design goal* for this project, with some cautions. With a large set of adjustable parameters it is of course possible to fit any model to a given set of data, and this rational method has yet to be implemented and validated. In general, arbitrary refinement to improve the fit between model and data is misleading and should be avoided (Cooney). A *rational* refinement seeks only to establish the domain of the parameter so that appropriate experiments might be designed to validate the estimate. The motivation (design goal) is to reduce the duration of a simulation run, and to automate the process of generating runs, and thus take an informed step toward an integrated facility which would accomplish a rational parameter study.

A final consideration was to optimize the modeling environment so that model changes were easier for a non-programming individual to effect. This is in part to satisfy an ethical desire for process efficiency but more so to justify the additional effort which would make the modeling process a more accessible and useful research tool.

# Material and Methods

This section will focus on the details of implementation necessary to facilitate exploration of a three compartment model for calcium release. It details the overall areas of Model Building, Simulation and Analysis as discussed in the Introduction, with particular emphasis for the problem at hand. The following sections on Model Equations and Development of Parameters constitute the Model Building part, and are a direct extension of the two compartment model by Diecke. Any interest in the rationale for compartmental modeling of Vascular Smooth Muscle should be directed to Diecke.

The addition of the Sarcoplasmic Reticulum introduces a number of additional parameters, few of which have been determined experimentally. The estimation of these unknown parameters can be constrained effectively with a parameter study or sensitivity analysis. In a *sensitivity analysis,* the initial estimate of some parameter is incremented slightly and the whole of the model is simulated with these new values. The effect of the new value is assessed and the overall effect of the parameter is expressed in terms of the current model, which yields the model's sensitivity to that particular parameter. In the course of the study, parameters which are insensitive can be discarded or lumped in with other parameters. Although a precise value for the parameter cannot be obtained through such methods, a good estimate of the likely range of values for a given parameter can be established (see Shannon).

While this approach is often applied to discrete models, which are easy to calculate, the method is used less frequently with continuous (differential) models, which require significantly more time to calculate. The calculation times are significant when, depending on the number of unknown parameters, upwards of 30,000 runs are required for statistical significance (Shannon). Since the two compartment model by Diecke took upwards of a day to calculate (PC-XT computer), the prognosis of obtaining any level of sensitivity analysis with a three compartment is poor. In fact the three compartment model took almost 10 days (PC-XT). Clearly, if there was to be any hope of systematically determining parameters for the Sarcoplasmic Reticulum, it would be necessary to strongly optimize both the simulation (calculation) of the model, as well as supporting the analysis and reduction of large amounts of data.

Thus the bulk of the methods used in this paper have to do with these issues. After some review of the available software tools, the variety of tools developed for each of the target compute platforms, and analysis scenarios, are introduced. Much of the focus on the development and use of the software tools are on the automation of the processes in which they are applied and this reiterates the theme of simulation as a process.

## Model Equations

The development of a model for calcium flow begins with the translation of the physical reality into a mathematical representation. The particular methodology is called *compartmental modeling* and this method basically assigns the physical components that have significant volume a unique compartment in the model. Next is the consideration of the components which occupy that volume in each of the compartments. This is conveniently handled with a mass-balance relation which accounts for the inputs and outputs of the compartment. A mass-balance relation is composed of variables, which describe the amounts of a component in a compartment, and parameters, or rate constants, which describe how fast a component can change in value depending on the relationship between compartments. The final set of equations, and their relationships, is then considered a model of the physical reality. (For more introduction to the development of compartmental models, see Randell.)

The physical properties of the SR dictate the form of experiments which elucidate its function. Since it is difficult to work on the SR directly (see Bond), it is through these experiments that the role of the SR is demonstrated. The value of any model is in its ability to emulate a variety of experimental conditions, so some attention will be made as to the form of experiments that yield some appreciation of the SR's role and capacities.

The addition of the Sarcoplasmic Reticulum (SR) into a compartmental model offers a number of unique challenges, as the following Figure will reveal.

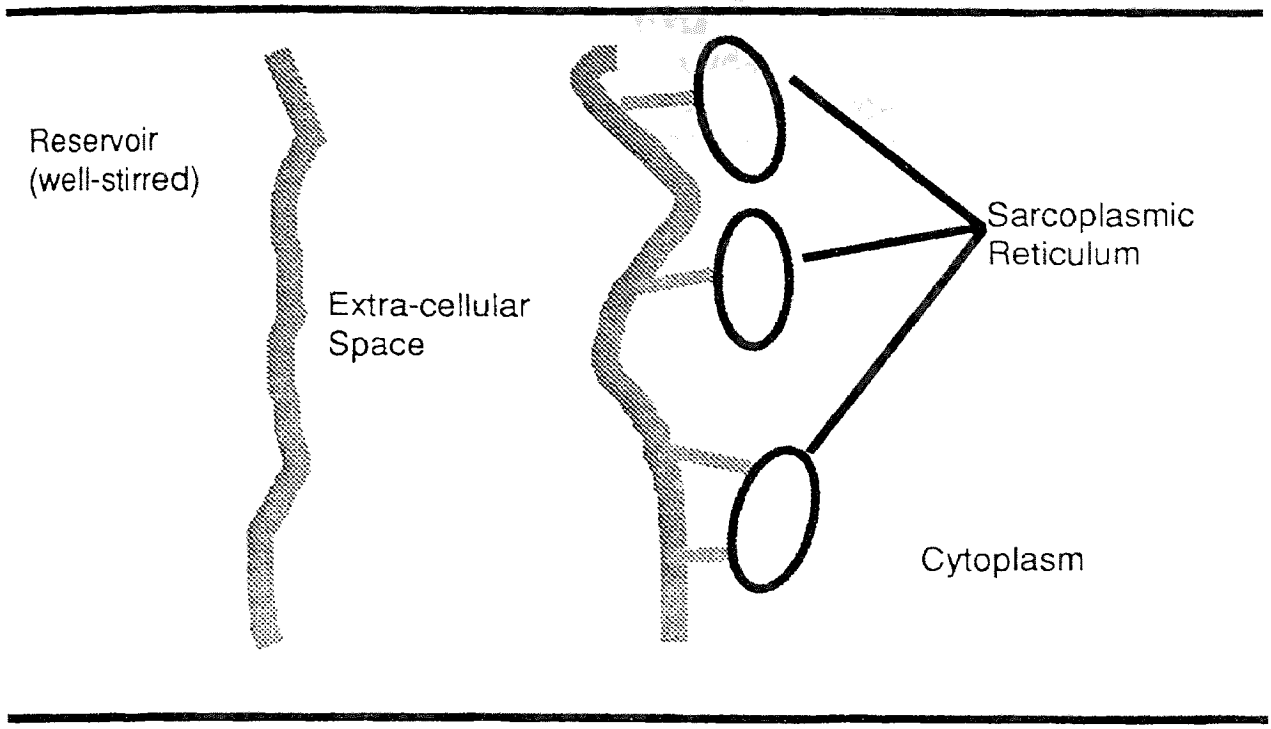


Figure 3 - Tissue Model

Figure 3 is adapted from various electron micrographs (see Allen; Bond; Jones). The two primary compartments (as modeled by Diecke) are the Extra-cellular space (EC) and the Cytoplasm (CY). As mentioned in the Introduction, this extra-cellular space is comprised largely of connective tissue. Together, these represent 56% and 28% of the VSM model volume. The SR, as the proposed third compartment, contributes 1.6% of the total cell volume. This is motivated from a range of values between 2% to 5% in different tissue samples (Stout). These volume differences have significant effects on the apparent and theoretical rate constants; the method of correcting these values will be presented later.

The shape, volume and proximity to the membrane of the SR gives it some interesting properties. The most significant of these is that the SR acts both as a buffer to incoming calcium (EC to CY) and as a third compartment in series with the other two (Hurwitz; van Breemen). Calcium can flow directly from the EC to the SR, and from the EC -> CY -> SR. Calcium is required to effect the contraction of the muscle fibers; it has the property to act both as an agent and as a messenger in various physiological systems. By regulating the uptake and release of calcium the SR regulates the contractile process. The measurement of tension (force development) is the fundamental metric for the performance of a smooth muscle preparation (Diecke; Dohi; Endo; Hurwitz; Jones; Somlyo).

In addition to the SR's role as a compartment in the diffusion of calcium, the SR is also the main store of calcium in the smooth muscle cell (Bond; Somlyo; van Breemen). The SR is able to store, or sequester, a large amount of calcium which is used for the maintenance of contraction. A second group of experiments (see Dohi; Hurwitz ) use the loading and discharging of calcium to measure a quantity known as the calcium-releasable store. The time course and magnitude of the uptake and release are, at present, the only indicators of the corresponding mechanisms contained in the SR.

The SR is a specific membrane analogous to a mitochondria or endoplasmic reticulum. It has the capacity to move calcium against a concentration gradient by way of an ATP activated translocation, or pump. This feature is the basis for a number of methods which assess the contribution of the SR to calcium flow by controlling the availability of ATP (see Hurwitz; van Breemen). Attached to the interior of the SR membrane are a variety of calcium specific binding proteins, one of which is called calsequestrin. A single binding protein is capable of binding 20 to 40 calcium ions, depending on the type of protein. The equilibrium for the binding proteins is at a high concentration in the SR, relative to the cytoplasm. A small decrease in concentration can effect a large release of calcium; that mechanism is presently unknown.

For the initial model, this suggests that the calcium concentration in the SR should be significantly higher than that of the cytoplasm, or third compartment. It also suggests that in order to estimate the diffusion parameters for the SR, it would be important to avoid sudden decreases in calcium concentration so that the sequestering function is not activated, causing a rapid release of calcium. This is achieved experimentally by effecting calcium release without contraction, or by not activating the contractile machinery of the cell.

### Effect of Volume on Rate Constants

While the actual rates of passive exchange between compartments are identical ($k_{12}$ and $k_{21}$ ), the apparent rates can be different when the compartments are not equal in size. For example, consider two compartments at equal concentration C. Where one compartment is two times the size of the other (the size of a compartment is its volume **V**), the larger compartment is caused to lose 10% of its material into the smaller compartment. Assuming that the flux J will be equal between the two compartments, then

$$(1) \quad J_{1\to 2} = k_{12}\frac{V_1}{A_1}C_1 \qquad \text{and} \qquad J_{2\to 1} = k_{21}\frac{V_2}{A_2}C_2$$

Note that $A_1$ (area) is always equal to $A_2$ when the separating distance, or membrane thickness, is small when compared to the area, so that $A_1$ and $A_2$ can be replaced by $A_{12}$, indicating the connecting area between compartments 1 and 2.

When $C_1 = C_2 = C$ (equal concentration), and $J_{12} = J_{21}$ (equal flux), the following holds:

$$(2) \quad k_{21} = k_{12} \frac{V_1}{V_2}$$

This is the adjustment to be made when the compartment volumes are dissimilar. While $k_{12} = k_{21}$, presumably, the apparent rate constant is modified by the volume ratio, and the following is found:

$$(3) \quad k_{21} = 2\,k_{12} \quad \text{using} \quad \begin{matrix} V_1 = 2 \\ V_2 = 1 \end{matrix} \quad \text{or } V_2 = 0.5\,V_1$$

This is how the volume difference between compartments affects the perceived rate constants.

In order to examine the effects of isotope exchange, it is necessary to achieve an equilibrium condition. Since the reaction of EGTA with Calcium is also accompanied with the exchange of these components between different compartments, the equilibrium values can be obtained through simulation, using initial values derived from the following method, until steady-state is achieved. An alternate method which solves explicity for equilibrium values follows the simulation method.

The estimate is obtained by first noting that a reversible 1st order reaction exists, which has the following form:

$$(1) \quad Ca + EGTA \rightleftarrows Ca\text{-}EGTA$$

Using $k_f = 2x10^6$ and $k_r = 0.4$ , the equilibrium constant for this reaction is:

$$(2) \qquad K_{eq} = \frac{[\,Ca\text{-}EGTA\,]_{\infty}}{[\,Ca\,]_{\infty}\,[\,EGTA\,]_{\infty}} = \frac{k_{foward}}{k_{reverse}} = 5.0 \times 10^{6}$$

Since we want to determine what the concentrations of bound and free EGTA will be, for a given total EGTA ( $EGTA_{total}$ ) such that ,

$$(3) \qquad [\,Ca\text{-}EGTA\,]_{\infty} + [\,EGTA\,]_{\infty} = EGTA_{total}$$

the following manipulations will be performed.

Rearranging the equilibrium expression yields (4), isolating the free EGTA ( [EGTA] ).

$$(4) \qquad [\,Ca\text{-}EGTA\,] = EGTA_{total} - [\,EGTA\,]$$

With (2), the bound concentration ( [Ca-EGTA] ) is isolated as (5)

$$(5) \qquad [\,EGTA\,] = \frac{[\,Ca\text{-}EGTA\,]}{[\,Ca\,] \bullet k_{eq}}$$

Substituting (4) into (5), to eliminate **[Ca-EGTA]**, yields (6)

$$(6) \qquad [\,EGTA\,] = \frac{EGTA_{total} - [\,EGTA\,]}{[\,Ca\,] \bullet k_{eq}}$$

Isolating **[EGTA]** yields (7), in terms of the total EGTA ( $\mathbf{EGTA_{tot}}$ ), the free Calcium ( **[Ca]** ), and the equilibrium constant ( $\mathbf{K_{eq}}$ ).

$$(7) \qquad [\,EGTA\,] = \frac{EGTA_{total}}{([\,Ca\,] \bullet k_{eq}) + 1}$$

The calculated result of (7) is substituted into (4) to determine the value for [Ca-EGTA].

With equation (7), the approximate equilibrium values are determined. No adjustment for compartment sizes has been made at this stage. These adjustments are made to the forcing function, which is given by equation (8),

$$(8) \quad FF = (k_{i_{01}})\,[C_{i_{res}}]$$

where **i** represents the species ( Ca, EGTA, Ca-EGTA ). The rate constant $\mathbf{k_{01}}$ is the diffusion constant between the reservoir and compartment one.

For the purposes of this study, the simulation is considered to be at steady-state when the change in total calcium is less than three significant figures. This amounts to a change on the order of $10^{-10}$ in Calcium concentration. It is again noted that at this point in the simulation there is still a significant exchange between compartments, and with the EGTA buffering reaction. This can be characterized as a bounded oscillation, since overall, the total calcium does not change.

For this system of Ca, EGTA and Ca-EGTA it is also possible to solve explicitly for each equilibrium value by using relations for the total EGTA ($\mathbf{EGTA_T}$) and total Ca ($\mathbf{Ca_T}$).

By noting that for the free concentrations of Ca and EGTA

$$(9) \quad Ca_{free} = Ca_T - \text{Ca-EGTA}$$

$$(10) \quad EGTA_{free} = EGTA_T - \text{Ca-EGTA}$$

For the EGTA binding reaction

$$(11) \quad \text{Ca-EGTA} = \frac{EGTA_{free} \bullet Ca_{free}}{k_D} \qquad \text{where} \quad k_D = \frac{1}{k_{equilibrium}} = 2 \times 10^{-7}$$

Substituting equations (9) and (10) into (11) and collecting like terms yields (12)

$$(12) \quad \text{Ca-EGTA} = \frac{(\text{Ca-EGTA})^2 - \text{Ca-EGTA} \bullet \left(Ca_T + EGTA_T\right) + EGTA_T \bullet Ca_T}{k_D}$$

This quadratic equation is then soved by standard methods to yield values for the Ca-EGTA concentration, from the total Ca and total EGTA. This is then substituted back into equations (9) and (10) to solve for the free Ca and free EGTA concentrations. Equation (12) can be rearranged into a standard form as follows

$$(13) \quad 0 = (\text{Ca-EGTA})^2 - \text{Ca-EGTA} \bullet \left(Ca_T + EGTA_T + k_D\right) + EGTA_T \bullet Ca_T$$

## Software considerations

This section will introduce the procedures by which the different software elements are used. This begins with the model equations which are developed with a text editor according to the conventions for the particular programming language. This **Model_file** is then compiled and linked into the rest of the simulation engine. This is carried out via the system compiler and linker, or through a **Makefile**. Any computer operating system has some facility to carry this out, though the details will vary somewhat. The parameter file for the model is also created with a text editor. This method is superior to interactive input of the various run parameters and permits successive parameter files to be adapted from existing parameter files. The organization of the parameter file is straightforward as it is designed to be generated by a future application program which would automate the parameter file creation. A sample parameter file can be found in the Appendix.

A group of parameter files is submitted for processing by the SimEngine as a Runlist. This is also a text file and contains the pathnames of the individual parameter files. Earlier versions of the SimEngine (DIFSIM) used enternal text files to control execution of the program. This was useful for single-user, single-task systems (such as a PC) and allowed the program to be halted or re-started without losing any data. This feature is not in the current implementation because the runs are much more reliable and, when necessary, more sophisticated control schemes are available.

One other file, SIMSTATUS, is generated by the running SimEngine and contains a copy of the last set of data that was written to the file. Depending on the version of the software, this will also report how many simulations remain to be processed in the current Runlist.

The SimEngine program is executed as a background process, on multi-user systems, and the form of this command will vary a little depending on the particular system.

Once the SimEngine program has completed a parameter set and closed the file, post-processing can begin. It is more convenient to wait until all of the parameter sets are completed because the SPAWN program can re-use the Runlist, saving the trouble of typing each of the filenames to be processed. This may not be an advantage if each of the runs takes a few hours, and results are of the essence. The final choice is up to the user.

Some thought should be given to the number of data points required for analysis. In order to get smooth curves for plotting, it is necessary to generate a lot of points. This does result in very large data files which will be difficult to transfer to other systems. In other situations, such as a half-time study, it may only be necessary to generate the minimum number of points to get a good curve fit. This is very desirable when the curve fitting program is on a PC, copy protected, and not part of the computing network (but part of a sneaker network). Certainly other utilities can be developed to more closely fit the demands of a particular simulation environment.

The initial run is often not the actual experimental run. For this study, starting values for an experiment are first estimated, using the expressions for EGTA estimation previously developed, and then run until equilibrium is achieved. Equilibrium is defined as the point at which the Total Calcium is constant for at least 1200 simulated seconds. The duration for the equilibrium run is a function of the EGTA concentration, while the estimating function is a linear equation.

Once the proper initial values have been established, the run variants can be produced. For the Calcium Release experiments, this involves a reference run with $^{45}Ca$ washout conditions for 2400 seconds. The variant runs, which modify the value of calcium diffusion from the third compartment, are initialized with values taken from the reference run at 1200 seconds and will run for 1200 seconds. In this fashion, the simulation can track a proposed transient event (calcium release) without modifying the original model. With the additional file handling it is possible to introduce simple errors, particularly when working with 12 digit numbers, and additional work will be needed to make this part of the simulation environment more reliable.

After a number of runs have been completed and analyzed, it would be prudent to review performance and precision expectations. At the beginning of this project, this meant a hiatus from modeling because the run times were very poor and much development was needed. The result is the Adaptive Step-control algorithm. For a future investigator, this juncture should now only require some adjustment in the simulation configuration parameters, in order to meet expectations.

# Application

## Simulation

This section will cover two areas. The first half will detail enhancements to the integration method and an evaluation of its effectiveness. This will include discussions of Runge-Kutta and Step Change algorithms. The Step Change Algorithm is a unique implementation and comprises the main contribution of this project.

Before embarking on this discussion, it would be prudent to review the rationale for a numerical solution to a physiological model. Specifically, how is it that a particular algorithm has broad applicability in evaluation systems of this type. The system can be considered as modeling by compartmental analysis. This type of analysis uses a simple mass balance approach to develop linear differential equations representing how the mass of each species changes with time. Numerical integration is the method through which these equations are solved.

The general case under which compartmental analysis falls is an *initial value problem*. This means that we have a set of answers initially and we want to know what the answers will be at some other final point in time (simulated time). Solving the compartment model also suggests that *a numerical solution to a system of ordinary differential equations* is desired. It is fortunate that the diffusion processes that are to be investigated are all first order differential equations, for which numerical methods are well understood. Of course second order, or higher, equations can also be reduced to a combination of first order equations and a number of new variables. Those are , however, concerns for more intractable problems.

The basic approach to solving any system of differential equations (DE's) is to rewrite the equations in terms of finite steps, as shown below.

(1) $$\frac{dy}{dx} = z(x)$$ Differential Form

$$\frac{\Delta y}{\Delta x} = z(x)$$ Finite Step Form

(2) $$\Delta y = z(x)\,\Delta x$$ Numerical Approximation

The Differential Form is a continuous representation of how the function **z(x)** changes when the difference between two values is infinitely small. Since this approach is available only in the calculus, and not available practically, it is necessary to make an approximation in terms of a finite difference, as indicated by the **D** operator. Rearranging the Finite Step Form yields the expression which is a solution via Numerical Approximation. The product of the function **z(x)** with **Dx** yields an increment in **Dy**. Thus the step size is given by **Dx**, and in the limit of making this step size very small, a good approximation to the underlying differential equation is achieved.

For completeness, an exact representation of this technique would be given by Euler's method:

$$(3) \qquad y_{n+1} = y_n + hf'(x_n, y_n)$$

which is the basis for other numerical integration methods. Equation (3) suggests that the next value of $y_{n+1}$ can be obtained by incrementing the previous value $y_n$ by the product of the step size **h** and the derivative of the function. This is identical in purpose to the previous Equation (2) by noting the following:

$$(4) \qquad \Delta y = y_{n+1} - y_n$$

$$(5) \qquad h = \Delta x$$

$$(6) \qquad f'(x_n, y_n) = \frac{dy}{dx}(f) = z(x)$$

By rearranging (4) and substituting according to (5) and (6), equations (3) and (2) are shown to be identical, except for the differences in notation.

These methods are generally applicable to any system of first order differential equations. However, the essential limitation of these numerical approximation methods is a question of computing efficiency, which is a strong function of the step size **h**. The smaller the step size becomes, the more accurate the solution, but at the expense of additional calculations. The step size is also not an arbitrary value and is instead related to the differential equations at hand. Since it is possible to evaluate these equations at incorrect step sizes (as will be shown later) it is imperative to develop an early understanding of what a particular system will require in terms of step size.

There are a number of algorithms available which in some cases offer enhanced efficiency while compromising any generality of application. It is a more significant goal to maintain the broadest possible scope by using a

reliable, predictable algorithm which will allow us to more easily explore the relationship of step size to the accuracy of the simulation.

The method of solution for the system of differential equations will be the Runge-Kutta-Gill integration algorithm. This is a general purpose method, having the advantage over other methods in that it is self-starting (it does not require another algorithm to produce the initial values necessary to begin approximating the solution of the model equations). One way to evaluate the suitability of a general purpose algorithm is to look at the circumstances under which it will break down, or become unsuitable for the problem at hand. Until this is encountered, no other algorithms need be considered for the purposes of this paper.

**Runge-Kutta Integration**

The details of the Runge-Kutta algorithm and its implementation in code will be introduced as a prelude to the discussion of the Step Change Algorithm. Although the algorithm is widely known and referenced, it will be detailed here for the sake of completeness and also to serve as a platform for introducing some of the parameters that are required for the Step Change Algorithm. It will certainly not be the last attempt to discuss numerical integration in more tractable terms. Of necessity, this discussion builds on the content of the previous discussion.

From Press, the mechanism of a Runge-Kutta method is to

> ...propagate a solution over an interval by combining the information from several Euler-style steps, and then using the information obtained to match a Taylor series expansion up to some higher order.

This is concise, and perhaps sufficient, though the Runge-Kutta algorithm is better described in geometric terms. Essentially, it breaks a single integration interval **h**, or step size, into three trial calculations and a final calculation. These estimates are made at the starting point $(x_n, y_n)$, at the midpoint $(x_n + h/2, y_{n+s})$, and at the end point $(x_n + h, y_{n+1})$. The subscript s is to reflect that a number of evaluations are made at the midpoint. It begins by calculating an initial slope $k_1$ at the starting point, and uses this value to calculate a slope $k_2$, at the midpoint. This value for $k_2$ is used to re-evaluate the initial estimate, producing a new slope $k_3$ at the midpoint. The result at this stage is used to calculate the slope $k_4$ at the endpoint. Each of the four slopes are averaged with weights, and with this last evaluation, a full step from $(x_n, y_n)$ to $(x_{n+1}, y_{n+1})$ is made. The four slope calculations can be represented as follows:

$$k_1 = h\dot{f}\,(x_n, y_n)$$

$$k_2 = h\dot{f}\,(x_n + \frac{h}{2}, y_n + \frac{k_1}{2})$$

$$k_3 = h\dot{f}\,(x_n + \frac{h}{2}, y_n + \frac{k_2}{2})$$

$$k_4 = h\dot{f}\,(x_n + h, y_n + k_3)$$

As a 4th order Runge-Kutta method, this mechanism can be more simply stated by noting that at each step, the derivative is evaluated four times: once at the initial point, twice at trial midpoints, and once at a trial endpoint. From these trial derivatives the final function value is calculated, as shown below.

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

Since we will be working with a number of equations in our models, it will be necessary to perform these calculations repetitive and to group the operations efficiently. A sample segment in FORTRAN, which calculates the slopes $\mathbf{k_1}$ and $\mathbf{k_2}$, is excerpted below:

```
1.          CALL MODEL_TR(X, Y, AA, BB)
2.          DO 21 i = 1, NUMB
3.          AK1(i) = Y(i)
4.          T = T + DT/2.0D0
5.          DO 22 i = 1, NUMB
6.          x(i) = Z(i) + DT * AK1(i)/2.0D0
7.      22  CONTINUE
8.          CALL MODEL_TR(X, Y, AA, BB)
9.          DO 23 i = i, NUMB
10.         AK2(i) = Y(i)
11.     23  CONTINUE
```

Figure 4 - Sample FORTRAN Runge-Kutta Integration

First the function is evaluated at the starting point. This effectively yields the slope at that point. Next a loop is set up, over all of the equations present and the result of the function evaluation is assigned to **AK1**. This is in accordance with the algorithm outlined at the beginning of the discussion. The subscript **(i)** suggests that we are indexing into an array of values. In this fashion, it is possible to group similar operations together and thus apply the algorithm stepwise over all of the equations, instead of computing each equation to completion. The increment in the time **T**, in line 4., is just for reference and does not enter into the calculations since they are dependent on the time step **DT** (or **h** as before) alone. A loop is set up again and, in line 6., the initial values of **x** are increment by a half step. These new x parameters are then passed to the **MODEL_TR** subroutine for the function evaluation. The procedure for the next, and successive slopes, is similar. A complete implementation in C can be found in the Appendix.

The essence of what should be understood here is that we have characterized and reduced the model equations to a set of slopes. This slope information is a more meaningful measure, or metric. For example, the direction of an automobile can be determined by the movement of the steering wheel but the direction at any time can be more easily obtained by using a compass. The compass is a better metric than the smaller, more frequent perturbations of the steering wheel. In a similar fashion, the equation slopes can produce a metric of what is generally going on with the model equations. Over these metrics shall be a policy and it is at the level of policy that more abstract quantities such as performance can be readily evaluated.

### Adaptive Step Change Algorithm

The algorithm for adaptively changing the step size used during the integration interval is found in Carnahan. It develops an expression for the error over the estimates for the slope, which, after exceeding some predefined limit, can be interpreted to mean that a change in step should be made. The expression, which will be referred to as the **ErrorRatio**, is as follows:

$$\text{ErrorRatio} = \left| \frac{(k_3 - k_2)}{(k_2 - k_1)} \right|$$

It is intended that this be calculated after each integration step. If the ratio becomes large then the step size should be decreased. The suggested criteria for this is on the order of a few hundredths and can be considered extremely qualitative. The current criteria, which is established later on, is to keep the ratio between $10^0$ and $10^{-15}$. This domain was established because at steady-state the denominator frequently evaluates to zero. In other situations, the **ErrorRatio** evaluates to unity, when the slope estimates are particularly good.

Though the additional computation to evaluate this ratio is small, it seems desirable to limit the use of the slope monitoring procedure to something less frequent than every step. This introduces another parameter, **CheckFrequency**, which can also be considered to be application-dependent.

This method is further extended so that the slope expression can be used to suggest when the step size should be increased. This of course is also qualitative and therefore application dependent, requiring that parameters be selected for both upper and lower bounds on the error. These will be called **ErrorUpLimit** and **ErrorLowLimit**, respectively. If we consider under what criteria the step is to be changed under the aforementioned constraints, the following policy is developed:

StepRequest {
- DECREASE <-- Slope >= ErrorUpLimit
- PASS <-- Slope < ErrorUpLimit
- PASS <-- Slope > ErrorLowLimit
- INCREASE <-- Slope <= ErrorLowLimit

Figure 5 - StepRequest Fundamentals

The meaning of **INCREASE** and **DECREASE** is obvious. The **PASS** is intended to suggest that no change in step is required. The value of **StepRequest** would be passed to another procedure which would accomplish the actual change in step. This is done so as to permit the exploration of various policies on changing the **StepSize**. The best policy to date is a buffering of the **StepRequests** so that any changes in the **StepSize** are more strategic. This policy can be depicted as follows:

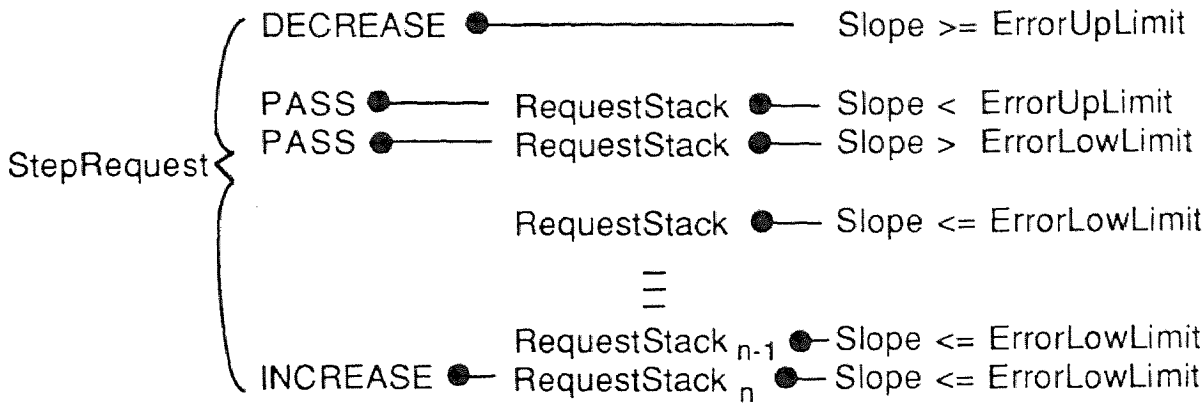


Figure 6 - StepRequest Strategy

When the slope is greater than or equal to the **ErrorUpLimit**, this is cause for an immediate **DECREASE**. When a slope is between the **ErrorUpLimit** and the **ErrorLowLimit**, a **PASS** is placed on the **RequestStack**. Since the overall effect is to send a **PASS** along the chain to **StepRequest**, it is depicted as such in Figure 7. It is important that **PASS** is placed on the **RequestStack** since this could be part of some cyclic behavior as far as the number of **StepRequests** that would be processed. For those slopes <= **ErrorLowLimit**, the path to **StepRequest** is buffered by the **RequestStack**.

In order for a request for an **INCREASE** to get through **StepRequest**, it is necessary that the entire **RequestStack** contain a **DECREASE** at each one of its levels. In this way, any increase in step size must be ongoing for some period of time, while any oscillation between **PASS** and **INCREASE**, or between **DECREASE** and **INCREASE**, will be effectively ignored. There are two parameters that affect the success of this procedure. They are the size of the stack (**StackSize**) and the frequency at which the slopes are examined (**NextTest**).

Once the decision (by the main program, via **CheckFrequency**) to attempt a step change is achieved, there is a little more work to do. This will involve various tests to assess what the impact on the step size should be, and will introduce the routines that manage the **RequestStack**: **PUSH_STACK** and **TEST_STACK**. The essentials of this process are detailed in the next figure:

```
evaluate the ErrorRatio for each equation and keep the largest value

If LARGEST_ERROR > ErrorUpLimit Then
    Call PushStack( DECREASE )
    Else If LARGEST_ERROR < ErrorLowLimit Then

        Calculate new step size
        If the NEW_STEP_SIZE is valid Then
            Call PushStack( INCREASE )

            If TestStack( INCREASE ) is TRUE Then
                Change the StepSize
            Else PASS

        Else PASS

Else PASS
```

Figure 7 - Pseudo Code for Step Change

The first step is to evaluate the **ErrorRatio** for each of the equations present in the model and collect the **LARGEST_ERROR**. If there were any interest in determining which equation was the problem, then that code would be implemented here. As mentioned previously, additional code within this subroutine is not likely to impact performance.

With the **LARGEST_ERROR** in hand, it is then tested against the bounds for the error. If a **DECREASE** is called for, that action is placed onto the **RequestStack**. What is not explicit in Figure 7 is that the new decreased time step has been calculated and tested for bounds within the smallest allowable step size. This will be illustrated with the **INCREASE** portion, and is omitted for the sake of keeping the Figure uncluttered. If no **DECREASE** is possible, meaning that the minimum value for the step size had been achieved, no further action will be taken. The call to **PushStack** is still necessary so that the **RequestStack** is invalidated for any **INCREASE** until at least **StackSize** (or **n**) number of **INCREASE**s have been made. This ensures that there will be no cyclic **INCREAS**ing and **DECREAS**ing about the lower limit for the step size.

If no **DECREASE** is necessary, the **LARGEST_ERROR** is checked against the ErrorLowLimit to see if an **INCREASE** is possible. First the new step size is calculated and checked against the bounds for the step size. If this is valid, then the **INCREASE** request is pushed onto the **RequestStack** via the **PushStack** subroutine. Before the new step can be implemented, the contents of the **RequestStack** must be evaluated to determine that a request to **INCREASE** has been suggested for **n** or **StackSize** times. This function is

performed by **TestStack**, which will return a value of **TRUE** if the evaluation is successful. When **TRUE**, the new step will be passed back to the simulation engine.

In all other cases, the result will be to **PASS**, as suggested at the bottom of each of the decision points. There are of course numerous opportunities throughout this algorithm to collect meaningful metrics about the frequency of **DECREASES**, **PASSES**, **INCREASES**, the **ErrorRatio**, and the actual step size.

In order to successfully implement this algorithm, a number of other considerations (such as how to establish error criteria, when to increase or decrease, and how to evaluate the performance of various strategies) need to be explored.

## Model Changes

Ideally, all of the variable components of a model are represented in the parameters for the model. In this fashion, the changes to the actual equations of a model are infrequent, and the focus is instead on managing the parameters used in the simulation. With the use of the Adaptive Step Algorithm, a second level of changes might arise in the course of simulation and analysis: those parameters associated with the configuration. Figure 8 summarizes the scope and frequency of the changes that should be expected.

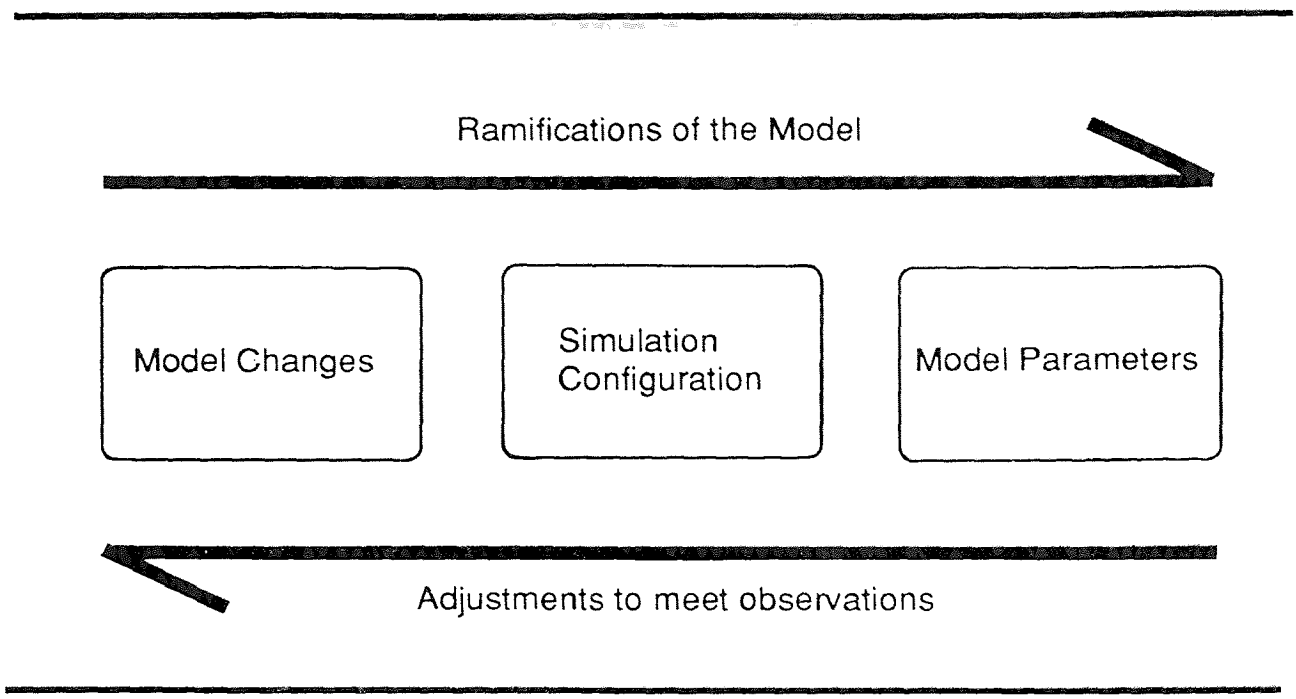


Figure 8 - Hierarchy of Model Changes

The initial decision on the Model has ramifications that extend to the parameters used in the model. When adjustments are desired in order to emulate additional experiments, these changes should be confined to the parameter file. If performance monitors should indicate that the calculations are inefficient or encountering problems, changes should be made in the adaptive parameters, while keeping the model parameters constant. Finally, when further extensions to the model are desired, the model equations themselves will be changed. Each of the preceding parameter files should then be re-run to verify that the new model substantially achieves the older model results.

**Translating the Model into DIFSIM**

In order to prepare the model for simulation the equations must be translated into FORTRAN using the variable and parameter notation described by Diecke. Recalling equations (1) and (2), the following code segments reflects the complete translation into the DIFSIM methodology. This example is for the verification model, which is detailed in the Discussion section.

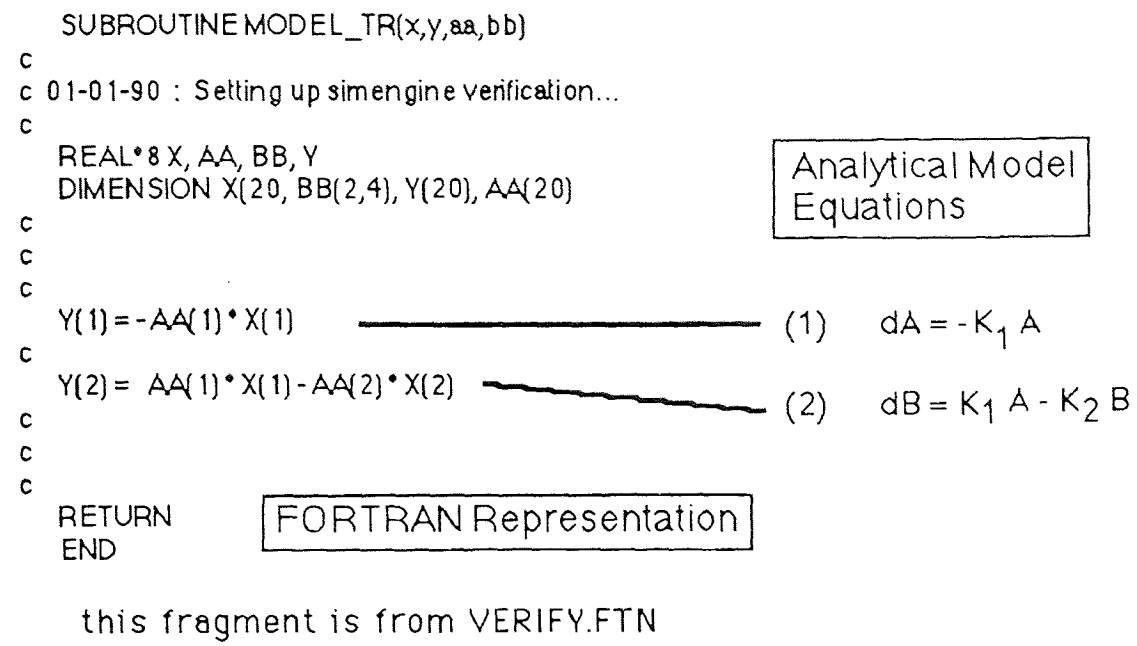


Figure 9 - FORTRAN Representation of Verification Model

The differential quantities are contained in the **Y** array, while the current value for each equation are contained in the **X** array. The equation coefficients, or parameters are kept in array **AA**. The model is kept as a separate subroutine and linked into the main program. This makes efficient use of disk storage space and also focuses any changes in the code onto one file. This is also convenient for documenting various models. The declaration of the variable types and the storage definitions (**REAL** and **DIMENSION**) are necessary because the subroutine is external to the main program. For anyone familiar with FORTRAN, the translation from analytical to FORTRAN equations is clearly straightforward.

It is also possible to substitute more descriptive labels for the equations and parameter coefficients, as is done with the C language (program SimEngine) version.

C Representation

```
/*  <C>oefficients */
#define K1          param_set.ec[ 0].value
#define K2          param_set.ec[ 1].value

/*  <E>quations  */
#define Compt_A       0
#define Compt_B       1

extern double x[ MaxEqns ];
extern double y[ MaxEqns ];
extern struct run_param param_set;

void Model_TR (void)
{
y[ Compt_A ] = - K1 * x[ Compt_A ];
y[ Compt_B ] =    K1 * x[ Compt_A ] - K2 * x[ Compt_B ];
}
```

Figure 10 - C Representation of Verification Model

This is a much more useful approach when the number of equations (and their relationships) are both large and complex. Details of the model for calcium release, in either programming language, can be found in the Appendix.

## Analysis

The development of tools to aid analysis is essential because management of and insight into the simulation are opposing policies. In management, the objective is to keep the output data "manageable" -- how much raw data is enough? For insight, it is necessary instead to determine the "right" parameter that characterizes the simulation. The difficulty is that it is not known initially what or where the "right" parameter is.

The emphasis on the idea of a tool, rather than a procedure, is that a tool has an obvious use and a procedure always seems open to debate and modification. In this sense, one would always know what a hammer will do but can still have plenty of debate as to when a hammer should be used in the course of a building project. At the beginning of this project, appropriate tools, and an environment in which to develop them, were not available. More convenient tools did exist on platforms other than the compute platform, although the level of integration (in this case the effective data transfer speed) was poor.

The following Figure 11 summarizes the procedure used to analyze the results of the simulation run and to generate the parameter files for additional runs. This procedure is convenient when the run duration is longer than one might be willing to wait for. It also allows for remote access via a PC, which is more accommodating for a graduate student's schedule and working habits.

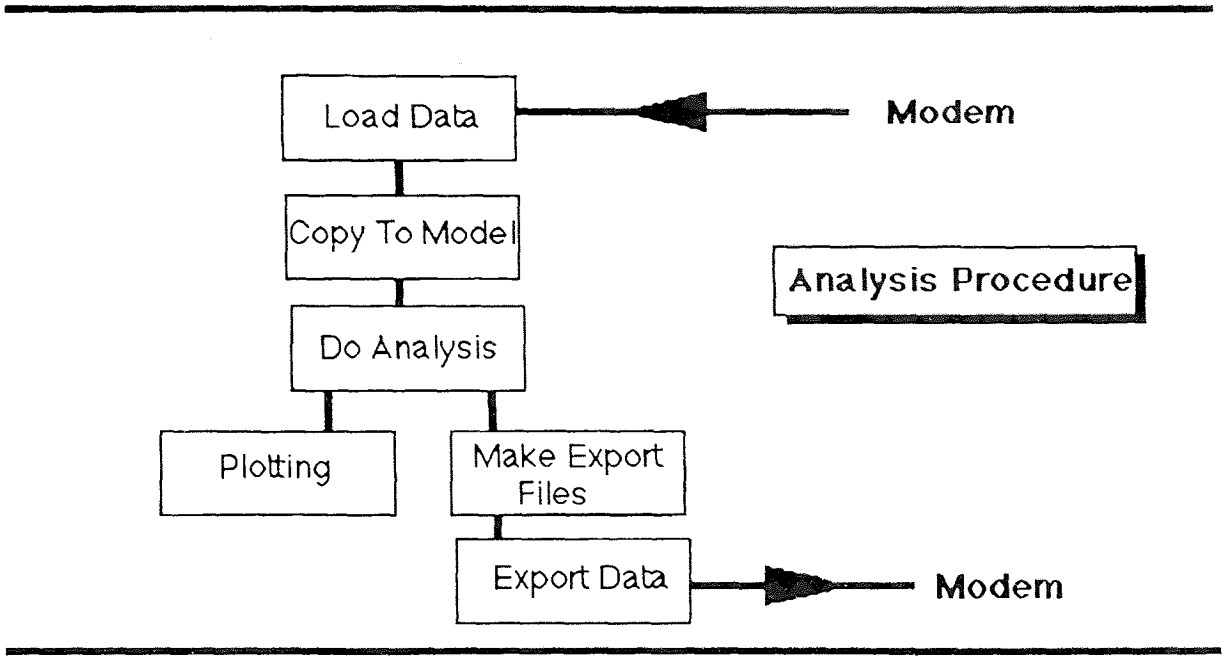


Figure 11 - Analysis Procedure

The ability to access a system via a modem or some other network is actually an example of a distributed processing environment where various compute platforms and facilities are optimized for a particular task. For example, a CRAY is very efficient at numerical calculations but very poor at display graphics; in fact it has no direct graphics facility. A Macintosh is handy for displaying graphics but is a terribly poor compute platform. More sophisticated modeling endeavors should actually expect to use a variety of compute and analysis resources, which would be available both locally and remotely.

**Analysis**

In any event, the conclusion of a simulation often results in a large file of data. At the initial phase of a modeling project, a good deal of flexibility is desired because it may not yet be apparent as to what the more interesting aspect of the results will be, or how best to analyze them. Toward the end of the simulation, when the analysis procedures are well known, the emphasis

will be on production -- the rapid and automated analysis of additional experiments. Towards this goal, a number of additional programs were developed in order to extract appropriate data from the simulation results and prep in for analysis on a remote platform.

The first of these is SPAWN, which takes the **.data** file and creates three other files: **.time**, **.plot**, and **.stat**. The **.time** file has only the simulation time and the time stamp for when that particular set of data was written out to the file. This file permits analysis of where the simulation is spending its time, as revealed in the interval between the data reports. For a constant step size run each of these intervals should be the same. Using a constant step run can reveal if the compute process is getting enough compute time. This is useful with multi-user and multi-tasking machine.

The **.plot** file is the primary form for utilizing the simulation data. The **.data** file contains a header and a summary of the parameter file on other run parameters and documentation. This is intended to put together all the useful information about the run, so that it may be reconstructed at a later date. The **.plot** file is the subset of the **.data** file without this header file.

The **.stat** file is a further subset of the **.plot** file in that it contains the Adaptive parameter values for each simulation time step. This is useful to track what the **SlopeError** is doing as well as track when increases and decreases in step are occurring during the simulation. Any of these can reveal additional transients in the simulation, as well as establish the onset and duration of steady-state, in terms of what the Adaptive algorithm needs to respond to. On the basis of these quantities, changes in the values of the simulation parameter would be made.

This program has been recently redeveloped for UNIX systems and is designed to be operated via command-line parameters and uses **stdin** and **stdout**. This makes the procedure more automated and precise, and the use of the standard I/O locations allows the command to pass information to another command (perhaps to do some analysis) without the use of intermediate files.

The common form for the results is to add the compartments which contain $^{45}Ca$ together. It is also useful to modify these values, prior to summation, by multiplying by the respective compartment volume. With this result, the data can be fitted and compared directly with the experimental results which have been analyzed in similar fashion. There is a program CRUNCH which does this calculation. The calculation is also done with a Spreadsheet program and has the added benefit of an integrated plotting facility through which to view the data. The **.plot** file is loaded into a spreadsheet template which has the appropriate formulas set up. The flexibility of the spreadsheet, tempered by the additional time it takes to transfer and load the file, is an

example of the initial phase analysis. A spreadsheet is not suitable for production analysis and manipulation but is amenable for seat-of-the-pants inquiry and graphical presentation.

### Determining the Rate of Release

The typical analysis of the total Calcium is the rate or arithmetic mean, given by the following expression:

$$\frac{(t_1 - t_2)}{\left(\frac{t_1 + t_2}{2}\right)} = \text{Arithmetic Mean}$$

This is evaluated within the CRUNCH program, the spreadsheet templates, and more recently with an AWK program **get_rate**. This last program is an example of the ongoing automation of the analysis procedures, where the results of the SPAWN program can be piped directly to the **get_rate** program, without the use of an intermediate file. The application of each of these programs results in a much smaller data file, lessening the time it takes to transfer the file for the final analysis and presentation.

### Exponential Curve Fitting

Curves for the Isotope exchange study were generated by fitting the results of the AM calculation to the following equation:

$$y = ae^{-(K_1 t)} + be^{-(K_2 t)}$$

The non-linear regression was calculated by Graph Pad and the parameters were fit to an error < 0.1%. The results of the curve fit yield parameters for the slope and time constant for each exponential term. The half-time is also reported and these values are present in the Discussion section.

# Discussion

The discussion of the results of the various methods delineates two distinct areas. The first section addresses the overall performance of the adaptive step change algorithm in terms of ease of use, relative increases in computational speed and the impact on the accuracy of the simulation. The second section introduces the experiments that both validate and explore the use of the simulation engine and associated analysis tools.

## Performance

### Run Times

A number of experiments have been collected that impart some interesting insights into the computational challenges of the current model. No one result was verified on all versions of software or hardware used in the course of this project and various results should not be directly compared. Given that the initial run times were on the order of days, and current run times are on the order of minutes, these magnitudes of difference make direct comparison impossible. However, the overall trend of the results will be the same for the successive pairings of hardware and software. Where possible, the results are stated in terms of constant and adaptive step, for the same combination of software and hardware. Any other combination of hardware and software is expected to have a linear relation to the initial combination.

### Stiffness of Model Equations

The largest single contributor to the computational complexity has been the *stiffness* of the model equations. A set of differential equations is considered stiff when the components of the equations differ by a few orders of magnitude. This effect is not apparent when a constant step size is used because the relative error is not considered. The accuracy does suffer and this is illustrated when runs with successively smaller step sizes are compared. The following experiment utilized the same parameter and adaptive step change values, varying only the EGTA concentration. Each of the runs began at equilibrium values for the particular EGTA concentration and was washing out $^{45}Ca$.

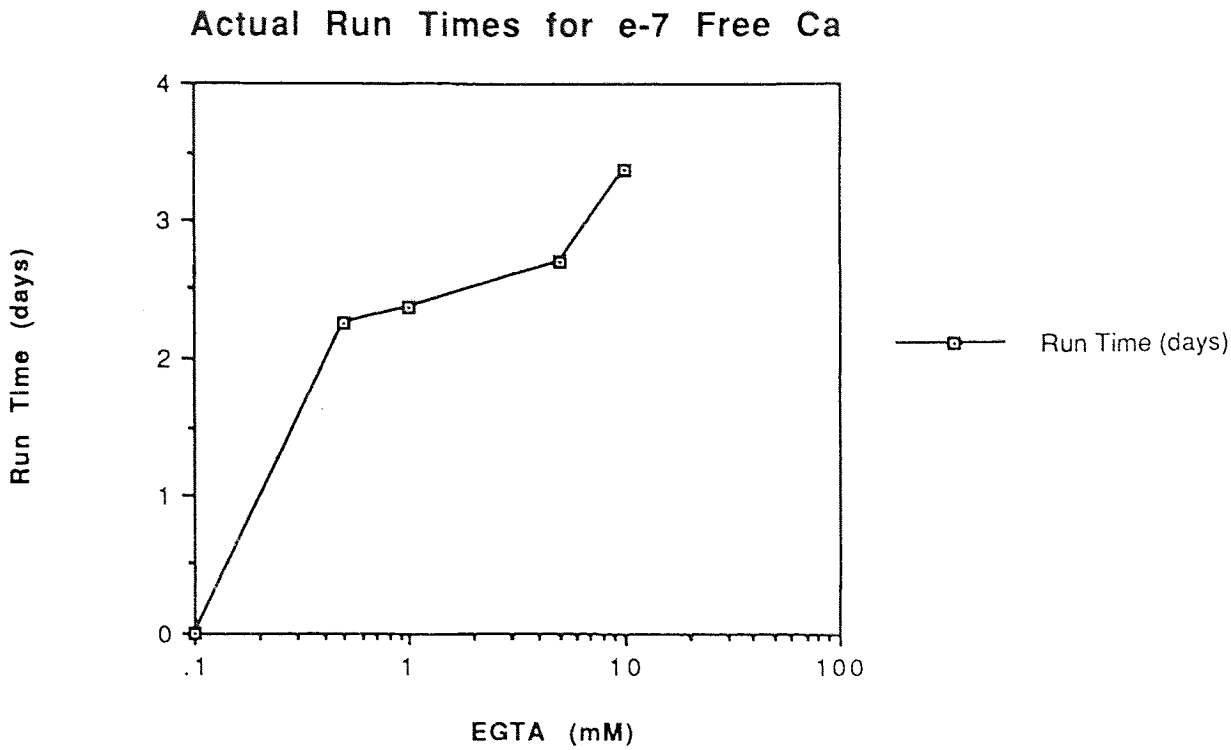


Figure 12 - Actual Run Times for e-7 Free Ca

The result for the $10^{-7}$ free calcium indicates that the model equations are sensitive to the EGTA concentration. While the experimental conditions are more often on the order of 0.1 to 1 mM EGTA, the current free Ca concentrations of $10^{-10}$ only exacerbates this sensitivity.



## Overall Gain in Compute Efficiency

Determining the optimum step-size for a particular model is often through trial and error. Typically, a series of runs are attempted, with varying step-size, and some estimate is made as to the accuracy and duration each value of step-size produces. The Adaptive method obviates the need for such exploratory simulations, since it will automatically select the optimum step, but must also consider this in terms of the minimum step size desirable. Because of the nature of numerical integration, which is very predictable in terms of its computational performance, it is very often easier to determine the step-size at which the simulation will take too long to run. This worst case scenario, where the step-size is so small that the simulation takes too long, is called the **StepLimit** and is a parameter used in the Adaptive algorithm. The choice of this parameter affects the overall efficiency for the algorithm.

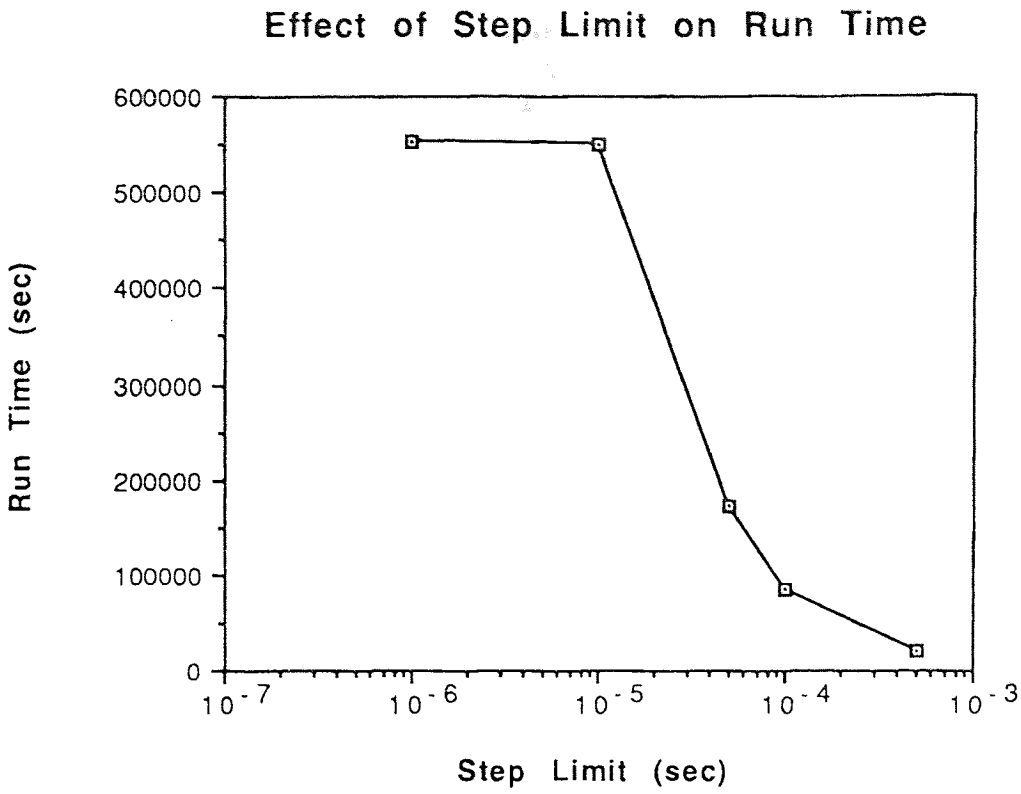


Figure 13 - Effect of StepLimit on Run Time

In Figure 13, the result of the runs in terms of the duration versus the **StepLimit** is illustrated. The runs varied in duration from about 5 hours to 153 hours. The range of values for the StepLimit were selected to bracket a typical value for the step-size of $10^{-4}$, used by Diecke, in studies of two-compartment models for calcium release. All values for the free calcium at the end of the run were identical to 14 decimal places. The figure suggests that moving from $10^{-6}$ to $10^{-4}$ in **StepLimit** represents a significant savings in run time. Furthermore, increasing the **StepLimit** to $10^{-3}$ has no significant effect on this system. From this, a maximum **StepLimit** of $10^{-3}$, and a minimum **StepLimit** of $10^{-5}$ were selected for the all successive runs in the ensuing studies.

The rationale for this effect is that a step size of $10^{-5}$ is a point beyond which the simulation will consume more than a reasonable amount of time. Clearly, the arbitrary selection of step size as a means of insuring precision for the calculation is not useful. Since some 2400 seconds of simulated time was experienced for each run, without any significant change in the free calcium (details follow in the next sections), the calculated results for the calcium release model are not sensitive to a particular **StepLimit**. If an arbitrary assignment should be made, then let it arbitrarily be the one that insures the greatest savings in Run Time.

Another important parameter for the Adaptive algorithm is the size of the **RequestStack**. This parameter controls a mechanism which buffers incoming requests for an increase in step while allowing requests for a decrease in step to be immediately acted on. It has a more subtle effect on the overall efficiency of the run, although not as dramatic as the **StepLimit** parameter. Exploring this **RequestStack** parameter is straightforward. First, collect a control simulation, without using the **RequestStack**. Then generate some variations, taking care to only vary the **StackSize**. The following Figure reveals the effect of the size of the **RequestStack** buffer.

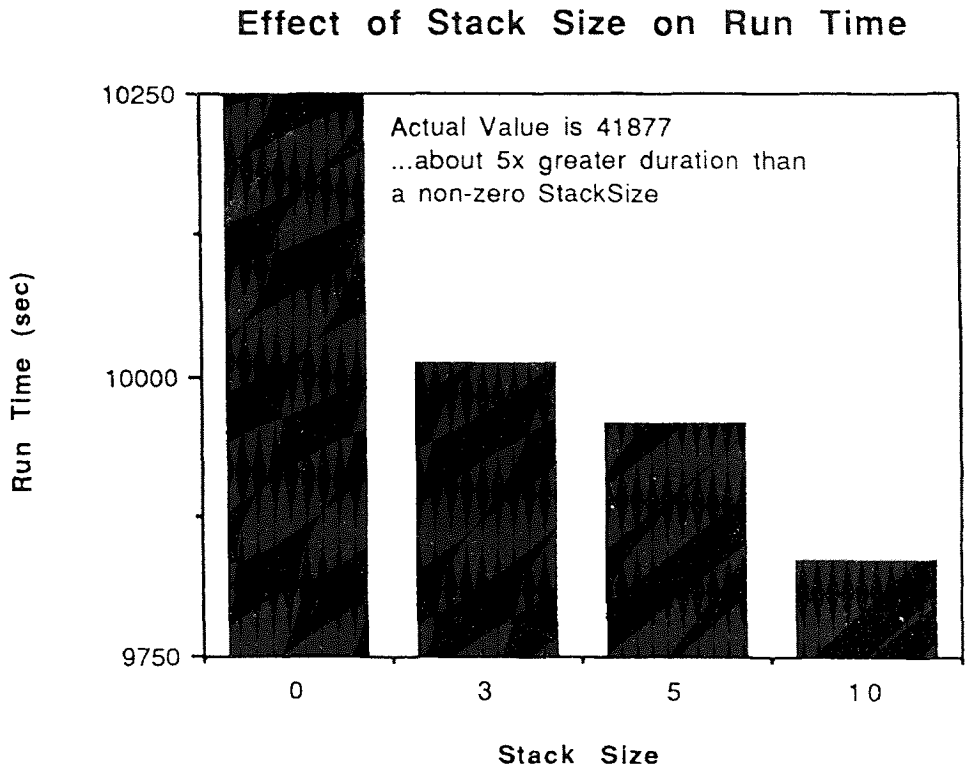


Figure 14 - Effect of Stack Size on Run Time

The scale is somewhat exaggerated in order to highlight the the slight differences between the set of parameters {3, 5, 10}. The difference between using the **RequestStack** and not using such a buffering mechanism is more dramatic and on the order of about one fifth the duration of the control run. Frequently, it has been observed that the adjustments to the **RequestStack** can oscillate around a given **ErrorRatio**, in terms of a rapid succession of increases followed by decreases, etc. The **RequestStack** mechanism circumvents this phenomenon by insuring that any request for an increase in **RequestStack** is motivated by a general trend. Figure 15 illustrates how the **StepSize** varies over the course of the simulation, in response to the variations in the size of the **RequestStack**.

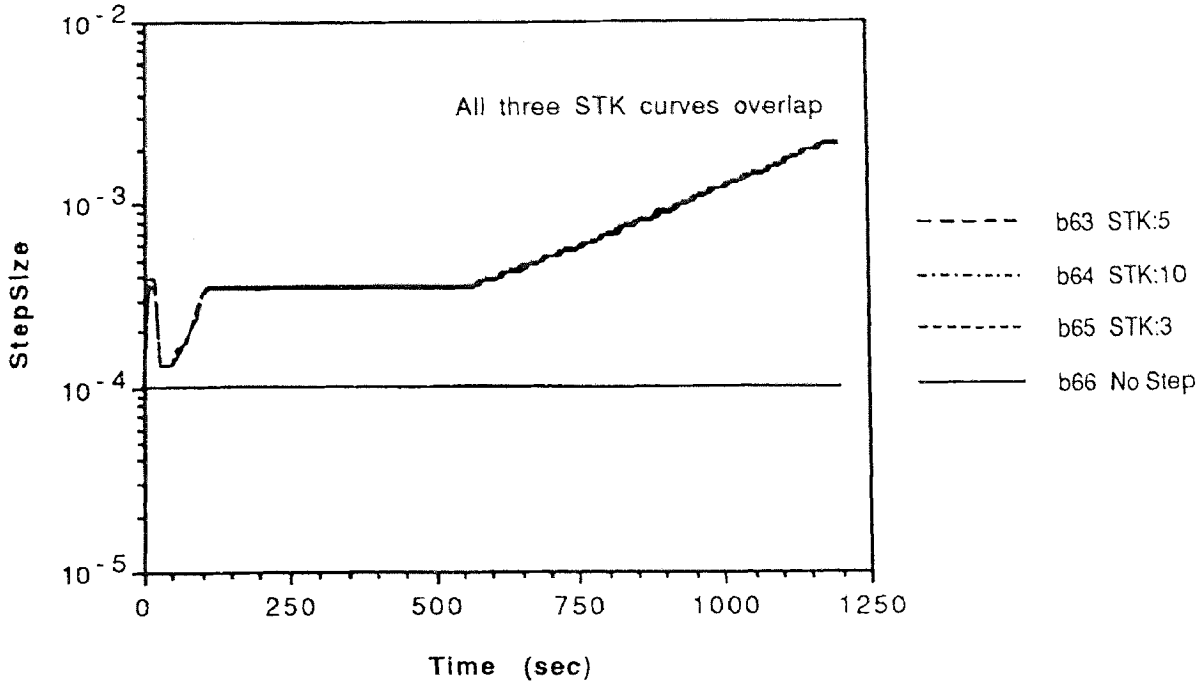


Figure 15 - StepSize as a Function of StackSize

Basically, we find that the same pattern of step size adjustment is present for each of the **StackSize** variants; that is, all three STK curves overlap for the duration of the run. Thus, it will be necessary to explore some other metrics before a conclusion can be drawn as to the effectiveness of each **StackSize** variant; this will be covered in later sections. The curve revealing how the **StepSize** is varying is more important now, since this is the observational foundation underlying the Adaptive step-size approach.

In Figure 15 there are four distinct phases. In the first 50 seconds (0->50), there is a rapid increase in step size representing the initial accommodation to the low inertia of the system. This rapid initial increase in step is obscured somewhat by the scale of the graph, and it should be noted that all four curves start at $10^{-4}$ **StepSize**. Within the first 50 seconds of the simulation the three STK curves rise form $1x10^{-4}$ to $4x10^{-4}$ and then drop to $1.3x10^{-4}$.

In the next 50 seconds (50->100), or phase two, system inertia increases and the step size drops suddenly to about $1.3 \times 10^{-4}$. This can be characterized as a response to the acceleration of the flux. At about 120 seconds, phase three, the step has increased to a constant level for the next 400 seconds. This represents the optimum step for the particular simulation (as defined by the **StepChange** parameters), once the initial flux velocities have become constant. In the fourth phase, at about 550 simulated seconds, the step size begins a constant linear increase, until the simulation ends. This can be characterized as the steady-state response, or equilibrium, for the system.

With regard to any possible limit on the change in the step size that is related by some function to the step change parameters, the slope of the fourth phase is clearly under no such constraint. Compared with the much steeper slope of the second phase, it suggests that the slope in the fourth phase may be reflective of the approach to steady state. This offers a significant check as to the planned duration of the simulation, to insure that all of the dynamic phenomena have been examined.

Finally, the overall difference between the step change algorithm and the constant step suggests that a vast improvement in the overall run time can be achieved, without sacrificing accuracy. This is on the order of 76%, for this system.

### How Does Precision Compare Between Adaptive and Non-adaptive Methods

There are differences between simulations performed with Adaptive step-size control and simulations with constant step-size. The issue is whether or not these differences are significant. To make this evaluation it would be necessary to compare some range of bulk properties, such as the Total $^{45}Ca$ (as is measured experimentally), and the smallest quantity measured in the model, which would be $^{45}Ca$ in the first compartment. This will allow some determination of how any fit of experimental data would be affected, and whether or not the model can accurately track the smallest values of its components.

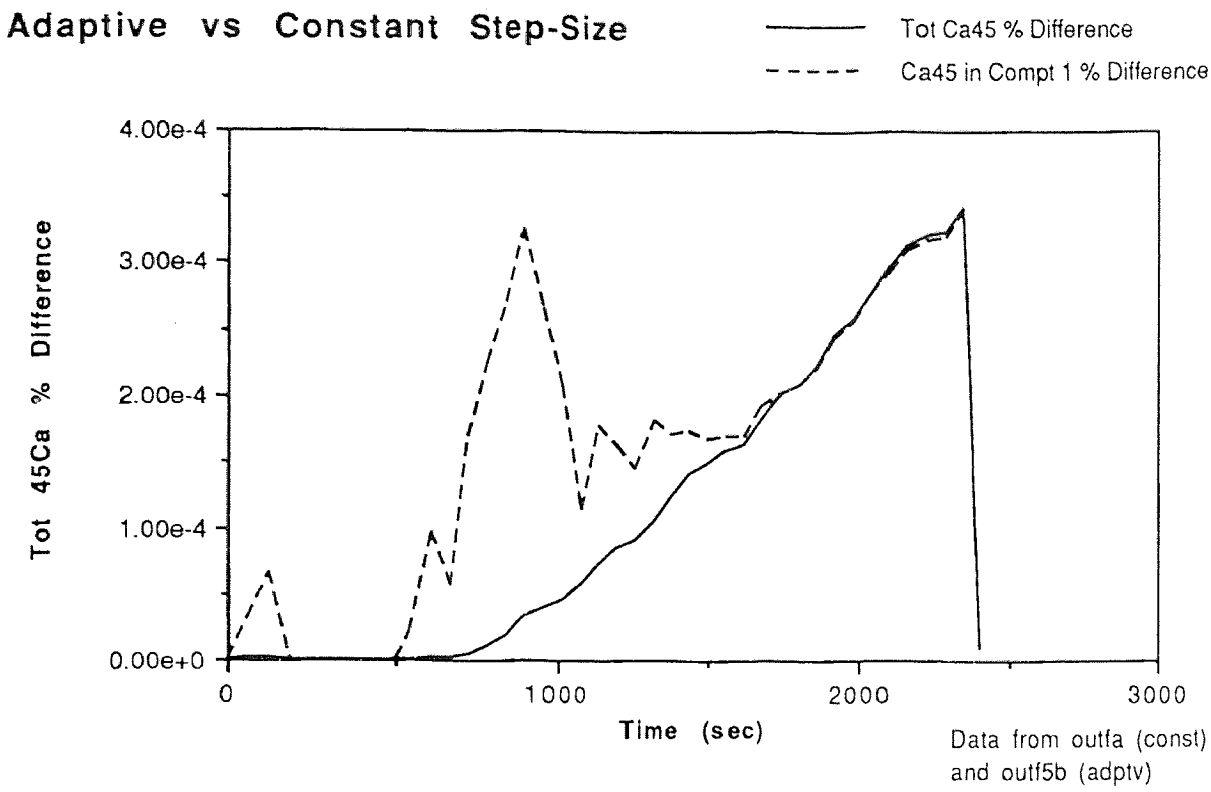


Figure 16 - Adaptive vs. Constant Step Size

This Figure summarizes the differences between Adaptive and non-adaptive runs and is the difference between the values, divided by the average of each pair of values, in percent. The overall, worst-case error is on the order of $4 \times 10^{-4}$% and, in terms of the Total $^{45}Ca$, is certainly well within any experimental error. In terms of the smallest quantity tracked by the simulation, the error is comparable in magnitude but has a significantly different time-course. At the end of the simulation run, the error becomes very small, suggesting that steady-state for the model can be achieved through multiple pathways.

The difference between the two curves suggests that the smaller component is more sensitive than the bulk $^{45}Ca$, and is reasonable given that there are four orders of magnitude difference in the values. The appearance of this error, relative to simulation time, coincides with the third phase of Figure 15 and indicates the the rapid increases in step size are over-estimating the $^{45}Ca$ in the first compartment. Apparently, this over-estimate is absorbed by the model, as the differences again begin to even out. In the ending phase of the simulation, as the **StepSize** is consistently increased, comparable levels of over-estimation are encountered. While the curves themselves are dramatic it should not be overlooked that the magnitudes of difference, in percent, are very small. The curves lend greater insight into the response of the Adaptive

step-size control algorithm more than they reveal any impact on the result of the simulation.

The ability of the model to absorb these small changes can be made more prominent if the ratio of the two error curves is plotted, as in the following Figure.

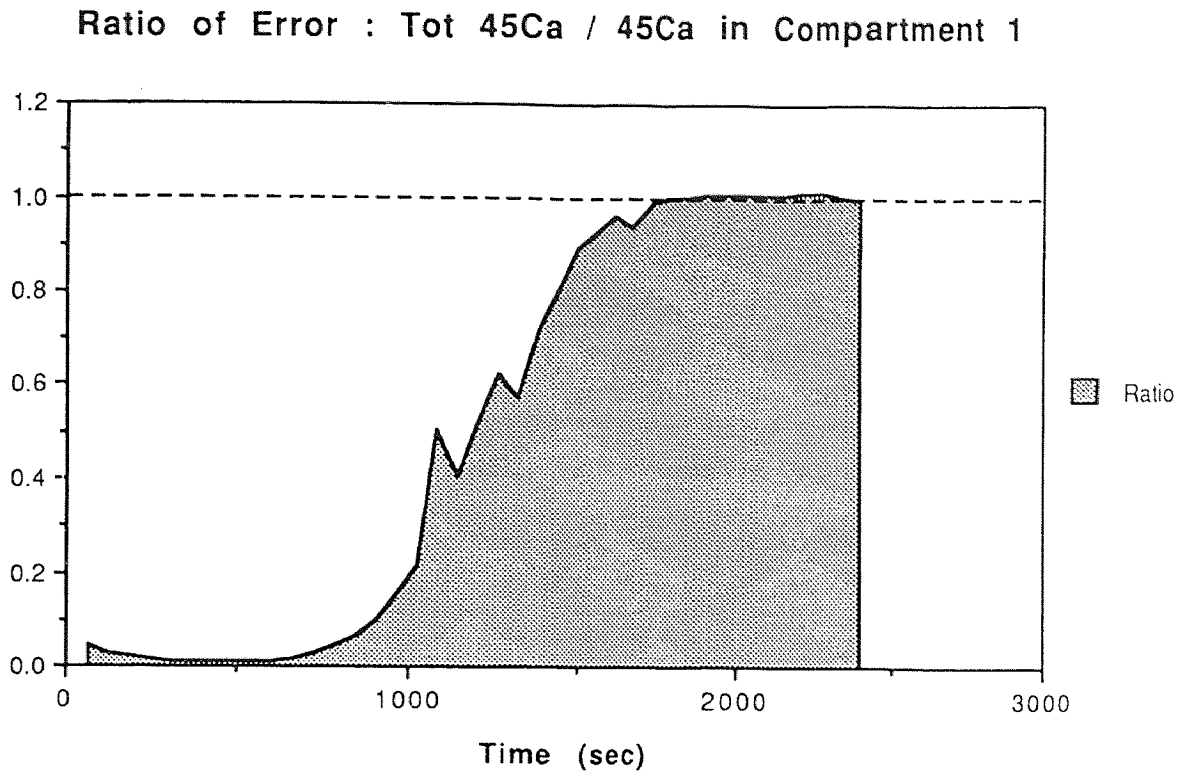


Figure 17 - Ratio of Error : Tot $^{45}Ca$ - $^{45}Ca$ Compartment 1

Note that the curve does not begin at the origin, where the error between Adaptive and Constant step control schemes is zero, but at the first report interval at 60 seconds. The ratio is small initially and then grows as the smaller values of Total $^{45}Ca$ are under-estimated. It is unlikely that the actual value is in error because the differences are within the precision for the numbers used and error due to numerical round-off would not be present. It is more likely that the differences are attributable to the increased step-size for the Adaptive run. This would suggest that the 1st compartment is changing more rapidly than the Total $^{45}Ca$, which is supported by experiment (Stout). This rapidly changing component is not being tracked efficiently by the Adaptive algorithm and perhaps some changes in the **StepChange** parameters should be explored. The actual error, however, does not support that suggestion.

## Validation of the Simulation Engine

The most reliable approach to verification of a simulation is to evaluate a well understood model and compare its results to that provided by the simulation. In this, we must conduct a series of tests with the model in order to build up confidence in its predictive power. An even better choice is to use a model that can be solved analytically so that a numerical comparison can be made, and with this reduce the number of iterations required to achieve confidence. Such a model was borrowed from Randell, and simulates a two-compartment *leaky tank*.

The parameters used for this model are of no particular consequence since the objective is a explore a model which has an analytical solution and can also be evaluated numerically. Randell's software is in BASIC and should not be considered useful for a more computationally intensive simulation. It was intended as a teaching exercise in homeostatic physiology.

What follows is the analytical solution of the model, its representation in the current simulation environment, and a brief analysis of the results.

### Analytical Solution

Essentially, this example model accounts for accumulation and clearance of some substance according to first order kinetics. This model can be considered analogous to a two-compartment leaky tank, as diagrammed below:

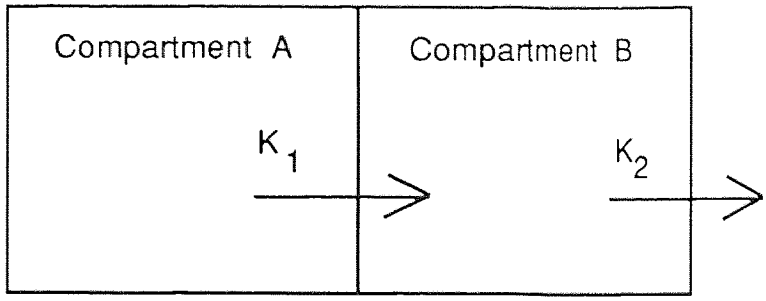


Figure 18 - Two-compartment Leaky Tank Model

Here, compartment A is charged with some amount of material which flows into compartment B, where the material disappears. The parameter K1 is for the rate of exchange between compartments A and B, and the parameter K2 is for the rate of disappearance. All exchanges are one way, as indicated.

The leaky tank is a suitable model for compartmental analysis of various species in a physiologic system (see Murthy), and is suitable as an analogy for absorption, metabolism and loss of the components.

The equations for the net rate of change are as follows:

$$(1) \quad dA = -K_1 A$$

$$(2) \quad dB = K_1 A - K_2 B$$

Analytically, we find for equation (1), by inspection, that

$$(3) \quad A = e^{-k_1 t}$$

Substituting (3) into (2) and rearranging, we get

$$(4) \quad dB + k_2 B = k_1 e^{-k_1 t}$$

It is also easy to determine from (4) that there are two partial solutions

$$(5) \quad B_{\text{Transient}} : C_2 e^{-k_2 t}$$

$$(6) \quad B_{\text{Steady State}} : C_3 = \frac{k_1}{k_2 - k_1}$$

Combining the results found in (5) and (6) gives a total solution of

$$(7) \quad B(t) = C_1 e^{-k_1 t} + \frac{k_1}{k_2 - k_1} e^{-k_1 t}$$

$$\text{where } C_1 = - \frac{k_1}{k_2 - k_1}$$

or,

$$(8) \quad B(t) = \frac{k_1}{k_2 - k_1} ( e^{-k_1 t} - e^{-k_2 t} )$$

This example model was selected for its simplicity so as to provide an exact analytical solution with which to verify the model. For an introduction into more formal mathematical representations of models, see Casti. As a note, it is possible to derive an analytical solution to a three-compartment model (see Jacobs) although that system is not applicable here.

The result of this verification is presented in Figure 19, and is in good agreement with that found by Randell. For this particular set of diffusion parameters, there is accumulation in the second compartment which is maximal at 3 simulated seconds.

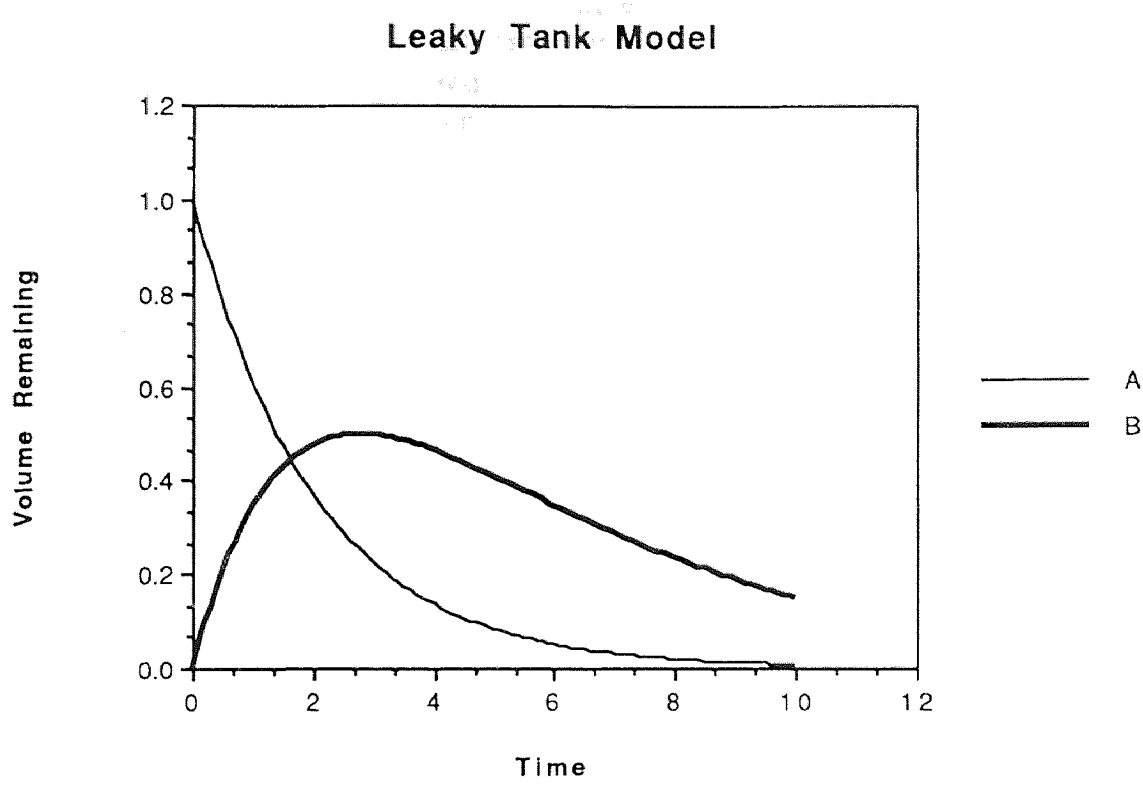


Figure 19 - Leaky Tank Results

To review the implementation of these equations, see Figures 9 and 10. The CPU time needed to evaluate this model is generally less than one minute, although a typical PC could take about five minutes. Reviewing the numerical results yields errors on the order of $10^{-7}$ to $10^{-10}$ percent when compared with the analytical values. This is certainly acceptable for compartmental modeling, and suggests that the simulation engine is operating as expected.

### Isotope Exchange

The first extension to the two-compartment model by Diecke is the additional equations for $^{45}Ca$. Taking these equations into consideration precipitated the need for additional compute resources. As the number of equations in a given model increases, so goes the run duration for the simulation (Cooney). The addition of $^{45}Ca$ introduced four more equations, for a total of ten, while the run times went from one day to five days on an IBM PC with an accelerator card. Moving from the IBM PC to the HPA900 reduced the run duration to about two hours. The application of the adaptive algorithm reduced the run durations to between 6 and 150 minutes, depending on how close the system was to equilibrium.

The basic run procedure for this experiment is to generate estimates for the Ca/EGTA equilibrium and simulate these parameters until they are at steady-state. With the calcium distributed as 50% Ca and 50% $^{45}Ca$, the labeled species are removed from the reservoir and replaced with equal amount of the unlabeled species. This keeps the overall concentration the same so that an isotope exchange or $^{45}Ca$ washout will take place. A wide range of conditions will be explored to assess how the calcium and EGTA concentration might affect the washout rates. The following Figure 20 summarizes the model.

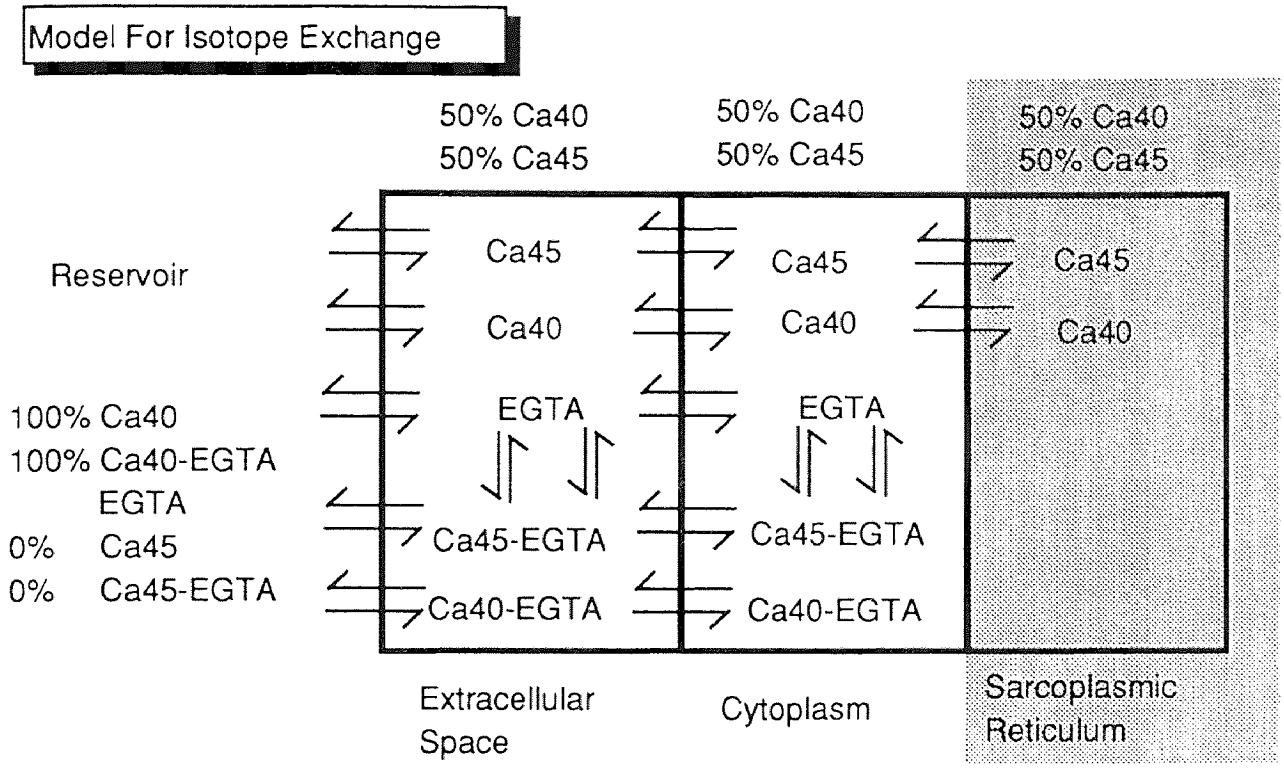


Figure 20 - Model For Isotope Exchange

The third compartment is shaded to indicate that it is not considered in this part of the study and illustrates that two more equations will need to be added in order to have a three-compartment model for $^{45}Ca$ washout. The horizontal arrows indicate diffusion between compartments while the vertical arrows indicate a reaction within a compartment. The EGTA because of its size does not cross into the third compartment.

The Total $^{45}Ca$ remaining is calculated and the form of the washout curve is a double exponential decay, easily fitted by computer to reveal the rate constants for each of the components. These rate constants are much more

revealing than the washout curves (omitted), as the following Figure 21 reveals.

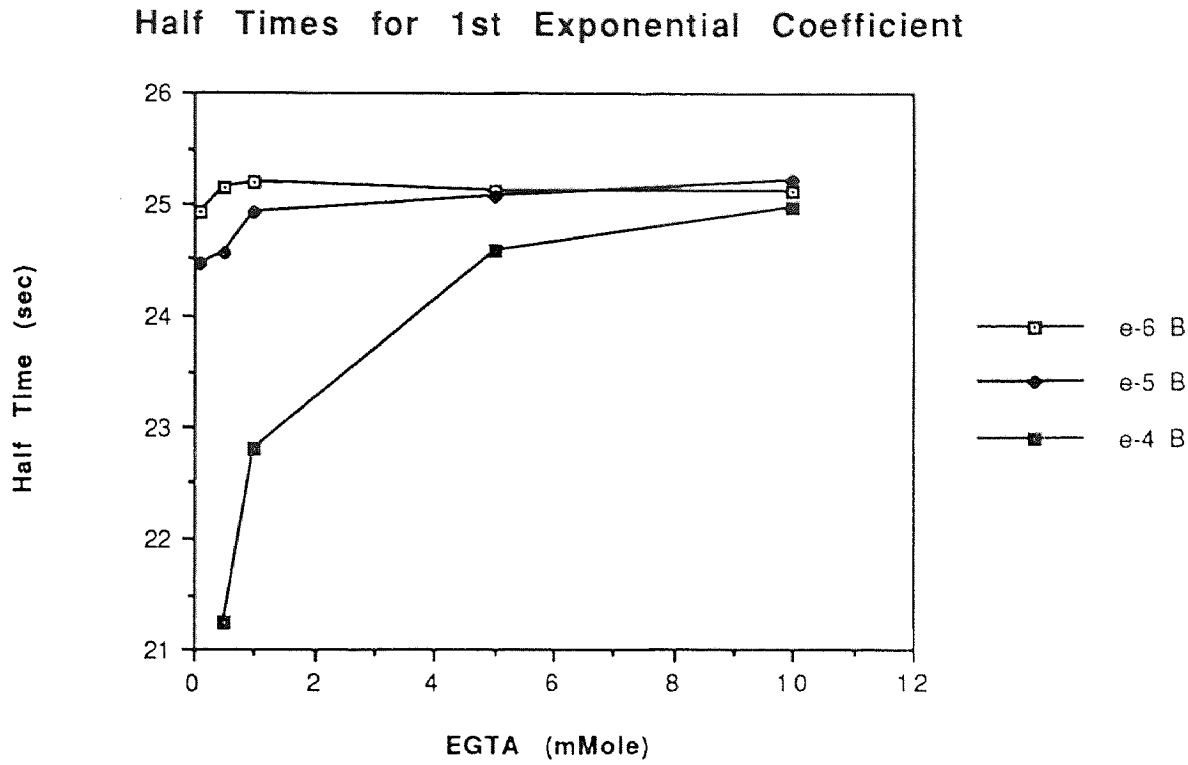


Figure 21 - Half-times for First Exponential Term

This set of curves depicts the half times for first washout component, fitted using GraphPad. The range of calcium explored is higher than that used during the experiment (typically $10^{-7}$ and recently $10^{-10}$) but is still useful in illustrating the relationship between Ca, EGTA, and washout rates.

The primary observation of Figure 22 is that calcium and EGTA have a distinctly non-linear relationship. While the calcium levels that yield the more unusual behavior are not used experimentally, it is not unlikely that a transient will generate some part of this phenomenon. With the **e-4 B** (Ca $10^{-4}$) curve at low EGTA, the washout has over-run the buffer system; while at high EGTA concentrations the buffering has linearized the washout differences attributable to calcium concentration. The **e-6 B** curve has a different effect. While this curve seems to be linearly related to the EGTA concentration, the washout half times are increasing slightly at low EGTA. This suggests that a low concentration of buffer and a low free calcium level will result in an underestimation of washout rate. Similar results are obtained for the second exponential component.

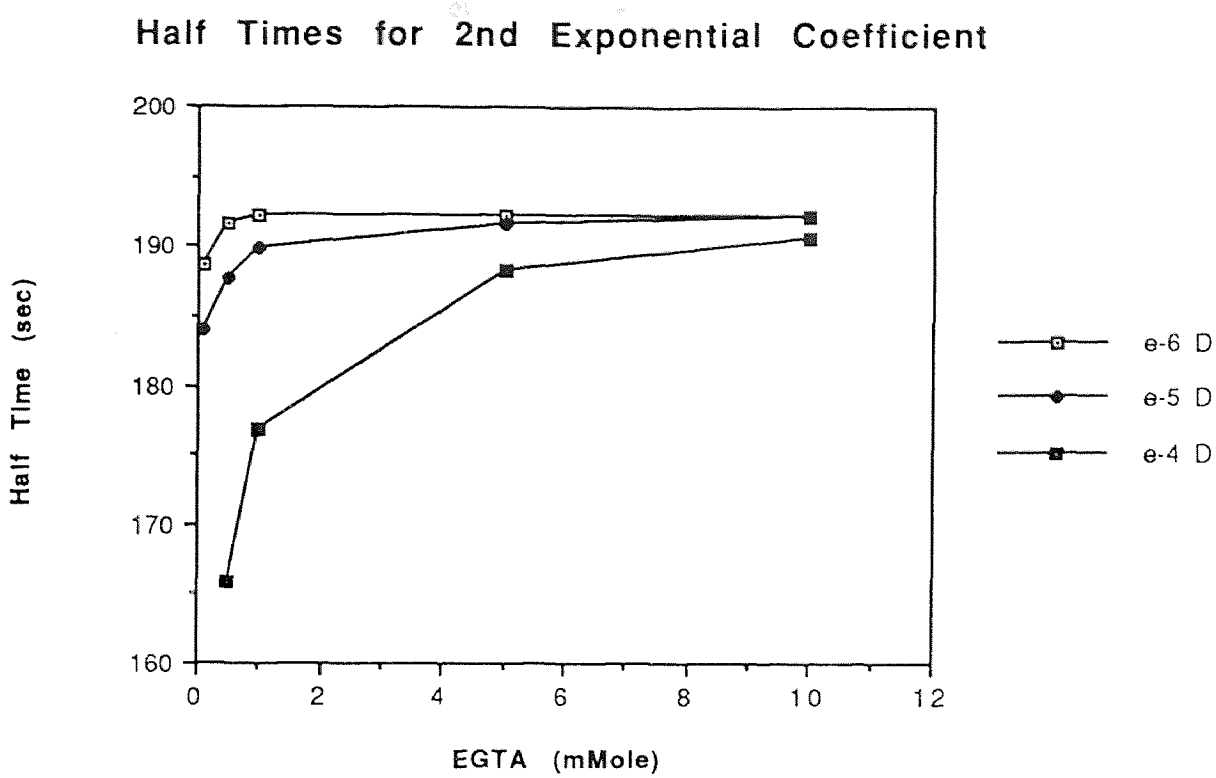


Figure 22 - Half-times for Second Exponential Term

This set of curves has the same non-linear behavior as Figure 21 and is in better agreement with expectations. The EGTA buffer system is acknowledged to be exhibit linear performance at low EGTA (0.1mM to 1.-mM) and low calcium ($10^{-7}$). While the non-linearity at higher calcium and EGTA is present as before, the **e-6 D** curve is flat over the major range of EGTA. The left hand part of the curve still suggests that calcium is capable of overwhelming the buffer system and underestimating the release half-time.

This may be a critical point and additional runs will need to be considered before a conclusion is drawn. This is because any significant calcium release is certainly capable of an order of magnitude jump in concentration. Even if the jump is from $10^{-8}$ to $10^{-7}$, as opposed to $10^{-6}$ to $10^{-5}$, there will certainly be an underestimation of the washout half time. Whether or not this is significant experimentally, when the half times differ by 5 or 10 seconds, the calcium washout rate will appear to be faster at low calcium, low EGTA concentrations.

**Release of Calcium from the SR**

Before the calcium that participates in Calcium Release can be distributed among the components of some future model, it is important to quantify that the quantity of calcium participating in release can be measured with current experimental techniques. This simulation experiment will explore a number

of release rates to determine what effect on the total calcium can be perceived. Since the present model does not include a mechanism to cause activation of the release phase, the simulation will be re-started at 1200 simulated seconds with a new parameter for diffusion from the third compartment, and will run for another 1200 seconds.

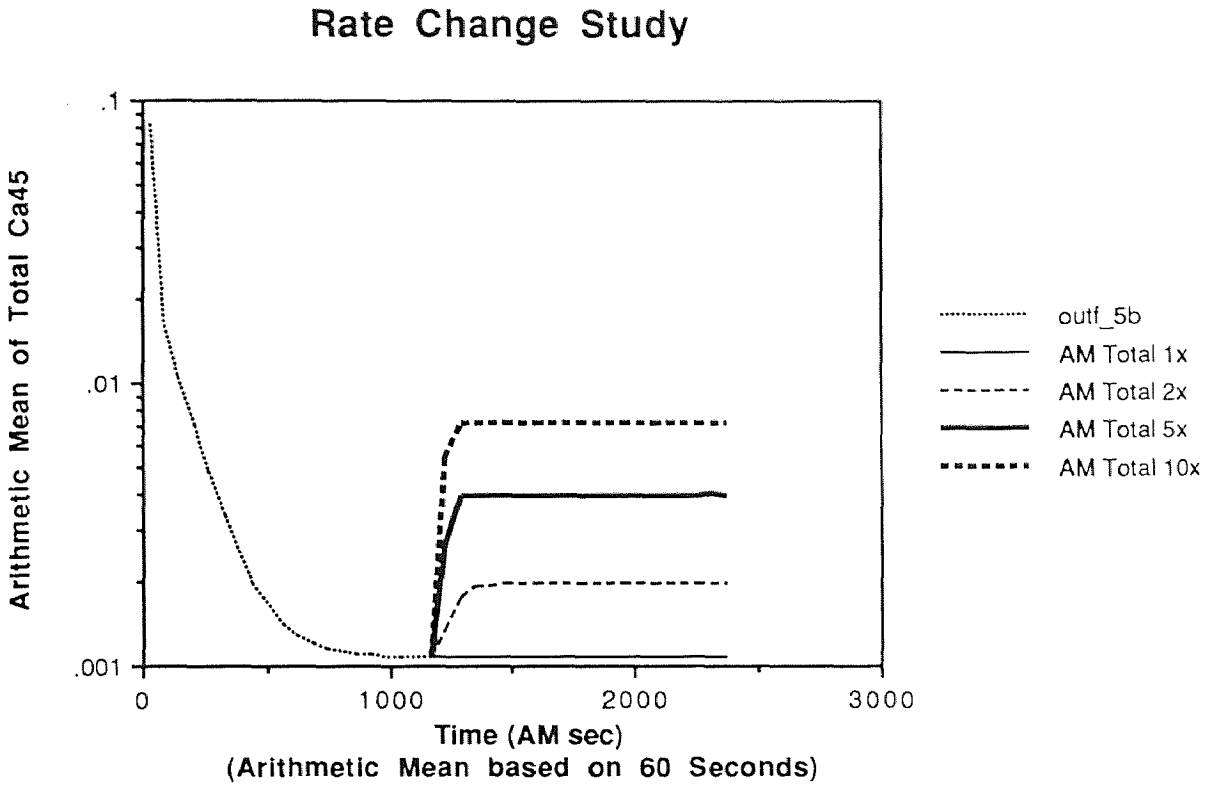


Figure 23 - Rate Change Study AM Total $^{45}Ca$

Each of the curves in Figure 23 was developed by collecting all of the $^{45}Ca$ components and calculating the rate or arithmetic mean as described in the Methods. To insure that the inflections in the release variants were not due to the restarting of the simulations, a control run using the original parameter values was included with the variants. The results of the control run were identical with the initial run. The initial run is the dotted line on the left and the continuation of that curve is indicated as solid where the control run is present.

The two outstanding features of Figure 23 are the levels at which the calcium release stabilizes and the slope of the rise from the onset of the new rate, until the final level is achieved. These features can be accentuated if the curves are normalized with respect to the baseline, as in the following Figure 24.

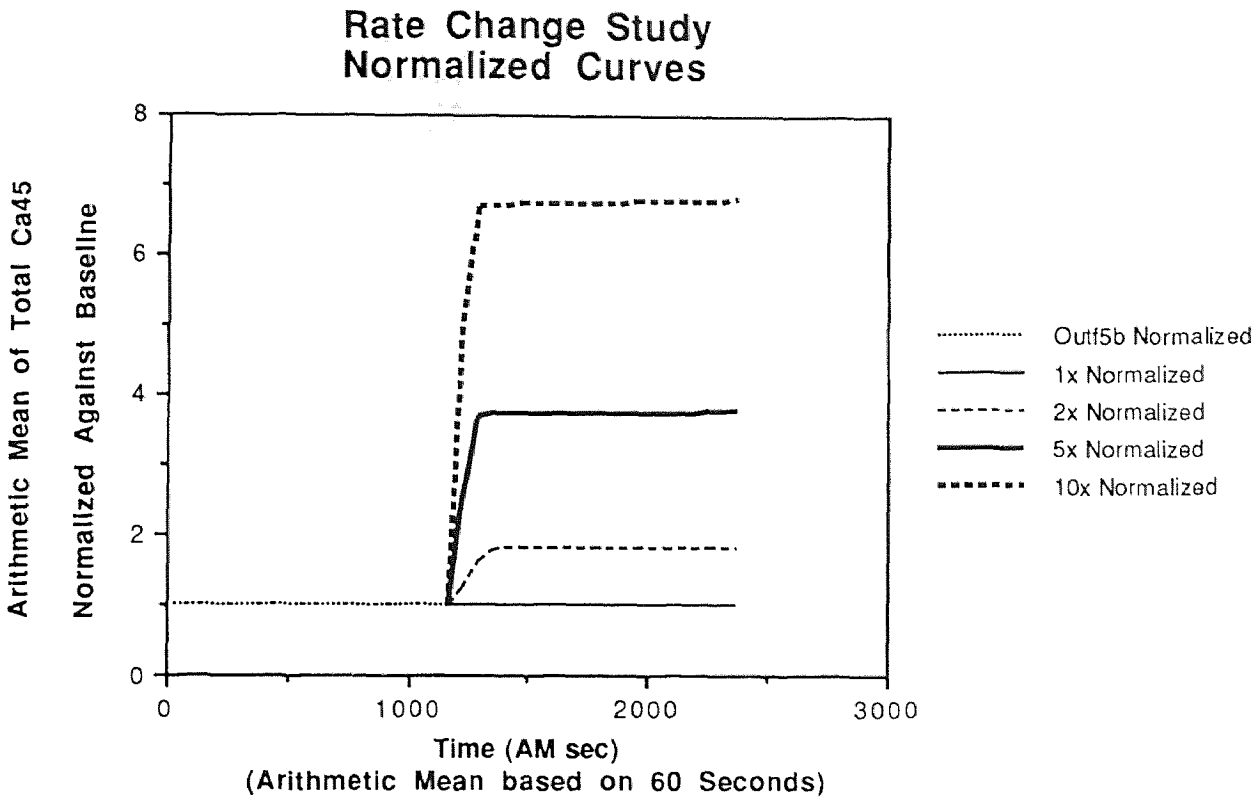


Figure 24 - Rate Change Study : AM Total $^{45}Ca$ Normalized

Consider the slopes of the changes in each of the variants. For the 2x variant the slope is noticeably more gradual than the variant at 10x. By extrapolating reference lines perpendicular to the baseline and intersecting the onset of the plateau it can be determined that the 10x variant actually reaches its plateau before the 2x run is halfway to its own plateau. This suggests that a large release of calcium is more quickly absorbed by the model than a smaller release. In terms of a physiologic calcium release this would suggest that a large release of short duration (impulse) would be rapidly equilibrated (absorbed) by the tissue. This would result in slightly elevated levels of $^{45}Ca$ over a longer period of time. Following this reasoning, a smaller release, which equilibrates more slowly, would probably result in a small transient increase and a lower equilibration level in comparison with a large release.

Before addressing the relationship between the plateau values of the variants it would be useful to point out another issue that arises from this type of analysis: the interval at which the arithmetic mean is calculated. This relates to the earlier issue of how many data samples are appropriate. Figure 25 presents some of the differences between data sampling at 10 seconds and 60 seconds and the effect this has on the final determination of the relationship between rate variants and their final plateaus.

| Run | 1x AM | 2x AM | 5x AM | 10x AM |
|---|---|---|---|---|
| rs_5d | 1.77090E-04 | 3.14960E-04 | 6.57700E-04 | 1.18674E-03 |
| rs_5b | 1.06272E-03 | 1.88083E-03 | 3.94653E-03 | 7.12051E-03 |
| *Absolute Difference* | 0.00088563 | 0.00156587 | 0.00328883 | 0.00593377 |
| *Average %* | 142.9% | 142.6% | 142.9% | 142.9% |
| *Normalized Results* | | | | |
| rs_5d:AM base10 | 1 | 1.778530691 | 3.71393077 | 6.701338303 |
| rs_5b:AM base 60 | 1 | 1.769826483 | 3.71361224 | 6.700269121 |
| **Average** | **1.0** | **1.8** | **3.7** | **6.7** |

Figure 25 - Arithmetic Mean Summary

In the top portion of Figure 25 are the final values for each of the variants obtained by calculating the arithmetic mean on the Total $^{45}Ca$ remaining in the model. The middle box calculates the absolute difference between the 10 second and 60 second basis and relates it as the average of of each of the percent errors for each variant. The net result of this is that the runs, despite their sampling interval, are linearly related. The bottom portion shows that when the curves are normalized, the linear relationship is removed and the results are in agreement. The conclusion from this is that normalizing the curves facilitates comparison with other data sets, and is independent of the sampling interval used to calculate the rate (arithmetic mean).

The increase in the rate constant does not have a corresponding increase in the magnitude of the maximal change in rate but they are linearly related as Figure 26 reveals.

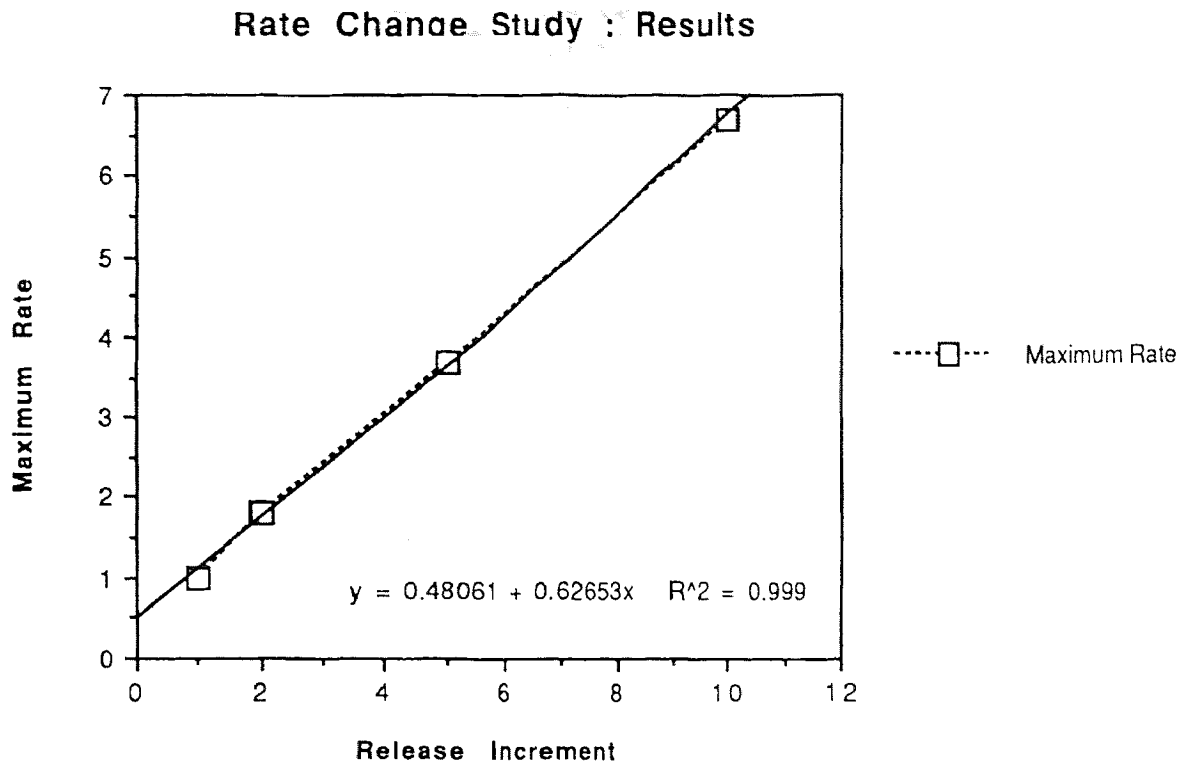


Figure 26 - Rate Change Study : Results

It seems likely that this represents the response of the buffer to the release of calcium rather than that of the compartment. In simple systems (those without a buffer reaction) changes in rate resulting from concentration differences are linearly related. It would be interesting to see what relationships this curve might have for varying free [Ca] and [EGTA], considering the behavior of the buffer system previously discussed in Figures 21 and 22.

Figure 27 summarizes the impact of adaptive vs constant step-size when considering the Total $^{45}Ca$ that would be available for experimental verification. This will help determine if the adaptive algorithm has the potential to introduce erroneous results.

| Run | 1x Tot Ca45 | 2x Tot Ca45 | 5x Tot Ca45 | 10x Tot Ca45 |
|---|---|---|---|---|
| rs_5d | 6.15184E-03 | 6.14278E-03 | 6.11282E-03 | 6.06391E-03 |
| rs_5b | 6.08467E-03 | 6.13889E-03 | 6.10478E-03 | 6.04951E-03 |
| *Absolute Difference* | 6.71728E-05 | 3.89214E-06 | 8.04362E-06 | 1.43951E-05 |
| *Average %* | 1.10% | 0.06% | 0.13% | 0.24% |
| *Step Size* | 1.2E-02 | 9.5E-04 | 1.4E-02 | 8.9E-03 |
| *Log Difference* | 2.08 | 0.98 | 2.15 | 1.95 |
| *Normalized Results* | | | | |
| rs_5d:AM base10 | 1 | 0.9985 | 0.9937 | 0.9857 |
| rs_5b:AM base 60 | 1 | 1.0089 | 1.0033 | 0.9942 |
| **Average** | **1.0** | **1.0** | **1.0** | **1.0** |

Figure 27 - Total $^{45}Ca$ Summary

The top section contains the total $^{45}Ca$ at approximately 1350 seconds. There is a five second difference between the **rs_5b** (1350 sec) and **rs_5d** (1355 sec) samples and this is due to the difference in output intervals for each of the simulations. The intent is not to measure the absolute differences, which were presented earlier, but to evaluate the impact of these slight differences on the results as a whole.

The middle section calculated the differences between the actual adaptive step-size, and the constant $10^{-4}$ step-size of the non-adapting run. There does not appear to be a conclusive relationship between the Average % Error and the running step-size. The averages ending up at approximately 1.0 suggests that relying simply on the absolute value of $^{45}Ca$ is not sufficient to distinguish differences within the smaller components (volume based) of the model. Only by examining the relative rates of change can the effects of the various components be delineated.

The 1.10% difference in the **1xTot $^{45}Ca$** column is probably due to error introduced by the adaptive step algorithm. Adaptive step is more suitable when there is something to adjust for, in terms of error. When the system approaches steady-state, the step-size increases more than is accurate for the particular calculation. This will tend to under-estimate the more accurate value, which is established by running at a constant step, as can be inferred from the table. The adaptive method should be considered more appropriate for simulations in which some significant transient is expected to occur which would give the adaptive algorithm something to respond to. Another advantage in using adaptive step-size control is that rapid decreases in step size will be made even if the initial step is inaccurate. The constant step-size result will not reveal when the step-size is inappropriate, except in the case where the step is so large that the calculation overflows. This makes the

adaptive approach more attractive, especially in determining the optimum step size, as well as the overall computation efficiency. The arbitrary selection of step-size is likely to be conservative and would result in a smaller-than-needed step-size. While the overall impact on computational precision is small and subjective, the effect on the total duration of the simulation run can be quite dramatic and undesirable.

# Conclusions

## Adaptive StepSize Control

The accuracy of the Adaptive Algorithm is good, at approximately $10^{-4}$% of comparable runs using constant step-size. The additional overhead to implement the Adaptive control is minimal, compared with traditional step size control algorithms which typically make a step-size correction at each calculation interval. (See Press for examples of these methods.) The flexibility in sizing the response parameters, and the availability of performance monitors, should allow the Adaptive method to be extended to other model systems. The Algorithms' intended function is to easily handle non-linear models, or models with pronounced transient events, and it should be quite versatile in handling these more difficult calculations.

The bottom line for the performance of the Algorithm, questions of precision notwithstanding, is in how much it improves the overall duration of the run. The following Figure summarizes the performance increases for the most current generation of Computer platforms considered in this study.

| CPU | Constant StepSize (minutes) | Adaptive StepSize (minutes) |
|---|---|---|
| SG320VGX | 58 | 12 |
| HP9000 | 119 | 25 |

Figure 28 - CPU Performance Increase, Adaptive vs Constant StepSize

The parameters used were **outf5a** and **outf5b**, for 12 equations, 0.1mM EGTA, $^{45}Ca$ Washout. The simulation results were exact, which is some testimony to the better standardization of the C programming language. The run durations reveal that an improvement of about 5-fold can be expected and it would appear that it is independent of the CPU platform. The SGI, a more recent product, is two times faster than the HP9000, for this application. The suggestion that the results only seem independent of hardware is because both of these machines are of a much higher performance computationally than those used for the bulk of this study. The improvements in performance between a PC and a minicomputer were much more dramatic because a PC is not intended for this type of intensive computing. It is suggested, however, that any extension of the adaptive step-size method be limited to a higher performance compute platform since this would be more in line with the goal of being able to achieve some type of parameter study.

## Modeling Environment

Overall, the simulation environment has alleviated most of the problems originally encountered. The management of multiple models, straightforward generation of model parameters, and facility to batch and schedule a series of runs, are clear steps towards streamlining the simulation process and improving its reliability. The generation and maintenance of the parameter sets is still cumbersome, and really needs an application program to take care of the details for generating the files.

Part of the motivation to move the simulation engine over to a UNIX compute platform was to take advantage of the additional utilities for data reduction that are available. The most useful tool is called AWK (Aho), and the ongoing plan is to convert all of the data extraction and analysis procedures to UNIX scripts and AWK programs. UNIX is a programming language in its own right, and its additional tools would go a long way towards completing the automation of the simulation process.

At that point when the new utilities are developed, and a better user interface for the generation of the parameters is available, the facility for initiating a comprehensive parameter study will be achieved. The ongoing benefit in working towards this goal is that in the development of these additional tools a more consistent framework for doing large-scale simulations is achieved, regardless of the application area.

## Effectiveness of the 3 Compartment Model

Clearly the changes in calcium can be tracked and the contribution of the SR quantified. Whether or not this can be achieved experimentally is not clear. The apparent ability of the model to buffer the transient calcium release clearly suggests that the phenomenon will not be characterized with current experimental procedures. It remains a challenge to physiologically validate the predictions for the Sarcoplasmic Reticulum, obtained with the current model. Until then, work can progress by modeling the other significant components of the SR to see what their likely contribution would be, in anticipation of improved experimental techniques that might validate those new findings.

There are a number of opportunities for future work. Some of the results need to be extended with additional conditions in order to more completely characterize the effect of the EGTA buffer system on calcium release. Once the time duration of release for the SR has been established, it would be interesting to develop a kinetic model of the SR to see if initial parameters for the calcium pump and calcium sequestering can be developed based on the number of those components. To provide more insight into the tissue experiments, it would be challenging to quantify the variability of the 1$^{st}$

compartment (presently attributed to physical differences in the connective tissue) to see if it would account for the variability in those results.

# Appendix

## References


**Abelson**, H., Eisenberg, M., Halfant, M., Katenelson, J., Sacks, E., Sussman, G.J., Wisdom, J., Yip, K. : Intelligence In Scientific Computing, Communications of the ACM: (1989) May Volume 32 Number 5

**Aho**, A.V., Kernighan, B.W., Weinberger, P.J. : The AWK Programming Language Addison-Wesley Publishing Company : Chapter 6 (1988)

**Allen**, J.C., Seidel, C.L. : The Regulation of Calcium in Smooth Muscle, appearing in "Sarcoplasmic Reticulum in Muscle Physiology Volume II", Entman, M.L., Van Winkle, W.B. Editors, CRC Press Inc. (1985)

**Biosym** Technologies : Discover 2.6 Reference Manual

**Bond**, M., Kitazawa, T., Somlyo, A.P., Somlyo, A.V. : Release and Recycling of Calcium by the Sarcoplasmic Reticulum in Guninea-Pig Portal Vein Smooth Muscle, Journal of Physiology: (1984), 355, pp 677-695

**Carnahan**, B., Luther, H.A., Wilkies, J.O. : Applied Numerical Methods. Addison-Wesley : 361-366 (1968)

**Casti**, J.L. : Alternate Realities - Mathematical Models of Nature and Man John Wiley & Sons, Inc.: 1-44, 466-474 (1989)

**Cooney**, D.O., Biomedical Engineering Principles : An Introduction to Fluid, Heat, and Mass Transport Processes, Marcel Dekker, Inc. (1976)

**Diecke**, F.P.J., Gardner, J., Hausser, R. : Mathematical Modeling of the Simultaneous Diffusion of EGTA, Ca-EGTA, and Ca2+ in a Two-Compartment System Representing Skinned Smooth Muscle, Preprint

**Dohi**, Y., Aoki, K., Fujimoto, S., Kojima, M., Matsuda, T. : Alteration in Sarcoplasmic Reticulum-dependent Contraction of Tail Arteries in Response to Caffine and Noradrenaline in Spontaneously Hypertensive Rats, Journal of Hypertension 1990, Vol 8 No 3

**Dostal**, D.E., Murahashi, T., Peach, M.J. : Regulation of Cytosolic Calcium by Angiotensins in Vascular Smooth Muscle, Hypertension, Vol 15, No 6, Part 2, June 1990

**Endo**, M. : Calcium Release from Sarcoplasmic Reticulum, Current Topics in Membranes and Transport, Volume 25 1985

**Hamming**, R.W. : Numerical Methods for Scientists and Engineers. McGraw-Hill Book Company, Inc.: 211-222, 370-382, 394-401 (1962)

**Hasselbach**, W. : Structural and Enzymatic Properties of the Calcium Transporting Membranes of the Sarcoplasmic Reticulum Annals of the New York Academy of Sciences Vol 137, Art 2. pp 403-1048 (1966)

**Hurwitz**, L.: Characterization of Calcium Pools Utilized for Contraction in Smooth Muscle, appearing in Smooth Muscle Pharmacology and Physiology, Worcel, M., Vassort, G. Editors (1975), Vol 50, pp. 369-380

**Iyengar**, S.S., Rao, R.M., Quave, S. : Chapter 2; A Four-Level Software Engineering Approach to Model Complex Biological Systems, "Computer Modeling of Complex Biological Systems", Iyengar, S.S. Ed. CRC Press, Inc.: 13-19 (1984)

**Jacobs**, J.R. : Analytical Solution to the Three-Compartment Pharmacokinetic Model, IEEE Transactions on Biomedical Engineering, Vol 35, No. 9, September 1988

**Jones**, A.W. : Vascular Smooth Muscle and Alterations During Hypertension, appearing in "Smooth Muscle: an assessment of current knowledge", edited by Bulbring, E., Brading, A.F., Jones, A.W., Tomita, T., University of Texas Press (1981)

**Kootsey**, J.M. : Complexity and Significance in Computer Simulations of Physiologic Systems Federation Proceedings Vol. 46, No. 8, June 1987

**Law**, A.M., Kelton, W.D. : Simulation Modeling and Analysis McGraw-Hill, Inc. (1982)

**MacLennan**, D.H. : Isolation of a Second Form of Calsequestrin Journal of Biological Chemistry Feb. 249(3) 1974

**Miyamoto**, H., Kasai, M. : Asymetric Distribution of Calcium Binding Sites of Sarcoplasmic Reticulum Fragments Journal of Biochemistry (Tokyo) Mar. 85(3) 1979

**Murthy**, E.V.K. : Chapter 5; Modeling and Simulation for Drug Design, "Computer Modeling of Complex Biological Systems", Iyengar, S.S. Ed. CRC Press, Inc.: 77-90 (1984)

**Ostwald**, T.J., MacLennan, D.H. : Effects of Cation Binding on the Conformation of Calsequestrin and the High Affinity Calcium-binding Protein of Sarcoplasmic Reticulum The Journal of Biological Chemistry Vol. 249, No. 18, Issue of September 25, pp. 5867-5871, 1974

**Ostwald**, T.J., MacLennan, D.H. : Isolation of a High Affinity Calcium-Binding Protein from Sarcoplasmic Reticulum The Journal of Biological Chemistry Vol.249, No. 3, Issue of February 10, pp. 974-979, 1974

**Peachey**, L.D. : The Sarcoplasmic Reticulum and Transverse Tubules of the Frog's Sartorius, Journal of Cell Biology 25:209 June 1965

**Press**, W.H, Flannery, B.P., Teukolsky, S.A., Vetterling, W.T. : Numerical Recipes in C The Art of Scientific Computing, Cambridge University Press: 566-580 (1988)

**Randall**, J. E. : Microcomputers and Physiological Simulation. Addison-Wesley Publishing Company: 57-64 (1980)

**Roeseler**, A. : Program GNRKG (General Numerical Runge-Kutta-Gill) 1984

**Shannon**, R.E. : Systems Simulation - The Art and Science Prentice-Hall, Inc.: 1-35 (1975)

**Somlyo**, A.P., Wasserman, A.J., Kitazawa, T., Bond, M., Shuman, H., Somlyo, A.V. : Calcium and Sodium Distribution and Movements in Smooth Muscle, Experientia 41 (1985)

**Stewart**, P.S., MacLennan, D.H. : Surface Particles of Sarcoplasmic Reticulum Membranes The Journal of Biological Chemistry Vol.249, No. 3, Issue of February 10, pp 985-993, 1974

**Stout**, M.A., Diecke, F.P.J. : Ca Distribution and Transport in Saponin Skinned Vascular Smooth Muscle, The Journal of Pharmacology and Experimental Therapeutics, Vol 225, No 1, 1983

**Tanford**, C. : Twenty Questions Concerning the Reaction Cycle of the Sarcoplasmic Reticulum Calcium Pump, CRC Critical Reviews in Biochemistry Volume 17, Issue 2 1984

**van Breemen**, C., Leuten, P., Yamamoto, H., Aaronson, P., Cauvin, C. : Calcium Activation of Vascular Smooth Muscle, State of the Art Lecture, Suppl II Hypertension, Vol 8, No 6, June 1986

**Vander**, A.J., Sherman, J.H., Luciano, D.S. : Human Physiology: The Mechanisms of Body Function, McGraw-Hill Book Company, 3rd Edition (1980)

The following are the manuals used for code development in FORTRAN for the Hewlet-Packard systems.

FORTRAN 77 Reference Manual RTE-6VM and RTE-A HP1000 Computer System; Part Number 92836-90001; June 1983 E0683

LINK Relocating Loader Manual RTE-A and RTE-6VM; Part Number 92077-90009; January 1983 E0183

Symbolic DEBUG/1000 User's Manual; Part Number 92860-90001; May 1983 E0583

RTE-6VM CI User's Manual; Part Number 92084-90036; December 1983 E1283

The following are the manuals used for code development in C for the UNIX systems.

A Practical Guide to UNIX System V; Sobell; Benjamin/Cummings Publishing Company, Inc. (1985)

Programming in ANSI C; Kochan, S.G., Hayden Books (1988)

The UNIX Programming Environment; Kernighan, B.W., Pike, R., Prentice-Hall (1984)

The C Programming Language; Kernighan, B.W., Ritchie, D.M., Prentice-Hall (1978)

## Software Examples

### Sample Parameter (input) File

```
DIFusion Model OUTF 5b Model_12                                          OUTF5a
60 sec output interval...
 StepChange off, reduced Change Step Interval to 50...
.1e-3  12  2400  60
5
.507419919415497e-10  .253725269576535e-4  .127474624605384e-6  0.0e0  0.0e0

10
5.1e-2            K01c = K10c for Calcium
2.55e-1           K01e = K10e for EGTA
7.5e-2            K12c for Calcium
1.5e-1            K21c for Calcium
3.75e-3           K12e for EGTA
7.5e-3            K21e for EGTA
2.0e+6            Rf for Ca-EGTA binding
0.4e0       Rb for Ca-EGTA binding
1.332e-1    K23c for Calcium
1.998e-5    K32c for Calcium


.5021865e-6   .1660044e-3  .41699785e-3
.5021865e-6   .1660044e-3  .41699785e-3
.5021865e-6                .41699785e-3
.5021865e-6                .41699785e-3
.627733125e-2              .627733125e-2
```

```
outf5b.data
03-19-91     Config Header
1.0e0        Maximum Step Size
1.0e-5            Minimum Step Size
1.0e-2            Error Up Limit
1.0e-3            Error Low Limit
0.1e0        Percent Response
50           Change Step Interval
10           Change Request Stack Size
END OF HEADER
```

## Sample Output (.data) File

```
DIFusion Model OUTF 5b Model_12                                    OUTF5a
60 sec output interval...
StepChange off`, reduced Change Step Interval to 50...

Step Size           : 1.000000e-04    Time Limit      : 2.400000e+03
Number of Equations :         12      Output Interval : 6.000000e+01

There are  5 Forcing Functions : 0

 1>5.074199e-11   2>2.537253e-05   3>1.274746e-07   4>0.000000e+00
5>0.000000e+00


There are 10 Equation Constants

K01c = K10c for Calcium        : 5.100000e-02
K01e = K10e for EGTA           : 2.550000e-01
K12c for Calcium               : 7.500000e-02
K21c for Calcium               : 1.500000e-01
K12e for EGTA                  : 3.750000e-03
K21e for EGTA                  : 7.500000e-03
Rf for Ca-EGTA binding         : 2.000000e+06
Rb for Ca-EGTA binding         : 4.000000e-01
K23c for Calcium               : 1.332000e-01
K32c for Calcium               : 1.998000e-05


Initial Values for Model Equations

 1>5.021865e-07   2>1.660044e-04   3>4.169979e-04   4>5.021865e-07   5>1.660044
 6>4.169979e-04   7>5.021865e-07   8>4.169979e-04   9>5.021865e-07  10>4.169979
11>6.277331e-03  12>6.277331e-03

Runtime Data Output Filename
 outf5b.data

Parameters for the StepChange Algorithm

03-19-91     Config Header
Maximum Step Size              : 1.000000e+00
Minimum Step Size              : 1.000000e-05
Error Up Limit                 : 1.000000e-02
Error Low Limit                : 1.000000e-03
Percent Response               : 1.000000e-01
Change Step Interval           : 50
```

```
Change Request Stack Size      : 10


END OF HEADER
0.000000000e+00 5.021865000e-07 1.660044000e-04 4.169978500e-04 5.021865000e-07
1.660044000e-04
          4.169978500e-04  5.021865000e-07  4.169978500e-04  5.021865000e-07
4.169978500e-04
          6.277331250e-03  6.277331250e-03
          1.000000000e-04   0.000000000e+00      0      0      0  Tue Mar 19
21:59:19 1991
6.000004510e+01 1.777543864e-08 1.012562333e-04 8.841701060e-06 4.342610337e-07
1.250853561e-04
          2.715861846e-04  1.679831581e-08  8.346872860e-06  4.341218599e-07
2.714992198e-04
          6.273570608e-03  6.273570094e-03
          7.325494808e-05   2.168457285e-03     10     12      9  Tue Mar 19
22:01:33 1991
...
...
...
2.489659518e-05
          1.265183151e-05  9.273130659e-10  4.195440202e-07  1.018668117e-07
1.245159560e-05
          6.027041017e-03  6.026602966e-03
          1.531613150e-02   2.761856804e-04    359    277   1566  Tue Mar 19
22:11:13 1991
END OF DATA
Run complete @ Tue Mar 19 22:11:13 1991
END OF STATS
```

## FORTRAN Programs

### Extras.ftn

```
c  EXTRAS.ftn                8-01-88
c
c  use this module for testing the step change algorithm...
c
c   Link to DIFSIM for superior model simulations...
c
c
     SUBROUTINE push_stack( request )
c
c
c  Takes the current REQUEST (integer) and "pushes" it onto the
c  Request_Stack.  This is done to buffer incoming requests in hopes of
c  halting the cycle of increase followed by decrease, when the slope ratio
c  has a problem maintaing the optimum step size.  Decreases are favored
c  over Increases such that an Increase in step will be performed ONLY when
c  the Request_Stack is all increases.  This function is performed by
c  test_stack...

        Integer  stack_size, rqst_stack
         Dimension rqst_stack(20)
         Common /one/ stack_size, rqst_stack, true, false
```

```
        Integer request,iii,true,false

        Do 10 iii=stack_size, 2, -1
10      rqst_stack( iii ) = rqst_stack( iii-1 )
        rqst_stack( 1 ) = request

       Return
      End

c
c=============================================================
c
c
      integer FUNCTION test_stack( request )
c
c
c  This function evaluates the Request_Stack, comparing with the request.
c  If the Stack is homogeneous AND equal to the request (integer) then
c  test_stack remains TRUE.  Otherwise test_stack becomes FALSE...
c

        Dimension rqst_stack(20)
       Common /one/ stack_size, rqst_stack, true, false
       Integer  stack_size, rqst_stack, true, false
       Integer request,iii

      test_stack = 1
       Do 10 iii=1,stack_size
10      if (rqst_stack(iii).ne.request ) test_stack = 0
       return
      end
c
c
c     ====================Step_Change==========================
c
c
      Subroutine step_change
c
c


       Integer  stack_size, rqst_stack, increase, decrease, nt, numb
      Real*8  DT_increases, DT_decreases, DT_passes
       Integer  true, false
          Real*8  DT, min_step, max_step, error_up_limit, error_low_limit
        Real*8  largest_error, response, ak1, ak2, ak3, ak4
        Real*8  term1, term2, error, temp_DT
       Integer i, test_stack
          Dimension ak1(20), ak2(20), ak3(20), ak4(20)
        Dimension  rqst_stack(20)

        Common /one/ stack_size, rqst_stack, true, false
        Common /two/ nt, numb, largest_error, response, increase, decrease
          Common /three/ error_up_limit, error_low_limit, max_step, min_step
      Common /four/ DT, DT_increases, DT_decreases, DT_passes
```

```
      Common /five/ ak1, ak2, ak3, ak4


c
c  Evaluate slope's ratio and change step-size...
c
      nt = 0
       largest_error = 0
       Do 200 i = 1,numb

            term1 = ak3(i) - ak2(i)
            term2 = ak2(i) - ak1(i)

          If ( (term1.eq.0).or.(term2.eq.0) ) Then
            error = 0
       Else
             error = DABS( term1 / term2 )
        Endif

            If (error.GT.largest_error) largest_error = error

200    Continue

          If (largest_error.GT.error_up_limit) Then
          temp_DT = DT - response * DT * 3
          If (temp_dt.GE.min_step) Then
             Call push_stack( decrease )
c
c  NOTE*** the Rqst_Stack does not affect a decrease...always decrease!!!
c
           DT = temp_DT
          DT_decreases = DT_decreases + 1
        Endif
         Else If (largest_error.LT.error_low_limit) Then
           temp_DT = DT + response * DT
              If (temp_DT.LE.max_step) Then
                 Call push_stack( increase )
                  If (test_stack(increase).EQ.true) Then
                    DT = temp_DT
                     DT_increases = DT_increases + 1
                Else
                   DT_passes = DT_passes + 1
                Endif
            Endif
      Endif
       Return
      End
```

Model12.ftn

```
c      MODEL12.FTN       08-14-88
c
c  These equations model isotope exchange.  The third compartment is
c  considered.  Equations 1-6 have been verified with R. Hauser
```

```
c   2-compartment single species model...
c
c
      SUBROUTINE MODEL_TR(x,y,aa,bb)

      REAL*8  X, AA, BB, Y
        DIMENSION  X(20),BB(20),  Y(20),AA(20)
c
c   Identifiers for Forcing Function Array
c
c Ca40       EGTA       Ca40-EGTA     Ca45    Ca45-EGTA

c    Equation Identifiers...
c
c                  1st              2nd               3rd
c                          ---------        -----------      ----------
c  Ca-40            1              4                 11
c  EGTA             2              5
c Ca-40-EGTA       3              6
c  Ca-45            7              9                 12
c Ca-45-EGTA       8             10

          Y(1) = - AA(1)*X(1) - AA(2)*X(1)*X(2) + AA(3)*X(3) +
     $       AA(4)*X(4)
     $      + BB(1)

           Y(2) = - AA(5)*X(2) - AA(2)*(x(1)+x(7))*x(2)+aa(3)*(x(3)+x(8))
     $      +AA(6)*X(5)
     $      + BB(2)

          Y(3) = AA(2)*X(1)*X(2) - AA(7)*X(3) + AA(6)*X(6)
     $      + BB(3)

        y(4) = aa( 8)*X( 1) - aa( 4)*X( 4) - aa( 2)*X( 4)*x( 5) +
     $         aa( 3)*x( 6) - aa(10)*x( 4) + aa(11)*x(11)

          Y(5) = AA(6)*X(2) - AA(8)*X(5) - AA(2)*( x(4) + x(9) )*X(5) +
     $       AA(3)*( x(6) + x(10) )

          Y(6) = AA(6)*X(3) - aa(9)*X(6) + AA(2)*X(4)*X(5)

          Y(7) = - AA(1)*X(7) - AA(2)*X(7)*X(2) + AA(3)*X(8) +
     $       AA(4)*X(9)
     $      + BB(4)

          Y(8) = AA(2)*X(7)*X(2) - AA(7)*X(8) + AA(6)*X(10)
     $      + BB(5)

        y(9) = aa( 8)*X( 7) - aa( 4)*X( 9) - aa( 2)*X( 9)*x( 5) +
     $         aa( 3)*x(10) - aa(10)*x( 9) + aa(11)*x(12)

          Y(10) = AA(6)*X(8) - aa(9)*X(10) + AA(2)*X(9)*X(5)

          y(11) = aa(10)*x(4) - aa(11)*x(11)
```

```
      y(12) = aa(10)*x(9) - aa(11)*x(12)

   RETURN
   END
```

## C Programs

### Main.c

```
/*  Main.c

03-05-91 : MJS

*/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Dimensions.h"
#include "State.h"
#include "Environment.h"
#include "Difsim.h"
#include "Parameters.h"

extern int DumpData ();
extern FinishStamp();
extern GetGillCoefficients();
extern int GetParameterSet(char *filename);
extern InitEnviornment();
extern int InitHeader ();
extern InitState();
extern Model_TR();
extern Problem( char *message );
extern QuitStamp();
extern RungeKutta();
extern StepChange();

char *sys_control;
char *sys_runlist;
char *sys_status;


clock_t start, finish;/*start and finish times for the simulation run*/
clock_t begin, end;           /*begin and end times for the batch run */
FILE *FP_runlist,
       *FP_control,
       *FP_status;            /*file pointers for difsim system files.*/
struct run_param param_set;   /* parameter set ..................... */

int request_stack [ MaxStepRequestStack ];

main()
{
      int nt;           /* this is the NEXT TEST interval
counter..... */
      char runlist[LineWidth];/* array of the filenames
```

```
                                    for parameter sets   */
      int ok;             /* enables a clean shutdown for errors */
      int i;              /* counter.......................... */

      ok = TRUE;
      this_run = 0;

      InitEnviornment();
      InitState();

      GetGillCoefficients();

/* Open System Files...
*/

      if (ok)
            if ( (FP_runlist = fopen( sys_runlist, "r") ) == NULL ) {
                  Problem("Main: Unable to open runlist data file\n");
                  ok = FALSE;
                  }
      if (ok)
            if ( (FP_control = fopen( sys_control, "r") ) == NULL ) {
                  Problem("Main: Unable to open control data file\n");
                  ok = FALSE;
                  }
      if (ok)
            if ( (FP_status = fopen( sys_status, "w") ) == NULL ) {
                  Problem("Main: Unable to open status data file\n");
                  ok = FALSE;
                  }

/******** Main Loop...
*/
      fprintf(FP_status,"Beginning batch run on %s \n",ctime(&begin) );

      while( ok ) {


            if ( !feof( FP_runlist ) )  {
            if ( !(fscanf( FP_runlist, "%[^\n]\n", runlist ) > 0) ) {
                  Problem (
            "Read failed on RUNLIST, possible bad text format...\n" );
                        ok = FALSE;
                        }

                        if (ok) ok = GetParameterSet(runlist);
                        if (ok) ok = InitHeader();
                        next_test = param_set.config.next_test.value;
                        time_step = param_set.step_size;
                        for(i=0;i<param_set.number_of_equations;i++)  {
                              x[i] = param_set.i_v[i];
                              z[i] = param_set.i_v[i];
                              }

                        if (ok) ok = DumpData ();

                        while( (t < param_set.time_limit) && ok )  {
            if ( (next_output = t + param_set.output_interval)
```

```
                                    > param_set.time_limit )
                                    next_output = param_set.time_limit;
                                while (ok && t < next_output )  {
                                  if ( ( t + (next_test * time_step) )
                                                <= next_output )
                                    for (nt = 0; nt < next_test; nt++)
                                                RungeKutta ();
                                    else
                                          while ( t < next_output )
                                                RungeKutta ();

                                    StepChange ();
                                    }
                                if (ok) ok = DumpData ();
                                }
                          if (ok) ok = FinishStamp();
                          InitState ();
                          this_run++;
                    }
              else ok = FALSE; /* no more param sets to run */
              }
              QuitStamp (FP_status);
              fclose (FP_control);
}
```

**RungeKutta.c**

```
/*  RungeKutta.c

03-05-91 : MJS

*/

#ifndef SUCCESS
#include "Dimensions.h"
#include "Parameters.h"
#endif

extern double t;
extern double time_step;

extern double x[ MaxEqns ];
extern double y[ MaxEqns ];
extern double z[ MaxEqns ];

extern double ak1[ MaxEqns ];
extern double ak2[ MaxEqns ];
extern double ak3[ MaxEqns ];
extern double ak4[ MaxEqns ];

extern double a;
extern double b;
extern double c;
extern double d;

extern struct run_param param_set;


/* Side effects...
```

```
      - this routine increments the value of t
      - this routine modifies the value of z[]

*/
void RungeKutta ( void )
{
      int i;

      Model_TR();
      for(i=0;i<param_set.number_of_equations;i++)  {
            ak1[i] = y[i];
            x[i] = z[i] + time_step*ak1[i]/2.0;
            }
      t += (time_step/2.0);

      Model_TR();
      for(i=0;i<param_set.number_of_equations;i++)  {
            ak2[i] = y[i];
            x[i] = z[i] + time_step*(c*ak1[i] + a*ak2[i]);
            }

      Model_TR();
      for(i=0;i<param_set.number_of_equations;i++)  {
            ak3[i] = y[i];
            x[i] = z[i] + time_step*(d*ak2[i] + b*ak3[i]);
            }
      t += (time_step/2.0);

      Model_TR();
      for(i=0;i<param_set.number_of_equations;i++)  {
            ak4[i] = y[i];
            x[i] = z[i]+ time_step*(ak1[i]+ 2.0*a*ak2[i]+ 2.0*b*ak3[i]
                        + ak4[i])/6.0;
            z[i] = x[i];
            }
}
```

**StepChange.c**

```
/* StepChange.c

03-05-91 : MJS

*/

#include <math.h>

#ifndef SUCCESS
#include "Dimensions.h"
#include "Parameters.h"
#endif

void StepChange (void);
void PushStack ( int request );
int TestStack ( int request );
double GetLargestError ( void );

extern struct run_param param_set;
extern int request_stack [MaxStepRequestStack];
```

```
void StepChange (void)

{
      extern double time_step;
      extern int dt_decreases;
      extern int dt_increases;
      extern int dt_passes;
      extern double largest_error;

      double error;
      double temp_dt;

      error = GetLargestError ();

      if (error > param_set.config.error_up_limit.value)  {
            temp_dt = time_step -
param_set.config.response.value*time_step;
            if (temp_dt >= param_set.config.min_step.value)  {
                  PushStack (DECREASE);
                  time_step = temp_dt;
                  dt_decreases += 1;
                  }
            }
      else  {
            if (error < param_set.config.error_low_limit.value)  {
                  temp_dt = time_step +
                        param_set.config.response.value*time_step;
                  if (temp_dt <= param_set.config.max_step.value)  {
                        PushStack (INCREASE );
                        if ( TestStack (INCREASE) == TRUE )  {
                              time_step = temp_dt;
                              dt_increases += 1;
                              }
                        else
                              dt_passes += 1;
                  }
            }
      }
      largest_error = error;
}

void PushStack ( int request )

/*
Takes the current REQUEST and "pushes" it onto the request_stack.
This is done to buffer incoming requests so that a frequent cycle
of increase followed by decrease followed by increase, can be avoided.
This can occur when the slope_ratio has a problem maintaining the
optimum step size.  Decreases are fovored over Increases such that an
Increase in step will be performed only when the request_stack is all
Increases.

The testing of the request_stack is performed by TestStack...
*/

{
      int i;
```

```
      for(i= param_set.config.stack_size.value-1; i>0;--i)
            request_stack[i] = request_stack[i-1];

      request_stack[0] = request;
}

int TestStack ( int request )

/*
This function evaluates the request_stack, comparing with the current
request.  If the stack is homogeneous AND equal to the request, then
TestStack returns TRUE.  Otherwise, TestStack returns false.
*/

{
      int i;

      for(i=0;i<param_set.config.stack_size.value;i++)
            if (request_stack[i] != request)
                  return (FALSE);

      return (TRUE);
}

double GetLargestError ( void )

{
      extern double ak1[ MaxEqns ];
      extern double ak2[ MaxEqns ];
      extern double ak3[ MaxEqns ];
      extern double ak4[ MaxEqns ];

      double largest_error;
      int i;
      double error;
      double term1, term2;

      largest_error = 0.0;
      for(i=0;i<param_set.number_of_equations;i++)  {
            term1 = ak3[i] - ak2[i];
            term2 = ak2[i] - ak1[i];

            if ( (term1 == 0) || (term2 == 0) )
                  error = 0;
            else
                  error = fabs ( term1 / term2 );

            if (error > largest_error) largest_error = error;
            }
      return (largest_error);
}
```

**Model_12.c**

```
/*    MODEL_12.c

03-10-91 : MJS

*/
```

```
#ifndef SUCCESS
#include "Dimensions.h"
#include "Parameters.h"
#endif

/*  <C>oefficients  */

#define K01c            param_set.ec[ 0].value /* k01c = k10c        */
#define K10c            param_set.ec[ 0].value
#define K01e            param_set.ec[ 1].value /* k01e = k10e        */
#define K10e            param_set.ec[ 1].value
#define K12c            param_set.ec[ 2].value /*                   */
#define K21c            param_set.ec[ 3].value /*                   */
#define K12e            param_set.ec[ 4].value /*                   */
#define K21e            param_set.ec[ 5].value /*                   */
#define Rf              param_set.ec[ 6].value /* Ca-EGTA binding    */
#define Rb              param_set.ec[ 7].value /* Ca to EGTA binding */
#define K23c            param_set.ec[ 8].value /*                   */
#define K32c            param_set.ec[ 9].value /*                   */

/*  Variables and Equations  */

#define Ca40_1                  0
#define Egta_1                  1
#define Ca40_E_1                2     /* E:EGTA */
#define Ca40_2                  3
#define Egta_2                  4
#define Ca40_E_2                5     /* E:EGTA */
#define Ca45_1                  6
#define Ca45_E_1                7     /* E:EGTA */
#define Ca45_2                  8
#define Ca45_E_2                9     /* E:EGTA */
#define Ca40_3                  10
#define Ca45_3                  11

/*  <F>orcing <F>unctions  */

#define Ca40_res        param_set.ff[ 0]
#define Egta_res        param_set.ff[ 1]
#define Ca40_E_res      param_set.ff[ 2]
#define Ca45_res        param_set.ff[ 3]
#define Ca45_E_res      param_set.ff[ 4]

extern double x[MaxEqns];
extern double y[MaxEqns];

extern struct run_param param_set;

void Model_TR(void)
{
y[ Ca40_1 ] =     - K10c * x[Ca40_1]
                  - K12c * x[Ca40_1]
                  - Rf   * x[Ca40_1]*x[Egta_1]
                  + Rb   * x[Ca40_E_1]
                  + K21c * x[Ca40_2]
                  + Ca40_res;
```

```
y[ Egta_1 ] =      - K10e * x[Egta_1]
                   - K12e * x[Egta_1]
                   - Rf   * (x[Ca40_1] + x[Ca45_1])*x[Egta_1]
                   + Rb   * (x[Ca40_E_1] + x[Ca45_E_1])
                   + K21e * x[Egta_2]
                   + Egta_res;

y[ Ca40_E_1 ] = - K10e * x[Ca40_E_1]
                   - K12e * x[Ca40_E_1]
                   - Rb   * x[Ca40_E_1]
                   + Rf   * x[Ca40_1]*x[Egta_1]
                   + K21e * x[Ca40_E_2]
                   + Ca40_E_res;

y[ Ca40_2 ] =      - K21c * x[Ca40_2]
                   - K23c * x[Ca40_2]
                   - Rf   * x[Ca40_2]*x[Egta_2]
                   + Rb   * x[Ca40_E_2]
                   + K12c * x[Ca40_1]
                   + K32c * x[Ca40_3];

y[ Egta_2 ] =      - K21e * x[Egta_2]
                   - Rf   * (x[Ca40_2] + x[Ca45_2])*x[Egta_2]
                   + Rb   * (x[Ca40_E_2] + x[Ca45_E_2])
                   + K12e * x[Egta_1];

y[ Ca40_E_2 ] = - K21e * x[Ca40_E_2]
                   - Rb   * x[Ca40_E_2]
                   + Rf   * x[Ca40_2]*x[Egta_2]
                   + K12e * x[Ca40_E_1];

y[ Ca45_1 ] =      - K10c * x[Ca45_1]
                   - K12c * x[Ca45_1]
                   - Rf   * x[Ca45_1]*x[Egta_1]
                   + Rb   * x[Ca45_E_1]
                   + K21c * x[Ca45_2]
                   + Ca45_res;

y[ Ca45_E_1 ] =  - K10e * x[Ca45_E_1]
                   - K12e * x[Ca45_E_1]
                   - Rb   * x[Ca45_E_1]
                   + Rf   * x[Ca45_1]*x[Egta_1]
                   + K21e * x[Ca45_E_2]
                   + Ca45_E_res;

y[ Ca45_2 ] =      - K21c * x[Ca45_2]
                   - K23c * x[Ca45_2]
                   - Rf   * x[Ca45_2]*x[Egta_2]
                   + Rb   * x[Ca45_E_2]
                   + K12c * x[Ca45_1]
                   + K32c * x[Ca45_3];

y[ Ca45_E_2 ] = - K21e * x[Ca45_E_2]
                   - Rb   * x[Ca45_E_2]
                   + Rf   * x[Ca45_2]*x[Egta_2]
                   + K12e * x[Ca45_E_1];

y[ Ca40_3 ] = - K32c * x[Ca40_3] + K23c * x[Ca40_2];
```

```
y[ Ca45_3 ] = - K32c * x[Ca45_3] + K23c * x[Ca45_2];
}
```