Investigation and implementations of efficient algorithms for multiscale co-simulation in Neuroscience

JERKER NILSSON

Master’s Thesis in Scientific Computing (30 ECTS credits)
Master Programme in Mathematics (120 credits)
Royal Institute of Technology year 2014
Supervisors at KTH were Michael Hanke and Ekaterina Brocke
Examiner was Michael Hanke

TRITA-MAT-E 2014:42
ISRN-KTH/MAT/E--14/42--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci
Abstract

The function of the nervous system consists of a multitude of different processes and phenomena. These occur on different spacial and temporal scales and are described by different physical formalisms. Most models are limited to one scale and one type of physical or mathematical description.

In this thesis, an effective method for co-simulating such models was implemented. The method is an adaptive step size controller combining Backward Differentiation Formulas with a PI-controller. It was tested on a simple problem, coupling an electric model of a neuron with chemical reactions, with satisfying results.
Referat

En studie i effektiva algoritmer för flerskaliga kopplade simuleringar i neurovetenskap


Acknowledgments

A numerous of people have made this thesis possible. First, I like to thank my supervisors Michael Hanke and Ekaterina Brocke. I also want to give a thank to Mikael Djurfeldt. Thanks for all support and help along the way.

I would like to thank all my fellow students at Engineering Physics here at KTH, as well as all members of Fysiksektion. You have made the last five years fly by. An extra thank you to Jonas Adler and Erik Boström. Without our collaboration, the two last years would have been much harder.

Last, but certainly not least, I want to thank my dear family. Thanks for always being there for me and recognizing my need to study. An extra special thanks to Ksenia Chechet, who has been the ultimate support.

Many thanks,
Jerker
Contents

1 Introduction 1
   1.1 Background ................................................. 1
   1.2 Aim of the project ........................................ 1
   1.3 Scope and limitations ...................................... 2

2 Theory 3
   2.1 A brief introduction of neurons ............................. 3
   2.2 Modeling equations .......................................... 4
      2.2.1 Hodgkin and Huxley formalism ......................... 4
      2.2.2 Cable equations ....................................... 6
      2.2.3 Compartmental modeling ................................ 7
      2.2.4 Chemical kinetics ..................................... 7
   2.3 Numerical schemes .......................................... 7
      2.3.1 Runge-Kutta method .................................. 8
      2.3.2 Hines method ......................................... 9
      2.3.3 Backward Differentiation Formula .................... 9
   2.4 Decoupled systems .......................................... 11
      2.4.1 Organizations ......................................... 11
   2.5 Adaptive step size controller ............................... 13
      2.5.1 Error estimation ........................................ 14
      2.5.2 PI controller .......................................... 15

3 Method and Implementation 17
   3.1 Modeling the system ........................................ 17
      3.1.1 Electric system ....................................... 18
      3.1.2 Chemical system ...................................... 19
      3.1.3 Dynamics ............................................... 21
      3.1.4 System interaction .................................... 21
   3.2 Implementation ............................................ 21
      3.2.1 Adaptive solver for the Runge-Kutta and Hines methods 23
   3.3 How to compare solutions ................................... 23
      3.3.1 Fourier analysis ....................................... 24
      3.3.2 Solver efficiency ...................................... 24
Chapter 1

Introduction

1.1 Background

In many disciplines, problems arise that need mathematical modeling of their solutions. The solutions are rarely known in analytic form and must be approximated using numerical methods. Advances of computational technology make it possible to model more and more complex details of real world phenomena, making the simulation of models an alternative to experiments. These complex models often encompass multiple temporal and spatial scales described by different physical and mathematical formalisms that make them non-trivial to solve.

A brain is a multiscale phenomenon. We employ different levels of physical implementations to study the brain; from molecules on a sub-cellular level, to a single neuron and population of those on a network and a system level [1]. Often, only the levels of interest are taken into account when answering a certain scientific question, but all levels are naturally emerged and influences each other.

For instance, synaptic plasticity is a particular question that can be addressed with multiscale modeling in Neuroscience. However, many studies addressed this question on different levels of physical organization and some looked at interactions between these levels as well [2]. Unfortunately, technical difficulties such as absence of efficient coupling algorithms, lack of interoperability between the simulators [3] restrict the community from further investigation of multiscale phenomena in Neuroscience.

1.2 Aim of the project

The main objective of this project is to develop an efficient and reliable method for integrating neural models that span multiple physical and/or chemical processes, and act on different temporal and spatial scales.

This includes to

1. study and implement integration methods applicable and generally usable in Neuroscience.
2. investigate different organizations for coupling given systems.

3. learn local error estimation techniques and error control mechanisms.

4. propose and implement an efficient step size controller.

5. perform analyses for a given example implemented in MATLAB constructed by Ekaterina Brocke.

1.3 Scope and limitations

To limit the project, certain decisions were made. First, common methods for modeling these kind of systems were used. The numerical methods for simulations were chosen either because of their common usage, or for that they fitted a certain framework.

The example problem is not intended to be compared to analytic or laboratory results. The intention of it is to provide a system that is similar to systems arising in Neuroscience, so that the implemented method can be tested, not to simulate a real problem.
Chapter 2

Theory

The theory chapter is organized as follows. First, a brief introduction to neurons are presented. The meaning of this is not to introduce any difficult theories, rather just freshen up the biological knowledge of the reader. Secondly, the two models that are to be coupled are explained. After this, different methods for solving the individual systems are described as well as a short introduction in how to couple models. Finally, the theory behind an adaptive step size solver, that will later be implemented, is explained.

2.1 A brief introduction of neurons

The nervous system in animals and humans is responsible for the coordination of voluntary and involuntary actions of the body, as well as transmitting signals within it. The nervous system is an immense system partly consisting of nerve cells, or neurons. A neuron is a cell that transmit information via electrical or chemical signals and one of the oldest of all specialized cell types among animals [4]. The neuron is a type of cell known as an excitable cell. An excitable cell reacts to stimuli and changes its membrane characteristics accordingly [5]. The human nervous system consists of more than $10^{10}$ neurons.

A neuron consists of a soma (cell body), dendrites and an axon. A sketch of a neuron is shown in Figure 2.1 [6]. The soma is the end part of the cell and contains the cell nucleus, which synthesizes most of the protein made in the cell. The size of the soma can vary from 4 to $100 \mu m$. The dendrites are responsible for receiving signals from other cells via the dendritic spine. The dendrites form an intricate structure, often referred to as a dendritic tree. The dendrites are an average of $100 \mu m$ long and there are approximately 200 000 spines per cell.

While the dendrites are a neuron’s way of receiving signals, the axon is responsible for transmitting information to different neurons, muscles and glands. It has a length that varies between a few mm to a couple of meters, depending on nerve type and which animal it comes from. The axon of one cell connects with the dendrites of another cell, via the chemical synapse. The two cells are electrically isolated from each other by the synaptic cleft. The mechanism of transmitting over the synaptic cleft is usually indirect.
A change of the electric potential in the pre-synaptic cell triggers a release of small signal molecules known as neurotransmitters. The neurotransmitters diffuse over the synaptic cleft and binds to transmitter-gated ion channels in the post-synaptic cell. This causes an electric change in the cell which opens the channels, and the signal can continue.

2.2 Modeling equations

The aim of this thesis is to construct a method for co-simulation of models that occur in neuroscience. In particular, we will model the interactions of electric signals in a neuron with chemical reactions at a sub-cellular level. The electric dynamics of the neuron are often expressed by using the Hodgkin-Huxley formalism, which together with the cable equation form a mathematical system. The chemical reactions on the other hand, are described by chemical kinetics and rate equations which also result in a system of equations. These systems will later be coupled to each other and thus form a single system of equations.

2.2.1 Hodgkin and Huxley formalism

The electric behavior of neurons can be expressed by the so-called Hodgkin and Huxley formalism [7]. It was formulated by Alan Hodgkin and Andrew Huxley in 1952 and is commonly used still today. In the formalism, the membrane can be represented as an electric circuit, as is shown in Figure 2.2. Here, the cell membrane is modeled by the capacitance $C_M$ and the flow of ions by currents.

In this model, the ionic current is divided between the two most prominent ions, sodium ($I_{Na}$) and potassium ($I_K$), and a leakage current ($I_l$) corresponding to chloride and other ions. Each component of the ionic current can be modeled as a relation between an electric potential difference $E$ and a permeability coefficient $g$ with dimension
2.2. MODELING EQUATIONS

Figure 2.2. The membrane of a cell modeled as an electric circuit.

of conductance. The equations for the individual currents can be expressed as

\[ I_{Na} = g_{Na}(E - E_{Na}) \]  \hspace{1cm} (2.1a)
\[ I_{K} = g_{K}(E - E_{K}) \]  \hspace{1cm} (2.1b)
\[ I_{l} = g_{l}(E - E_{l}) \]  \hspace{1cm} (2.1c)

where \( E_{Na} \) and \( E_{K} \) are the equilibrium potentials for respective ion. \( E_{l} \) is the potential at which the leakage current is zero. Instead of working with the equilibrium potentials \( E \), it is more convenient to use the displacement from the resting potential \( E_{r} \), i.e. perform the variable substitution

\[ V = E - E_{r} \]
\[ V_{Na} = E_{Na} - E_{r} \]
\[ V_{K} = E_{K} - E_{r} \]
\[ V_{l} = E_{l} - E_{r} \]

This transforms the equations to

\[ I_{Na} = g_{Na}(V - V_{Na}) \]  \hspace{1cm} (2.2a)
\[ I_{K} = g_{K}(V - V_{K}) \]  \hspace{1cm} (2.2b)
\[ I_{l} = g_{l}(V - V_{l}) \]  \hspace{1cm} (2.2c)

To model the total membrane current, the Hodgkin and Huxley formalism divide the total membrane current into a current caused by the capacity and a current caused by the ions. Thus, the membrane potential can be expressed as

\[ C_{m} \frac{dV_{m}}{dt} = I_{Na} + I_{K} + I_{l} + I_{inj} \]  \hspace{1cm} (2.3)

where \( I_{inj} \) is an external stimuli, \( V_{m} \) the displacement of membrane potential and \( C_{m} \) the capacity per unit area.
The voltage dependent conductance $g$ are dependent of transfer probabilities through the membrane. The probability for crossing the membrane can be expressed by the ordinary differential equation

$$\frac{dp}{dt} = \alpha_p (1 - p) - \beta_p p$$

(2.4)

where $p \in n, m, h$ and $\alpha_p$ and $\beta_p$ are rate constants dependent of voltage, but not of time. $p$ is a dimensionless variable on the interval $[0, 1]$

The potassium conductance $g_K$ and the sodium conductance $g_{Na}$ can be described by the equations

$$g_K = \bar{g}_K n^4$$

(2.5a)

$$g_{Na} = m^3 h \bar{g}_{Na}$$

(2.5b)

where $\bar{g}_K$ and $\bar{g}_{Na}$ are constants with dimension conductance/area. The exponents are related to how many activation gates each channel has of each type. For example, each $K$-channel has four identical $n$-gates.

2.2.2 Cable equations

The conduction of a current along a dendrite can be compared to that of the current in an electric cable. If the cable is completely isolated, the change of current along the cable can be deduced from Ohm’s law as

$$\frac{1}{R_i} \frac{\partial V}{\partial x} = -I_l$$

(2.6)

where $R_i$ is the longitudinal intracellular resistance per unit length, $x$ the positive direction of the cable and $I_l$ the current along the cable.

If the cable isn’t isolated, some current will "escape". In the dendrite, this corresponds to a current through the membrane walls. Thus, the change of current along the cable due to the leakage can be expressed as

$$\frac{\partial I_l}{\partial x} = -I_m$$

(2.7)

where $I_m$ is the current through the membrane. The leakage can either be of the form of a current through the membrane, or by a change of capacitance. Thus, we can infer that

$$I_m = \frac{V}{R_m} + C_m \frac{\partial V}{\partial t}$$

(2.8)

Combining these three equations, give the cable equation

$$\frac{1}{R_i} \frac{\partial^2 V}{\partial x^2} = C_m \frac{\partial V}{\partial t} + \frac{V}{R_m}$$

(2.9)
2.3. NUMERICAL SCHEMES

2.2.3 Compartmental modeling

Combining the Hodgkin-Huxley formalism with the cable equation gives a well used model of a neuron. The space derivative of the cable equation is discretized along its length into different subcompartments. If the discretization is small enough, each compartment can be assumed isopotential and the current in Equation 2.9 can be replaced with the currents expressed in the Hodgkin-Huxley formalism. A second order central difference approximation of the spatial derivative gives

\[
\frac{V_{m}^{i+1} - 2V_{m}^{i} + V_{m}^{i-1}}{R_{i}} = -I_{HH} + C_{m} \frac{dV_{m}^{i}}{dt}
\]  

(2.10)

where \(V_{m}^{i}\) is the membrane potential of compartment \(i\) and \(I_{HH}\) the summed Hodgkin-Huxley currents for that compartment.

Together with the gate-equations in Equation 2.4, this forms a system of equations that models the electric behavior of the neuron.

2.2.4 Chemical kinetics

Signaling pathways are chemical reactions at a sub-cellular level [8]. To model these, a series of reaction rate equations are formulated. A simple rate equation has the form

\[
A + B \xrightleftharpoons{k_f}{k_b} C + D
\]  

(2.11)

For substrate A, this can be described by the differential equation

\[
\frac{d[A]}{dt} = k_b[C][D] - k_f[A][B]
\]  

(2.12)

where \([X]\) correspond to the concentration of molecule \(X\) and \(k_b\) and \(k_f\) are rate constants. Provided initial conditions for the molecule concentrations the laws of mass concentration constrain the remaining concentrations once the current concentration for molecule A is known.

A more complex case is the interaction with an enzyme catalyst. A substrate reacts with the help of an enzyme and forms a product. The equation for this reaction can be formulated as

\[
\text{Enz} + \text{Sub} \xrightleftharpoons{k_1}{k_2} \text{EnzSub} \xrightarrow{k_3} \text{Enz} + \text{Prd}
\]  

(2.13)

In Section 3.1.2 a model of a signaling pathway will be explained.

2.3 Numerical schemes

The common way to solve the problems formulated in previous section is to use a Runge-Kutta solver for the chemical reactions and a solver called Hines method for the electric system. In this thesis, I will introduce an adaptive step size controller for co-simulation.
This method uses a Backward Differential Formula for each subsystem. The individual solvers are presented in the following sections.

What these methods have in common, is that they aim so solve the initial value problem (IVP)

$$y' = f(x, y), \quad y(x_0) = y_0$$  \hspace{1cm} (2.14)

One of the easiest way to approximate the solution of the IVP is to use the Explicit Euler method, which approximates the solution as

$$y_{n+1} \approx y_n + hf(t_n, y_n)$$  \hspace{1cm} (2.15)

Here, $y_n$ is the approximated solution at time point $n$, $h$ the time step and $t_n = hn$ the $n$:th time. Explicit Euler is a first order method.

### 2.3.1 Runge-Kutta method

If the right hand side of Equation 2.14 would be independent of $y$, the IVP would have the exact solution

$$y(X) = y_0 + \int_{x_0}^{X} f(x) \, dx$$  \hspace{1cm} (2.16)

The integral can for example be approximated by the midpoint rule

$$y(X) \approx y_{n-1} + h_{n-1}f \left(X_{n-1} + \frac{h_{n-1}}{2}\right)$$  \hspace{1cm} (2.17)

which is a second order approximation. Here, $h_i = x_{i+1} - x_i$ and $x_0, x_1, ..., x_n = X$ is a subdivision of the integrated interval.

The increased order of accuracy inspired Runge (1895) to try to extend this method to approximate the original problem in Equation 2.14.

The first step of this new method, would be expressed as

$$y(x_0 + h) \approx y_0 + hf \left(x_0 + \frac{h}{2}, y \left(x_0 + \frac{h}{2}\right)\right)$$  \hspace{1cm} (2.18)

The question now is how to approximate $y \left(x_0 + \frac{h}{2}\right)$. Again, the Explicit Euler is the easiest approach, which results in the approximation

$$k_1 = f(x_0, y_0)$$

$$y_1 = y_0 + hf \left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right)$$  \hspace{1cm} (2.19)

This can be shown to be a second order approximation, i.e. a method of higher order than the Explicit Euler.

In 1901, Kutta formulated the general scheme of the above approach, and the methods are thus called Runge-Kutta methods. The methods are named by their order of accuracy, for example the approximation in Equation 2.19 is called Runge-Kutta method.
2.3. NUMERICAL SCHEMES

of order 2, or simply RK2 after its inventors. The Explicit Euler method is exactly the
same method as RK1.

One of the Runge-Kutta methods has been more popular than others. It is the fourth
order Runge-Kutta method presented in Equation 2.20, and is often referred to as The
Runge-Kutta method. It can be expressed as

\[ y_{n+1} \approx y_n + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) \]  (2.20)

where

\[ k_1 = f(t_n, y_n) \]
\[ k_2 = f \left( t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1 \right) \]
\[ k_3 = f \left( t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2 \right) \]
\[ k_4 = f \left( t_n + h, y_n + hk_3 \right) \]  (2.21)

2.3.2 Hines method

In 1983, Michael Hines formulated the so-called Hines method for solving branched nerve
equations [9]. The method is often used to solve the system of ODEs described in Section
2.2.3. Due to the difference in sizes between the spine and the other parts of the neuron,
the system of ODEs will be very stiff, and thus an implicit method is needed, which Hines
method is. To get full second order accuracy, the potentials \( V \), governed by Equation
2.9 and the gates \( p \), governed by Equation 2.4 are solved on a staggered grid with half a
time step difference. This also makes the system partly linearized and removes the need
of the expensive iterative approximation that an implicit method often requires.

The small size of the spine compared to the soma and dendrite, makes the electrical
model very stiff. The stiff system requires an implicit numerical method for simulation.

For a system with only one compartment and one gate, the Hines method would
have the form [10]

\[ p_{n+1/2} = p_{n-1/2} + h \left( \alpha(V_n) - \beta(V_n) \frac{p_{n+1/2} + p_{n-1/2}}{2} \right) \]  (2.22)
\[ V_{n+1} = V_n + \frac{h}{C} \left( \frac{2E - V_n - V_{n+1}}{2} \frac{g(p_{n+1/2})} \right) \]  (2.23)

2.3.3 Backward Differentiation Formula

As shown above, Equation 2.14 is often solved in integral form, i.e. finding solutions to
the equation

\[ y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(t, y(t)) dt \]  (2.24)

Instead of this integral approach, one could use methods based on differentiation, so
called BDF methods. These methods were introduced by C. Curtiss and J. Hirschfelder
Assume that there exist approximations to the ODE in Equation 2.14, \( y_{n-k+1}, \ldots, y_n \) for previous time steps. Using these, we can construct an interpolation polynomial \( q(x) \) as

\[
q(x) = q(x_n + sh) = \sum_{j=0}^{k} (-1)^j \binom{-s + 1}{j} \nabla^j y_{n+1}
\]  \hspace{1cm} (2.25)

The unknown value \( y_{n+1} \) will be determined such that the polynomial \( q(x) \) satisfy the differential equation in at least one grid point, i.e.

\[
q'(x_{n+1-r}) = f(x_{n+1-r}, y_{n+1-r})
\]  \hspace{1cm} (2.26)

The choice \( r = 1 \), gives after some manipulations an implicit formulation on the form

\[
\sum_{j=1}^{k} \frac{1}{j} \nabla^j y_{n+1} = hf_{n+1}
\]  \hspace{1cm} (2.27)

for a uniform distributed grid with step size \( h \). The number \( k \) determines how many points the interpolation should contain.

In this thesis, we are mainly interested in the two-step backward differentiation formula, called BDF2. This is due to the stability properties of the BDF2 method, as well as the known error results for second order predictors introduced by Skelboe [12].

On an equidistributed grid, this formula can be expressed as

\[
\frac{2}{3} y_{n+1} - 2y_n + \frac{1}{2} y_{n-1} = hf_{n+1}
\]  \hspace{1cm} (2.28)

On a non-uniform grid on the other hand, the BDF2 method can be written as [12]

\[
y_{n+1} = \alpha_1 y_n + \alpha_2 y_{n-1} + \beta h_{n+1} f(t_{n+1}, y_{n+1})
\]  \hspace{1cm} (2.29)

where

\[
\gamma_{n+1} = h_{n+1}/h_n
\]
\[
\alpha_1 = 1 - \alpha_2
\]
\[
\alpha_2 = -\gamma^2_{n+1}/(2\gamma_{n+1} + 1)
\]
\[
\beta = (\gamma_{n+1} + 1)/(2\gamma_{n+1} + 1)
\]

On an uniform grid, the method is A-stable and has a stability domain shown in Figure 2.3. It can analytically be shown that the root locus curve \( \mu \) has no roots on the left hand part of the complex plane. The root locus curve is given by

\[
\mu = (1 - \exp(-i\theta)) + \frac{1}{2}(1 - \exp(-i\theta)^2)
\]  \hspace{1cm} (2.30)

This has the real part of

\[
Re(\mu) = \frac{3}{2} - 2\cos(\theta) + \frac{1}{2}\cos(2\theta) \geq 0
\]  \hspace{1cm} (2.31)

and the A-stability is given.
2.4. DECOUPLED SYSTEMS

Figure 2.3. The stability domain of the BDF2 method lies outside the plotted curve.

2.4 Decoupled systems

In [12], Skelboe explains how systems of ODEs can be decoupled into loosely coupled subsystems. In this thesis, the approach is the opposite. Rather than decoupling systems, we want to connect them to each other, to successfully co-simulate the two models. How this was implemented in practice will be explained in Section 3.1.4. In short, we will have a coupled system on the form

\[
\begin{align*}
y'_{\text{chem}} &= f_{\text{chem}}(t, y_{\text{chem}}, y_{\text{elec}}) \\
y'_{\text{elec}} &= f_{\text{elec}}(t, y_{\text{chem}}, y_{\text{elec}})
\end{align*}
\] (2.32)

2.4.1 Organizations

When systems are coupled, a problem arises. The solution of one subsystem will depend on the solution of the other. Thus, it is of importance in which order the subsystems are solved, especially when implicit solvers are used which is the case for the electric subsystem. It is also of importance how the other systems are approximated.

Jacobi organization

With the Jacobi organization, each subsystem approximates the solutions of the others systems at the next time step, and uses that approximation to calculate the next solution of its own system. A schematic sketch of this can be seen in Figure 2.4. For the BDF2 method mentioned in the previous section, this organization gives the update formula

\[
\begin{align*}
y_{\text{chem}, n+1} &= \alpha_1 y_{\text{chem}, n} + \alpha_2 y_{\text{chem}, n-1} + \beta h_{n+1} f_{\text{chem}}(t_{n+1}, y_{\text{chem}, n+1}, \tilde{y}_{\text{elec}, n+1}) \\
y_{\text{elec}, n+1} &= \alpha_1 y_{\text{elec}, n} + \alpha_2 y_{\text{elec}, n-1} + \beta h_{n+1} f_{\text{elec}}(t_{n+1}, \tilde{y}_{\text{chem}, n+1}, y_{\text{elec}, n+1})
\end{align*}
\] (2.33)

with the same coefficients as before, and \( y_{n+1} \) is an approximation of \( y \) at time \( t_{n+1} \).

This organization works very well in a parallel computations, since no system needs to wait for the other.
Gauss-Seidel organization

The Gauss-Seidel organization updates the subsystem in a sequential order. Thus it requires less approximations than the Jacobi organization, but is not as easily parallelized. An schematic sketch of this organization is shown in Figure 2.5. For the BDF2 method mentioned in the previous section, this organization gives the update formula

\[
y_{\text{chem}, n+1} = \alpha_1 y_{\text{chem}, n} + \alpha_2 y_{\text{chem}, n-1} + \beta h_{n+1} f_{\text{chem}}(t_{n+1}, y_{\text{chem}, n+1}, \tilde{y}_{\text{elec}, n+1}) \\
y_{\text{elec}, n+1} = \alpha_1 y_{\text{elec}, n} + \alpha_2 y_{\text{elec}, n-1} + \beta h_{n+1} f_{\text{elec}}(t_{n+1}, y_{\text{chem}, n+1}, y_{\text{elec}, n+1})
\]  

(2.34)

with the same coefficients as before, and \(n+1\) is an approximation of \(y\) at time \(t_{n+1}\). Note especially that in this case, the second equation does not utilize the approximation.

Another aspect of the Gauss-Seidel organization is in what order the subsystems will be solved. For the problem in this thesis, this corresponds to solving the chemical or electric subsystem first. For the system shown above, the chemical system is taken first.
2.5 Adaptive step size controller

The goal of this thesis is to implement an adaptive step size controller and combine it with the BDF2 methods for solving a system on the form of Equation 2.32. The adaptive controller is described by Deuflhard [13].

To ensure the required precision, one would ideally have control over the global error. This is not achievable, but instead we can control the local discretization error, $||\epsilon_{j+1}||$. Unfortunately, we can not calculate this error exactly, and have to settle with a computable approximation $|\epsilon_j|$, and thus require

$$|\epsilon_j| < TOL$$  \hfill (2.35)

Here, $TOL$ is a tolerance defined by the user. The error estimate $|\epsilon_j|$ is a measure of the quality of the taken step. Using this, we want to calculate the optimal value for the next step. This is not doable, since it requires knowledge of the future that we do not have. Instead, we calculate the optimal current step size $\tau^*_j$. If the calculated quality of the current step is good enough, we continue to the next step with $\tau^*_j$ as a prediction of the next step size. If the quality is not good enough, we use the now calculated optimal step size $\tau^*_j$ as the current step size and redo this step. This leads to the algorithm shown in Algorithm 1. Of special interest is line number 9. This statement consists of two important parts. First, it gives an approximation of $\tau^*_j$ as

$$\tau^*_j = p^{\frac{1}{2}} \frac{\rho \cdot TOL}{||\epsilon_j+1||} \tau_j$$ \hfill (2.36)

This is based on the following;

For the "optimal" step $\tau^*_j$, we require that the corresponding error, $|\epsilon^*_j|$ is close to the tolerance level $TOL$. If it is to far below, the algorithm won’t be efficient and if it is much larger, the results won’t be dependable. Thus, we want

$$|\epsilon^*_j| \approx TOL$$ \hfill (2.37)

Since

$$||\epsilon_{j+1}|| \approx |\epsilon_{j+1}| = c(t_j)\tau_j^{p+1} + \Omega(\tau_j^{p+2}) \approx c(t_j)\tau_j^{p+1}$$ \hfill (2.38)

and

$$TOL \approx |\epsilon^*_j| \approx c(t_j)(\tau^*_j)^{p+1}$$ \hfill (2.39)

we get, after reorganization that

$$\frac{TOL}{(\tau^*_j)^{p+1}} = c(t_j) = \frac{|\epsilon_{j+1}|}{\tau_j^{p+1}}$$ \hfill (2.40)

Rearranging this expression gives

$$\left(\frac{\tau^*_j}{\tau_j}\right)^{p+1} = \frac{TOL}{||\epsilon_{j+1}||}$$ \hfill (2.41)
Rearranging this equation and including a safety factor $\rho$ for the tolerance gives Equation 2.36. The second part of the statement at line number 9 is also a precaution. It both gives an absolute maximum time step, $\tau_{max}$ as well as a maximal increase defined by $q$. This is to ensure that the step size don’t accelerate to fast.

**Algorithm 1** Step size controller [13]

1: Choose initial value $x_0$ and first time step $\tau_0$ \hspace{1cm} \triangleright \text{Initiate the iteration index}
2: $j = 0$
3: $\Delta_t = \{t_0\}$ \hspace{1cm} \triangleright \text{Initiate the time set}
4: $x_{\Delta}(t_0) = x_0$ \hspace{1cm} \triangleright \text{Initiate the solution set}
5: while $t_j < T$ do
6: \hspace{0.5cm} $t = t_j + \tau_j$
7: \hspace{0.5cm} $x = \Psi^{t,t_j} x_{\Delta}(t_j)$
8: \hspace{0.5cm} compute the error estimate $|\epsilon_j|$
9: \hspace{0.5cm} $\tau = \min\left(q\tau_j, \tau_{max}, \sqrt[\rho TOL]{|\epsilon_j|} \right)$
10: if $|\epsilon_j| \leq TOL$ then \hspace{1cm} \triangleright \text{Step is accepted}
11: \hspace{1.5cm} $t_{j+1} = t$
12: \hspace{1.5cm} $\Delta_t = \Delta_t \cup \{t_{j+1}\}$
13: \hspace{1.5cm} $x_{\Delta}(t_{j+1}) = x$
14: \hspace{1.5cm} $\tau_{j+1} = \min(\tau, T - t_{j+1})$
15: \hspace{1.5cm} $j = j + 1$
16: else \hspace{1cm} \triangleright \text{Step is rejected}
17: \hspace{1.5cm} $\tau_j = \tau$
18: end if
19: end while

To implement Algorithm 1, one primarily need two parts; a way to approximate the solution at the next step and an error estimation. The solution at the next step is approximated by the discrete evolution $\Psi$ which will be implemented using the BDF2 solvers. The error estimation will be explained in the next section.

### 2.5.1 Error estimation

To control the error, the norm of the difference between the second order polynomial predictor $Y_n^{p2}$ and the calculated solution $Y_n$ should be bound. The second order predictor was chosen as it works well with BDF2 [12]. In mathematical terms, this bound introduces the inequality

$$\left\| Y_n - Y_n^{p2} \right\| \leq Tol$$  \hspace{1cm} (2.42)

Often, the tolerance $Tol$ is set as a combination of a relative tolerance, $relTol$, and an absolute tolerance, $absTol$. This can be expressed as

$$\left\| Y_n - Y_n^{p2} \right\| \leq Tol = relTol \cdot \left\| Y_n \right\| + absTol$$  \hspace{1cm} (2.43)
2.5. ADAPTIVE STEP SIZE CONTROLLER

Considering this equation component-wise and rearranging it gives the error estimate $|\epsilon_{j+1}|$ with corresponding bound $TOL$

$$|\epsilon_{j+1}| = \max_i \frac{|Y_{n,i} - Y_{p2}|}{relTol \cdot |Y_{n,i}| + absTol_i} \leq 1 = TOL$$ (2.44)

This is how the error estimation is calculated at line 8 in Algorithm 1. It also defines the parameter $TOL$ as $TOL = 1$. Thus, we have exchanged the tolerance $TOL$ by the relative tolerance $relTol$, which also is user defined. For further simplifications, the absolute tolerance can be expressed as the relative tolerance multiplied with a typical solution as $absTol_i = relTol \cdot |Y_{typical,i}|$.

2.5.2 PI controller

With knowledge in control theory, one can identify the right hand side of Equation 2.36 as an I-controller. In [13], Deuflhard expands this controller to a PID-controller. This has the form of

$$\tau_{k+1} = \left( \frac{\rho \cdot Tol}{|\epsilon_{k+1}|} \right)^{\beta_I + \beta_P + \beta_D} \left( \frac{|\epsilon_k|}{\rho \cdot Tol} \right)^{\beta_P + 2\beta_D} \left( \frac{\rho \cdot Tol}{|\epsilon_{k-1}|} \right)^{\beta_D} \tau_k$$ (2.45)

Choosing $\beta_P = \beta_D = 0$ and $\beta_I = 1$ we get the same expression as before.
Chapter 3

Method and Implementation

Synaptic plasticity is one of the well-known phenomena in neuroscience that is involved in questions about memory and its development [14]. In order to better understand this phenomenon we focus on multiple temporal and spatial tightly coupled scales, electrical and biochemical signals. It is acknowledged that plasticity-inducing stimuli (i.e. calcium influx) activates the mitogen-activated protein kinase (MAPK/ERK signaling pathway) on the sub-cellular level of the hippocampal neurons. An activated signaling pathway, in turns, regulates synaptic strength and plasticity in the mammalian nervous system by altering the electrical properties of the membrane excitability (i.e. through modification of the ion channel properties).

In this thesis, a model of the interaction between chemical and electric signals in neurons is investigated. Especially, an electric current will be injected which triggers a release of calcium, which in turn will take the chemical system from one steady state to another. For this, a simple model of a neuron was set up, explained in Section 3.1. Once the model is constructed, the problem has to be solved. How this is done is explained in Section 3.2. The solutions obtained needs to be evaluated, which is explained in Section 3.3.

3.1 Modeling the system

In this thesis, the neuron is modeled to consist of a soma connected to a dendrite connected to a spine. The model is taken from Maya Brandi [10], with some modifications. Here follows a short description of the model.

Each part of the neuron is geometrically described by a cylinder. The dendrite has been divided into 15 different sub-compartsments to obtain more realistic dynamics of the membrane excitability. The dimensions of each component are given in Table 3.1 and are chosen to be in range of real neurons.
CHAPTER 3. METHOD AND IMPLEMENTATION

### Table 3.1. Geometric dimensions of the modeled neuron.

<table>
<thead>
<tr>
<th>Compartment</th>
<th>Height [µm]</th>
<th>Length [µm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soma</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Dendrite</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>Spine</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 3.1. A sketch of the compartment description of a neuron

#### 3.1.1 Electric system

The electric part of the neuron is modeled by 17 subsequent circuits (one for the soma, 15 for the dendrite and one for the spine), connected through resistances. A sketch of this is shown in Figure 3.1. Each circuit is of the model described in Section 2.2.3. The solution to the electric system can be seen in Figure 3.2. From this, it can be deduced that the electric system has a time scale of order 1/10 of seconds.

**Soma**

The soma contains two voltage dependent currents, $I_{Na}$ and $I_K$. The currents follow Equations 2.2. The membrane potential is denoted by $V_{m1}$ and follows Equation 2.10.

**Dendrite**

In the modeling of the dendrite, this system differs from that of Maya Brandi. Instead of one equation for the dendrite, it will be 15 equations. The changes are trivial and corresponds to relations with neighboring compartments.
3.1. MODELING THE SYSTEM

Spine

The model of the spine contains two types of active ion channels and two voltage dependent currents, \( I_{Ca} \) and \( I_{KCa} \). The current \( I_{KCa} \) is, not only dependent of the voltage, but also of the calcium current \( I_{Ca} \). The equation describing this differs a bit from the other, and has the form

\[
I_{KCa} = (E_K - V_m) \bar{g}_{KCa} \frac{K^*_Ca}{K^*_Ca + K_{Ca}}
\]

Here, \( K^*_Ca \) is the number of active calcium dependent potassium channels where as \( K_{Ca} \) is the number of closed ones. As can be seen, the dependency of the calcium current is not explicit, rather it is implicitly dependent via the fraction of active channels.

3.1.2 Chemical system

The chemical dynamics are only modeled in the spine. It is in the spine that the electric dynamics interact with the calcium concentration. The spine is modeled as the same size as for the electric system. The time scale of the chemical system is approximately 10 s, thus around 100 times larger than that of the electric system. This can be seen from the solution of the chemical system in Figure 3.3.

The chemical subsystem is a Mitogen-activated protein kinase model derived by U.S. Bhalla [2]. The model contains 18 molecular species, 6 reactions and 13 enzymes. The model is bistable, i.e. has two steady states. A switch between these steady states is what is sought for in this thesis.
Figure 3.3. The solution for the chemical system.

Figure 3.4. A description of the biochemical reactions in the MAPK model. The picture is taken from U.S. Bhalla [2].
3.2. IMPLEMENTATION

3.1.3 Dynamics

To trigger the switch mentioned in the previous section, a five second injection of Ca\(^{2+}\) is needed. This will be received by injecting a five second current in the soma of 90 pA, which will effect the potential of the soma. This in turn will change the dynamics for the dendrite. The reaction will propagate through all dendrite compartments until it reaches the spine. Here, the voltage dependent calcium channels releases the desired calcium level.

3.1.4 System interaction

As mentioned in Section 2.4, the two modeled system will in this thesis be coupled together. We connect these models through Ca-signaling since it is known to play an important physiological role in such brain functions as learning and memory (synaptic plasticity). For the sake of efficiency we exchange only required variables between the systems instead of an entire vector of state variables.

The required values by a remote system are calculated locally by each system before they are exchanged. The electric system is responsible to provide a calcium injection rate \([\text{M/s}]\) to a biochemical system (Equation 3.2). It uses the potential in the spine, gate variables of the calcium channel as well as a calcium concentration variable for calculating the current \(I_{Ca}\) in the formula. The latter variable is provided by chemical system. This can be expressed by the formula

\[
k_{\text{inj}} = \frac{6.242 \cdot 10^{18}}{2 \cdot Na \cdot vol} \cdot I_{Ca} \quad \left[ \frac{\text{M}}{\text{s}} \right]
\]

where \(Na = 6.022 \cdot 10^{23}\) is Avogadro’s number, \(vol = 10^{-18} \cdot 10^3\) l is the volume of the spine. The factor \(6.242 \cdot 10^{18} / 2\) comes from the number of calcium ions per Coulomb.

The chemical system, in turn, provides the calcium concentration value as stated above and the fraction of active calcium dependent potassium channels used by electrical system as shown in Equation 3.1 [10].

3.2 Implementation

At this point, the equations for each system have been formulated. We also know how the two subsystems interact and which values we need to exchange. What remains now, is to implement the algorithm to simulate the time propagation of the formulated problem.

For the fixed step case, a trivial time stepping algorithm was constructed. Since it only contains obvious steps an explanation is omitted. The MATLAB code presented in the appendix should give enough information.

The algorithm for the adaptive step solver has already been described in Algorithm 1. In this section, we address the question how the problem parameters was set up and how the different steps of the algorithm were implemented. The important parts of the implemented MATLAB code can be found in the appendix.
The implementation assumes that methods for solving each individual system exists. Thus, after the problem has been properly formulated, the solution at the next time step is given by these pre-requested solvers. In my implementation I have, as explained in Section 2.3, used either RK4 for the chemical system and Hines method for the electric (in the fixed step solver), or BDF2 for both systems (in both the fixed step and adaptive solver).

**Initializing**

The first part of the implementation is the set up. Here, the initial values of the two systems as well as the time intervals are introduced. The initial parameters for the system are chosen so that the chemical system is as close to one steady state as possible. To ensure that it is at the steady state, the simulation is run for a short period of time [1 s] without any injected current. After that, the current mentioned in Section 3.1.3 are injected for five seconds. This is the shortest time that is needed for the steady state switch. When the injection has ended, the simulation is run for another 39 seconds, for the total simulation time of 45 s. At this point, the chemical system has settled at the second steady state.

**Co-simulation**

To solve the decoupled systems implicitly, each system needs to know the exchange variables at the next time step. This is only possible for the second system using a Gauss-Seidel organization. In the other cases, the exchange variables are extrapolated from their last values. For the first time step, a constant extrapolation is used; at the second time step, a linear extrapolation is used; and at the remaining steps a second order extrapolation is used.

**Error Estimation**

The error estimation here is similar to that explained in Section 2.5.1. For the first step, no error estimation are calculated. Instead, the time step is automatically accepted. Thus we have to initialize the first time step to be quite small. In this thesis, the initial time step was chosen as $10^{-4}$ to ensure that both subsystems can be solved accurate. Since the BDF2 method requires the solution at at least two previous time steps, the first time step is taken by the implicit Euler formula.

For the second step, the error estimation is based on a linear predictor instead of the second order predictor previously explained. From now on, the time stepping utilizes the BDF2 methods.

For the remaining steps, the error estimation is calculated via the second order predictor and the solution at the next step is calculated via the BDF2 methods.

---

1The times in this section refer to simulated time. The wall clock time of the simulation differs for different solvers and computers.
3.3. HOW TO COMPARE SOLUTIONS

Step size controllers
In this thesis, a PI-controller and an I-controller was used. Express the exponents in Equation 2.45 as \( a = \beta_I + \beta_P + \beta_D, \ b = \beta_P + 2\beta_D \) and \( c = \beta_D \). The I-controller corresponds to choosing the parameters as \( a = 1, b = c = 0 \) whereas the PI-controller corresponds to parameters \( a = 0.7, b = 0.4, c = 0 \). The parameter \( c \) has always been chosen as \( c = 0 \), i.e. no parts of a D-controller was tested.

The controllers requires some additional parameters to be set. The parameter \( q \) was set as \( q = 2 \), i.e. the maximum increase of step size is to double the current step. This was chosen as a precaution so that the step sizes does not vary too much. The maximal time step, \( \tau_{max} \) on the other hand was set as \( \tau_{max} = \infty \). Since all systems have individual maximal steps, which are not necessary the same for coupled systems, this choice was made to reduce user input. Last, but not least, was defining the parameter \( TOL = 1 \). The reason for this was explained in Section 2.5.1.

3.2.1 Adaptive solver for the Runge-Kutta and Hines methods
An adaptive step size controller could be implemented for the Runge-Kutta and Hines methods. The problem with this, is that these methods need an error estimation, corresponding to Section 2.5.1.

With fourth order Runge-Kutta, the easiest error estimation is obtained using step doubling [15], i.e. taking each time step twice: once as a full step and once with half the step size. The step doubling takes 4+8 function evaluations, but the first is the same so it requires 11 evaluations. This is compared to 8 evaluations of non-adaptive Runge-Kutta, since the precision of using half the time step is received. Consequently, this would increase the number of function evaluations by a factor of \( \frac{11}{8} = 1.375 \). The difference between the two calculated solutions in the step doubling approach can be used as an error estimation.

Another adaptive step size algorithm for Runge-Kutta methods was invented by Fehlberg [16]. He discovered two different Runge-Kutta methods, one of order 4 and one of order 5, that both used the same six function evaluations. The difference between these to approximations can then be used as the error estimation. This new strategy, known as embedded Runge-Kutta methods, are roughly a factor of two more efficient than the step doubling.

For the Hines method, no other error estimation than the step doubling is known to the author. Since the number of function evaluations per iteration is fixed (see Section 3.3.2), the increase of function evaluations using step doubling would be of a factor 1.5.

3.3 How to compare solutions
To evaluate if a obtained solution is correct (or sufficiently correct) one has to decide which method to use to compare two given solutions. The simplest way to do this, even though not a stringent way, is to simply plot the solutions as a graph and use a visual comparison technique. As mentioned in Section 3.1, the desired response is the switch
CHAPTER 3. METHOD AND IMPLEMENTATION

from one stable state to another. For this, the elevation of calcium level during the
stimulation period is crucial. To evaluate the calcium level, Fourier analysis is used as
a rigorous way to compare the solutions.

3.3.1 Fourier analysis

The discrete Fourier transform is given by

\[ X_k \equiv n(N) \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}, \quad k \in \mathbb{Z} \tag{3.3} \]

where \( N \) is the segment size and \( n(N) \) is a normalization factor, often chosen as \( n(N) = \frac{1}{N} \) or \( n(N) = \frac{1}{\sqrt{N}} \). The Fourier coefficients are easily calculated via the Matlab native function \texttt{fft} which utilizes the Fast Fourier Transform.

Of special interest is the zeroth order Fourier coefficient. From Equation 3.3 one can see that the zeroth order coefficient correspond to a summation of the function values. With the correct normalization factor, \( n(N) = \frac{1}{N} \), this corresponds to the mean value at the chosen interval. To compare solutions we thus compare the magnitude of the zeroth order Fourier coefficient.

3.3.2 Solver efficiency

To get an estimation of not only the accuracy, but also the efficiency, the error is shown in relation to the number of function calls that are made to the functions corresponding to the right hand side of the system of ODEs. This gives a good approximation of the computational cost that is used. For the BDF2 methods as well as the Runge-Kutta method, this was very simple, since both functions actually calls a function that exactly matches the right hand side functions. The Runge-Kutta method calls the function exactly four times, one for each line coefficient defined by Equation 2.21, per time step. Since the BDF-method solves the system of ODEs implicitly, an iterative method is used (in this implementation a modified Newtons method was used). Thus, these solvers call for the function describing the right hand side several times per step.

For the Hines method, new functions were implemented to handle the right hand side and construct the matrices needed for the calculations. These corresponds roughly to four calls to the right hand side function. The method might be implementable in an more efficient way which would reduce the cost slightly. For example, in the current implementation of Hines method, some subroutines are called several times when constructing the matrices, instead of just once.

3.3.3 The correct solution

Since the system has no analytic solution, the correct solution was chosen as the numerical solution calculated by MATLAB inbuilt function \texttt{ode15s}. \texttt{ode15s} is a solver

\(^2\text{http://www.mathworks.se/help/matlab/ref/fft.html}\)

\(^3\text{http://www.mathworks.se/help/matlab/ref/ode15s.html}\)

24
3.3. HOW TO COMPARE SOLUTIONS

designed for stiff problems. It is a quasi-constant step size implementation of backward differences [17]. To achieve a high precision solution, the options 'Reltol' and 'Abstol' where set to $10^{-6}$ and $10^{-9}$ respectively.

This solution is shown in Figure 3.5, on two different time scales. As can be seen, the system settles at the second steady state after the stimuli.

![Figure 3.5](image)

(a) Solution for first 45 seconds  
(b) Solution for first 7 seconds

**Figure 3.5.** The correct solution, calculated by *ode15s*
Chapter 4

Results and Analysis

This section is organized as follows. First, a comparison between different fixed step size solvers are shown. Secondly, the effect of the adaptive solver with different controllers are shown, both for step sizes and precision. This also includes the behavior of the error as a function of the relative tolerance. Last is an ad hoc comparison on what effect a forced relation between step sizes for each system has on the efficiency.

4.1 Comparing different fixed step size solvers

As explained in Section 2.3, the common way to solve the chemical reactions and electric equations are the Runge-Kutta method and Hines method respectively. The goal here is to compare that solution with a solution obtained by solving both systems with BDF2 and using an adaptive controller for controlling the step sizes.

Before that, I will use both solver combinations on a fixed step size grid. Since the two methods can’t be compared directly for adaptive step sizes (see Section 3.2.1), the methods are firstly compared at equal terms. As mentioned before the solution of each simulation is compared to the correct solution, explained in Section 3.3.3, and the relative error of the zeroth Fourier coefficient is plotted versus the number of function calls to the function describing the right hand side of the original ODE.

For the fixed iterations, one could mix the order of stepping, in the same way as is explained in Section 2.4.1. The results for the fixed step solutions for the different iteration methods are shown in Figure 4.1.

As can be seen, the Runge-Kutta and Hines method is less accurate than the BDF2-combination and comes with a cost of slightly increased number of function calls. This makes the future, i.e. the implementation of the adaptive solver, looking promising.
4.2 Adaptive solver

Secondly, we will see the results of the adaptive step size controller as well as some aspects of its implementation. The aim of the adaptive solver is to reduce the computational cost of the simulation while not increasing the error. The reduced computational cost will mainly manifest as a reduced number of time steps taken, although each step will be more expensive due to the iterative method.

4.2.1 The effect of a step size controller on the step lengths

A desired aspect of the adaptive step size controller, is to have step sizes that do not vary to much in sizes. Swift changes in the step sizes will increase the number of times the step size controller needs to redo each step. It might also effect the stability of the solution. Thus, a smooth distribution of the step sizes are desired.

Figure 4.1. A comparison between fixed step solutions for different iteration strategies. The three different step sizes used in each curve are \((4 \cdot 10^5, 8 \cdot 10^5, 1.6 \cdot 10^6)\).
4.2. ADAPTIVE SOLVER

In Figure 4.2, the step sizes are shown for a simulation with an I-controller and one with a PI-controller. It is very clear that PI-controller has a much smoother step size variation, as is desired.

Noteworthy is that during the injection of the external current, i.e. from 1s to 6s, very small steps are needed. This is true for both controllers presented here, as well as all other PI-controllers tested during the work. This is also the case for the solution obtained by ode15s. The steps required during the stimulation are so small, that it is here the most computational power is spent. For the PI-controlled solution whose time steps are presented here, approximately 90% of the time steps reside in this time period.

4.2.2 The limiting subsystem

The error estimation calculated in the step size controller uses the maximum norm of the obtained errors. Thus, for each step there is a certain element that will set the limit of the next time step. From this, it can be determined which subsystem is setting the limit. An example of this can be seen in Figure 4.3. In this case the simulation is a Jacobi iteration with a PI-controller, but all simulations in the scope of this thesis show similar results.

As can be seen in Figure 4.3(b) it is the electric system that most frequently is the limit during the simulation. Considering Figure 4.3(a) on the other hand, we see that it is the chemical system that is the limit for the major part of the simulation time. We can also see that the part where the electric system is the limit, coincide very well with the time of injected current. This is consistent with the results of the previous section.

4.2.3 Changing the relative tolerance

Decreasing the relative tolerance of the algorithm explain in Section 2.5 should give more accurate solutions. As explained in Section 2.5 the specified relative tolerance only
affect the local error control. In this case, we also see that the global error is reduced for decreased relative tolerances, for all iteration methods and both the I- and PI-controller. The Gauss-Seidel iteration with the electric system stepped first seems to be performing the least satisfying.

The results compared to the original strategy
Up to now, we have examined the performance of the adaptive controller and compared different controllers and iteration methods.

Before any final conclusions are drawn, it is reasonable to compare the adaptive solvers to a fixed step size solution. In Figure 4.5 the results from the adaptive solver shown in Figure 4.4(b) are compared to the fixed step solution of the Runge-Kutta and Hines method shown in Figure 4.1(b).

The results speak for them self; the adaptive strategy has a higher accuracy, as well as a lower computational cost.

4.2.4 Fixed relations
Since the two subsystems operate on different time scales, one can conclude that the step sizes required for stability and accuracy differs between the two systems. We have already seen this in Section 4.2.2.

Ideally, one would want to separate the two systems even more and possibly solve them with adaptive step controllers for each subsystem, instead on a global level. These should be run for short periods of time and then exchange information. This would result in a global time step between synchronizations as well as individual time steps for each subsystem. Added on this could be that the synchronization points also would follow an adaptive controller. This has been found difficult to implement due to the weight-off between adapting the global or local step sizes.
Figure 4.4. The relative error versus function of calls to the right hand side, for two different adaptive controllers. The relative tolerance was chosen as $10^{-3}$, $7.5 \times 10^{-4}$ and $5 \times 10^{-4}$ with decreasing errors as a result.
To show the effect the difference in time steps could have on the solutions, a forced step size ratio method was implemented. Continuing with an adaptive step size controller, one system was forced to take a fixed number of time steps per synchronization step.

The ratios were chosen as shown in Table 4.1. Ideally, the ratios should be chosen in relation to the difference in time scales. As mentioned in Section 3.1.2, the difference in time scales are a factor of 100. Thus, a simulation was run with one step per synchronization for the chemical system and 100 steps per synchronization for the electric system. Unfortunately, this did not reduce the number of synchronization points significantly, as can be seen in Table 4.2. Since this on the other hand increased the number of function calls significantly, it was too inefficient to perform tests at this level. Thus, a much lower ratio was chosen to simplify the simulations.

The local errors in the method, arise mainly from two factors. The errors are caused by the numerical solvers of a system, as well as the approximation of the exchanged variables. The different ratios should mainly reduce the former. This is probably the cause for the forced ratio approach do not reduce the number of synchronization points.

The result of the forced ratio approach can be seen in Figure 4.6. As usual, it shows the errors of the zeroth Fourier coefficient as a function of function calls to the right hand side of the ODE. In this case the step sizes are controlled by a PI-controller, with parameters $(a = 0.7, b = 0.4)$ for the three different iteration methods.

The results are somewhat ambiguous. On one hand, some ratios decreases the rel-
### 4.2. ADAPTIVE SOLVER

#### Table 4.1. Number of steps per synchronization step.

<table>
<thead>
<tr>
<th>Chemical system</th>
<th>Electric system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Figure 4.6. A comparison between the error of the zeroth Fourier coefficient for different iteration strategies. The marker indicate what forced ratio resulted in that specific point.

(a) Jacobi iteration

(b) Gauss-Seidel iteration with chemical system first

(c) Gauss-Seidel iteration with electric system first
Table 4.2. Number of synchronization points for different forced ratios. The data comes from the same simulation as in Figure 4.6(a)

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Synchronization points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 : 1</td>
<td>32717</td>
</tr>
<tr>
<td>1 : 2</td>
<td>32632</td>
</tr>
<tr>
<td>1 : 5</td>
<td>32646</td>
</tr>
<tr>
<td>2 : 1</td>
<td>31754</td>
</tr>
<tr>
<td>5 : 1</td>
<td>31262</td>
</tr>
<tr>
<td>1 : 100</td>
<td>32167</td>
</tr>
</tbody>
</table>

ative error. In addition to this, some ratios have as desired, a decreased number of synchronization points, as shown in Table 4.2. On the other hand, these advantages are almost insignificant in comparison to the increased number of function calls that this requires.
Chapter 5

Summary and Future Work

The main objective of the project was to develop an efficient and reliable method for integrating neural models that span multiple physical and chemical processes and act on different temporal and spatial scales. This was successfully done, and the experiments on the constructed example problem showed good results.

In this thesis, integration methods applicable and usable in neuroscience were studied and implemented. Different organizations for coupling given systems where investigated. Two systems were coupled and co-simulated in different ways. Primarily, the Runge-Kutta method and Hines method where studied and compared to a fixed step size implementation of BDF2 methods. Here, the BDF2 methods performed better than the commonly used Runge-Kutta and Hines method. No significant difference was detected between the different organizations, although the Gauss-Seidel iteration showed slightly more promising results. Among the two different Gauss-Seidel implementations, i.e. which system was chosen as the first one, the iterations with the chemical system first showed slightly better results. On the other hand, the Jacobi iteration is parallel in its foundation and such an implementation, if possible, should reduce the wall-clock time of simulation which of course is desired.

Secondly, local error estimation techniques was studied and used to implement an efficient adaptive step size method. An I-controller was compared to a PI-controller, and showed the expected results. To quote K. Gustafsson [18], "... and a control-man knows that PI is always better than I...". The better performance of the PI-controller was mainly manifested in a smoother step size variation.

The method was tested on a given example in MATLAB and the results where analyzed. The adaptive step size controller was found to be much more efficient than the original fixed step solution, with reduced errors as well as function evaluations.

As was briefly discussed in Section 4.2, the limiting factor of efficiently co-simulate these models is the electric system during the injection of external current. It is during this time almost all the computational cost is spent and very small time steps are required. To improve efficiency of the computations further, this part needs more considerations.

During the work, an attempt to solve each system with an individual step size con-
troller was performed. The implementation suffered from time steps converging to zero. It could definitely be worth to spend more effort in the matter. As shown in Section 4.2.4, different time steps for each subsystem could offer a more accurate solution although the minor improvements presented in this thesis came at a cost of a higher amount of function calls.

The final conclusion, is that the method developed during the thesis was very successful in co-simulating the two systems and performed much better than the original strategy. Since the two subsystems of the example problem had very different time scales, it should be possible to co-simulate other models as well, using the implemented method.
List of Figures

2.1 Structure of a typical neuron. ............................................. 4
2.2 The membrane of a cell modeled as an electric circuit. ............. 5
2.3 The stability domain of the BDF2 method lies outside the plotted curve. ... 11
2.4 Schematic sketch of the Jacobi organization ................................ 12
2.5 Schematic sketch of the Gauss-Seidel organization ......................... 12

3.1 A sketch of the compartment description of a neuron ................... 18
3.2 The solution for the electric system for a short period of time. ........... 19
3.3 The solution for the chemical system. ..................................... 20
3.4 A description of the biochemical reactions in the MAPK model. The picture is taken from U.S. Bhalla [2]. ................................. 20
3.5 The correct solution, calculated by ode15s ............................... 25

4.1 A comparison between fixed step solutions for different iteration strategies. The three different step sizes used in each curve are \((4 \cdot 10^5, 8 \cdot 10^5, 1.6 \cdot 10^6)\). ................................................ 28
4.2 Comparing the step size taken by the I-controller and PI-controller, as a function of simulation time. ............................... 29
4.3 The limiting subsystem for a Jacobi iteration with PI-controller. Other simulations show similar results. ................................. 30
4.4 The relative error versus function of calls to the right hand side, for two different adaptive controllers. The relative tolerance was chosen as \(10^{-3}, 7.5 \cdot 10^{-4} \) and \(5 \cdot 10^{-4}\) with decreasing errors as a result. ............................. 31
4.5 The errors of the adaptive PI-controller for different organizations, compared to the fixed step size solver using the Runge-Kutta and Hines method. ........ 32
4.6 A comparison between the error of the zeroth Fourier coefficient for different iteration strategies. The marker indicate what forced ratio resulted in that specific point. ................................. 33
Appendix A

MATLAB code

In this appendix, some of the MATLAB code developed during the thesis work are provided. The code however is not enough for running the simulations. The equations describing the two systems are left out, as well as the individual solvers. For access to the entire code, send an e-mail to me at jerkern@kth.se.

A.1 Fixed step solver

A.1.1 Main program

```
function sim_combine2j2()

%%%% Solver parameters
global iterMethod;

%%%% Modelling parameters
global T_STI_START T_STI_END nDSeg

%%%% Function counters and control parameters
global erk_iter hh_iter ode_erk_iter ode_cell_iter

%%%% Defining step sizes
global Nsys Nerk Nhh
Nsys = 4e5;
Nerk = 1; Nhh = 1;

%%%% Define saving string
info = sprintf('Nsys=%uNhh=%dNerk=%d',Nsys,Nhh,Nerk);

%%%% Set up time scales
T_SIM = 45; %s
T_STI_START = 1; %s
T_STI_END = 6; %s
```
APPENDIX A. MATLAB CODE

%% Defining number of dendrites
nDSeg = 15;

%% Initiate solver
organizations = ...  % Organization for solve decoupled systems
    {'Jac','GSwErkfirst','GSwCellfirst'};
iterMethod = organizations{1};
solv1 = @RK4;  % Solver for chemical reactions (not provided)
solv2 = @HinesMethod;  % Solver for electric systems (not provided)

%% Initial state variables
cell_init_vals = get_inits(nDSeg, false);
erk_init_vals = get_species(true);
erk_size = length(erk_init_vals);

%% Interface between two systems
[~, erk_exch] = solve_erk_handle();
[~, cell_exch] = solve_cell_handle();

%% Run the simulation
[t, y] = ode_solver(@solve_sys,[0 T_SIM], ...
    [erk_init_vals, cell_init_vals]', [], ...
    erk_size, {solv1 erk_exch}, {solv2, cell_exch});

%% Saving the data
mkdir(sprintf('../../Data/%s',date));
filename = ...
    sprintf('../../Data/%s/%s%s%s%s.mat',date,iterMethod,func2str(solv1),...
        func2str(solv2),info);
save(filename);
end

A.2 Time stepping algorithm

function [tint, y]=ode_solver(solve_sys,tint, ...
%% Defining time intervals and step sizes

t0 = tint(1); T_SIM = tint(2);
tint = linspace(t0, T_SIM, Nsys);
dt = tint(2) - tint(1);

%% Set up initial values etc.
y = zeros(length(init_vals), Nsys);
y(:,1) = init_vals;

%% Time stepping
for sys_index = 1:Nsys - 1

  %% For the first steps, use constant predictors
  if sys_index < 3
    [frac_tilde, cai_tilde] = feval(erk_sys{2},... 
y(1:erk_size, sys_index));
    erk_tilde = [frac_tilde cai_tilde];
    [ca_flux_tilde] = feval(cell_sys{2},...
      y(erk_size+1:end, sys_index), cai_tilde);
    cell_tilde = ca_flux_tilde;
  else
    erk_tilde = zeros(3,2);
    cell_tilde = zeros(3,1);
    for jj = [2 1 0]
      [frac_tilde, cai_tilde] = feval(erk_sys{2},...
        y(1:erk_size, sys_index-jj));
      erk_tilde(end-jj,:) = [frac_tilde cai_tilde];
      [ca_flux_tilde] = feval(cell_sys{2},...
        y(erk_size+1:end, sys_index-jj), cai_tilde);
      cell_tilde(end-jj) = ca_flux_tilde;
    end
  end

  %% Current state
  states = y(:, sys_index);

  %% Get the solution at the next step
  out = solve_sys([tint(sys_index) tint(sys_index+1)], states, erk_size,...
    erk_sys, cell_sys, erk_tilde, cell_tilde);
  y(:, sys_index+1) = out;
end

%% Error handling
if(~sum(erk_iter))
  erk_iter = Nsys*Nerk;
end
if(~sum(hh_iter))
  hh_iter = Nsys*Nhh;
end

41
APPENDIX A. MATLAB CODE

A.3 Intersection between systems

function out = solve_sys(t, states, erkSize,...
    erkSys, cellSys, erkTilde, cellTilde)
    global iterMethod;

    %%% Solving according to the desired organization
    erk_vals = states(1:erkSize);
    cell_vals = states(erkSize+1:end);
    erk = erkSys{1};
    cell = cellSys{1};

    if(strcmp(iterMethod,'Jac'))
        out1 = erk(t, erk_vals, cellTilde, cellSys{2}, @dummy, erk_vals(1));
        out2 = cell(t, cell_vals, erkTilde, erkSys{2});
    elseif(strcmp(iterMethod,'GSwErkfirst'))
        out1 = erk(t, erk_vals, cellTilde, cellSys{2}, @dummy, erk_vals(1));
        [fracTilde, caiTilde] = feval(erkSys{2}, out1);
        erkTilde = [fracTilde caiTilde *1e3];
        out2 = cell(t, cell_vals, erkTilde, erkSys{2});
    elseif(strcmp(iterMethod,'GSwCellfirst'))
        out2 = cell(t, cell_vals, erkTilde, erkSys{2});
        [caFluxTilde] = feval(cellSys{2}, out2, erkTilde(1,2));
        cellTilde = caFluxTilde;
        out1 = erk(t, erk_vals, cellTilde, cellSys{2}, @dummy, erk_vals(1));
    else
        fprintf('Wrong Organization');
    end
    out = [out1;out2];
end

A.4 Adaptive solver

A.4.1 Main program
A.4. ADAPTIVE SOLVER

function sim_combine()

%%%% Solver parameters
global dt a b Nhh Nerk iterMethod

%%%% Modelling parameters
global T_STI_START T_STI_END noDendSeg

%%%% Function counters and control parameters
global erkIter hhIter ode_erkIter...
    ode_cellIter nanIters refineIters...
    acceptedErrorIndex rejectedErrorIndex

%%%% Initiate globally used counters
ode_cellIter = 0; ode_erkIter = 0; nanIters = 0;
refineIters = 0; hhIter = []; erkIter = [];
acceptedErrorIndex = []; rejectedErrorIndex = [];

%%%% Initiate solvers
organization = ... % Organization for solve decoupled systems
    {'Jac','GSwErkfirst','GSwCellfirst'};
iterMethod = organization(2);
solvErk = @BDF2CombErk; % Solver for chemical reactions (not provided)
solvCell = @BDF2Comb; % Solver for electric systems (not provided)

%%%% Choosing solver parameters
dt = 1e^{-4}; % Initial time step
relTol = 1e^{-3}; % Relative tolerance of the adaptive solver
Nerk = 1; % Number of steps per synchronization
Nhh = 1; % Number of steps per synchronization
a = 0.7; b = 0.4; % Parameters for the PI controller

%%%% Define string for saving
info = sprintf('Reltol=%gNhh=%gNerk=%g',relTol,Nhh,Nerk);

%%%% Chosing model parameters
T_SIM = 45; % Total run time [s]
T_STI_START = 1; % Start of stimuli [s]
T_STI_END = 6; % End of stimuli [s]
noDendSeg = 15; % Number of dendrites

%%%% Initial state variables (functions not provided)
cellInitVals = get_inits(noDendSeg, false);
erkInitVals = get_species(true);
erkSize = length(erkInitVals);

%%%% Interface between two systems (functions not provided)
[~, erk_exch] = solve_erk_handle();
[~, cell_exch] = solve_cell_handle();

%%%% Run the simulation
[t, y] = ode_solver_adapPI(@solve_sys,[0 T_SIM], ...
    [erkInitVals, cellInitVals]', relTol, ...
APPENDIX A. MATLAB CODE

```
% APPENDIX A. MATLAB CODE

erkSize, {solvErk, erk_exch}, {solvCell, cell_exch});

%%% Saving the data
mkdir(sprintf('%sAdap', date));
filename = sprintf('%sAdap/%s%g%g%s.mat', 
    date, iterMethod, a, b, info);
save(filename);
end

A.5 Time stepping algorithm

function [tInt, y] = ode_solver_adapPI(solveSys, tInt, ...
    initVals, relTol, erkSize, erkSys, cellSys)
    %%% Initializing global variables
    global nanIters refineIters acceptedErrorIndex rejectedErrorIndex
    global yCellVec yErkVec a b Nerk Nhh
    global dt sysIndex hhIter erkIter ode_cellIter ode_erkIter iterMethod

    %%% Defining time intervals and step sizes
    tStart = tInt(1); % Starting time
    tEnd = tInt(2); % Ending time
    t = tStart; % Current time

    %%% Initiating solution vectors
    Nsys = 1;
    hhIter = zeros(1, Nsys);
    erkIter = zeros(1, Nsys);
    yCellVec = zeros(length(initVals(erkSize+1:end)), Nsys);
    yCellVec(:,1) = initVals(erkSize+1:end);
    yErkVec = zeros(length(initVals(1:erkSize)), Nsys);
    yErkVec(:,1) = initVals(1:erkSize);
    sysIndex = 0;
    y = zeros(length(initVals), Nsys);
    y(:,1) = initVals;

    %%% Defining parameters for the adaptive solver
    q = 2; % The limit for how much the steps are allowed to grow
    rho = 0.90; % Safety factor for tolerance limit
    dtMax = Inf; % Maximum step size
    TOL = 1; % Tolerance level for error

    yTypical = importdata('yTypicalSolution.txt', ' '); % Typical solution
    eEstVec = [NaN rho*TOL rho*TOL]; % Estimations of last errors

    %%% Time stepping until the end
    while t < tEnd
        sysIndex = sysIndex + 1;

        %%% Defining current state
        states = y(:, sysIndex);
```

A.5. TIME STEPPING ALGORITHM

```matlab
%% Generate the approximations of the exchange variables
if sysIndex == 1
    %% For the first step, the previous value approximate the next
    [fracTilde, caiTilde] = feval(erkSys{2},y(1:erkSize,sysIndex));
    erkTilde = [fracTilde caiTilde *1e3];
    [caFluxTilde] = feval(cellSys{2},...
        y(erkSize+1:end,sysIndex),caiTilde);
    cellTilde = caFluxTilde;
elseif sysIndex == 2
    %% For the second step, use the two previous values
    erkTilde = zeros(2,2);
    cellTilde = zeros(2,1);
    for jj = [1 0]
        [fracTilde, caiTilde] = feval(erkSys{2},...
            y(1:erkSize,sysIndex-jj));
        erkTilde(end-jj,:) = [fracTilde caiTilde *1e3];
        [caFluxTilde] = feval(cellSys{2},...
            y(erkSize+1:end,sysIndex-jj),caiTilde);
        cellTilde(end-jj) = caFluxTilde;
    end
else
    %% For the remaining steps, use the 3 previous values
    erkTilde = zeros(3,2);
    cellTilde = zeros(3,1);
    for jj = [2 1 0]
        [fracTilde, caiTilde] = feval(erkSys{2},...
            y(1:erkSize,sysIndex-jj));
        erkTilde(end-jj,:) = [fracTilde caiTilde *1e3];
        [caFluxTilde] = feval(cellSys{2},...
            y(erkSize+1:end,sysIndex-jj),caiTilde);
        cellTilde(end-jj) = caFluxTilde;
    end
end

%% Calculate the solution at the next step
out=solveSys([t t + dt(end)], states, erkSize,...
    erkSys, cellSys,erkTilde, cellTilde);

%% Update time and timestep
if any(isnan(out))
    dt(end) = dt(end)/2;
    sysIndex = sysIndex - 1;
    nanIters = nanIters + 1;
else
    %% For the first step, use Euler formula with no error estimation
    if sysIndex == 1
        t = t + dt(end);
        y(:,sysIndex+1) = out;
        dt(end+1) = dt(end);
    %% For later steps, find the error estimate
    else
        %% For the second step, use BDF2 with linear predictor
```
APPENDIX A. MATLAB CODE

```matlab
if sysIndex == 2
    gamma = dt(end)/dt(end-1);
    Ypn = y(:,sysIndex)+gamma*(y(:,sysIndex)-y(:,sysIndex-1));
    [eEst, eIndex] = max(abs(out-Ypn)./...
        (relTol*(abs(out)+ abs(yTypical))));
    \%\%\% For later steps, use BDF2 with second order predictor
else
    gamma = dt(end)/dt(end-1);
    \Delta = 1+dt(end-2)/dt(end-1);
    a2 = gamma*(gamma+\Delta)/(1-\Delta);
    a3 = gamma*(gamma+1)/((\Delta*(\Delta-1)));
    a1 = 1-a2-a3;
    Yp2n = ... \% 1*y(:,sysIndex)+a2*y(:,sysIndex-1)+a3*y(:,sysIndex-2);
    [eEst, eIndex] = max(abs(out-Yp2n)./...
        (relTol*(abs(out)+ abs(yTypical))));
end

% Once we have an error estimation, define the optimal timestep
% for the current time. Order 2 method => 3

eEstVec = circshift(eEstVec',2)';
eEstVec(3) = eEst;

% Using notation from Hairer, Deuflhard uses a different
% notation, but does not specify the constants...
p = 2; \% Order of the method
c = 0;
dtSuggest = ...
dt(end)*(rho*TOL/eEstVec(3))^(a/p)... \%Proportional part
    *(eEstVec(2)/(rho*TOL))^(b/p)... \% Integrating part
    *(rho*TOL/eEstVec(1))^(c/p); \% Differenting part

diffTvalues = [q*dt(end), dtMax, dtSuggest];
dt_next = min(diffTvalues);
if eEst < 1
    t = t + dt(end);
dt(end+1) = dt_next;
y(:,sysIndex+1) = out;
    acceptedErrorIndex(end+1) = eIndex;
else
    dt(end) = dt_next;
    refineIters = refineIters + 1;
sysIndex = sysIndex-1;
    eEstVec = circshift(eEstVec',1)';
    rejectedErrorIndex(end+1) = eIndex;
end
end
```

46
A.6 INTERSECTION BETWEEN SYSTEMS

1.46 %%% Ensure that we don't leave the interval
1.47 if t + dt(end) > tEnd
1.48     dt(end) = tEnd-t;
1.49 end
1.50
1.51 if (~sum(erkIter)), erkIter = length(dt); end
1.52 if (~sum(hhIter)), hhIter = Nsys*Nhh; end
1.53
1.54 %%% Return values
1.55 tInt = cumsum(dt);
1.56 y = y(:,1:length(dt));
1.57
1.58 %%% Display solver statistics
1.59 fprintf('%d steps had to be refined and taken again
',refineIters)
1.60 fprintf('%d steps resulted in NaN and had to be taken again
',nanIters)
1.61 fprintf('%-20s %-8d %-8d
','System',sum(erkIter),sum(hhIter))
1.62 fprintf('%-20s %-8d %-8d
','Failed solver calls',... 
1.63      sum(erkIter)-length(erkIter),sum(hhIter)-length(hhIter))
1.64 fprintf('%-20s %-8d %-8d
','ode calls',... 
1.65      sum(ode_erkIter),sum(ode_cellIter))
1.66

A.6 Intersection between systems

1. function out=solve_sys(t, states, erkSys, cellSys, erkTilde, cellTilde)
2. global iterMethod;
3.
4. %%% Solving according to the desired organization
5. erk = erkSys{1};
6. cell = cellSys{1};
7.
8. if(strcmp(iterMethod,'Jac'))
9.     out1 = erk(t, [], cellTilde, [],[],[]);
10.    out2 = cell(t, [], erkTilde, []);
11.    out = [out1;out2];
12. elseif(strcmp(iterMethod,'GSwErkfirst'))
13.    out1 = erk(t, [], cellTilde, [],[],[]);
14.    if any(isnan(out1))
15.        out = states*NaN;
16.    else
17.        [fracTilde, caiTilde] = feval(erkSys{2},out1);
18.        erkTilde = [fracTilde caiTilde*1e3];
19.        out2 = cell(t, [], erkTilde, []);
20.        out = [out1;out2];
21.    end
22. elseif(strcmp(iterMethod,'GSwCellfirst'))
23.    out2 = cell(t, [], erkTilde, []);
APPENDIX A. MATLAB CODE

    if any(isnan(out2))
        out = states*NaN;
    else
        [caFluxTilde] = feval(cellSys{2},out2,erkTilde(1,2)*1e−3);
        cellTilde = caFluxTilde;
        out1 = erk(t,[],cellTilde,[],[],[]);
        out = [out1;out2];
    end
    else
        fprintf('Wrong Organization');
    end
end


Bibliography


