

# **Characterization of Neuron Models**

A Thesis  
Presented to  
The Academic Faculty

by

William Boatin

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science in Bioengineering  
School of Electrical and Computer Engineering  
Georgia Institute of Technology

August 2005

# Characterization of Neuron Models

Approved by:

Dr. Robert J. Butera, Chair  
School of Electrical and Computer Engineering  
*Georgia Institute of Technology*

Dr. Robert H. Lee  
Department of Biomedical Engineering  
*Georgia Institute of Technology*

Dr. Kurt A. Wiesenfeld  
School of Physics  
*Georgia Institute of Technology*

Date Approved: July 5 2005

# ACKNOWLEDGEMENT

I thank God for all that I have been given and allowed to achieve. My sincere gratitude goes to my advisor, Dr. Bob Lee, for his knowledgeable insight and for always being approachable and supportive. I would like to express my appreciation to the members of my thesis committee; Dr. Rob Butera and Dr. Steve DeWeerth. This learning would not have been the same without the support, advice and friendship of the Lee Lab members; Chris Church, Estelle Graas, Sarah Jones, Philippe Karam, Autumn Schumacher, Nick Shapiro, Puja Thakker Randy Weinstein and all the undergraduate students who came and went. I would like to thank all in the NeuroLab for their friendship and support, as well as all my friends outside the NeuroLab, for reminding me there is life outside the lab. Thank you for everything Mum and Dad; your support and encouragement is well cherished. Thanks to my brother Kojo and my sister Adjoa for checking up on me. Finally, I would like to thank Doyin for her unconditional love and support, for keeping me focused and being my inspiration. Without her, this thesis would not have been completed.

Thank you, all.

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENT .....</b>	<b>III</b>
<b>LIST OF TABLES .....</b>	<b>V</b>
<b>LIST OF FIGURES .....</b>	<b>VI</b>
<b>SUMMARY .....</b>	<b>vii</b>
<b>INTRODUCTION AND BACKGROUND .....</b>	<b>1</b>
<b>METHOD AND TOOLS .....</b>	<b>6</b>
The Simulator.....	6
The Test Model.....	7
A Simulation Run .....	9
The necessity for a networked computer approach.....	10
The Inter-Process Communication Method.....	12
LAM/MPI .....	12
Porting from Windows to Linux.....	13
Code Design Process.....	14
The Beowulf Cluster.....	16
<b>RESULTS AND ANALYSIS.....</b>	<b>18</b>
CLUSTER ANALYSIS .....	18
PRINCIPAL COMPONENT ANALYSIS .....	29
<b>CONCLUSION.....</b>	<b>32</b>
<b>APPENDIX A MODEL PARAMETERS AND BASE VALUES.....</b>	<b>33</b>
<b>APPENDIX B MODEL BEHAVIORS AND BASE VALUES.....</b>	<b>36</b>
<b>REFERENCES .....</b>	<b>38</b>

# LIST OF TABLES

TABLE A1	GLOBAL MODEL PARAMETERS.....	34
TABLE A2	INITIAL SEGMENT PARAMETERS.....	35
TABLE A3	SOMA PARAMETERS.....	35
TABLE A4	DENDRITE PARAMETERS .....	35
TABLE A5	INTER-COMPARTMENT PARAMETERS.....	35
TABLE B.1	MODEL BEHAVIORS.....	37

# LIST OF FIGURES

Figure 1	Block diagram of neuron model layers .....	3
Figure 2	3 compartment model with structures .....	8
Figure 3	Master/Server Paradigm .....	11
Figure 4	Cluster tree of batch 1 data using average linkage method and Euclidean distance measure. Horizontal axis is distance (0 to 2); vertical axis is the run number (1 to 126).....	19
Figure 5	Cluster tree of uniform random correlation data using average linkage method and Pearson distance measure.....	20
Figure 6	Cluster tree of batch 1 data normalized by parameter and behavior scales, clustered using Pearson distance measure and average linkage method ....	22
Figure 7	Cluster tree of batch 2 data normalized by parameter and behavior scales, clustered using Pearson distance measure and average linkage method ....	23
Figure 8	Cluster tree of converging batch 1 data normalized by parameter and behavior scales, clustered using Pearson distance measure and average linkage method.....	24
Figure 9	Cluster tree of converging batch 2 data normalized by parameter and behavior scales, clustered using Pearson distance measure and average linkage method.....	25
Figure 10	Random cross-correlation matrices from cluster 1 in Figure 8 .....	26
Figure 11	Random cross-correlation matrices from cluster 2 in Figure 8 .....	27
Figure 12	Random cross-correlation matrices from cluster 3 in Figure 8 .....	28
Figure 13	Scree plot generated from PCA of base case parameters .....	29
Figure 14	Principal Component Loadings from PCA on base run data .....	31

# SUMMARY

Modern neuron models are more often than not large, complex entities. The ability to better simulate these complex models has been facilitated and driven by the development of ever more powerful and cheap computers. The capacity to manage and understand the models has lagged behind improvements in simulation ability and has done so almost from the inception of neuron modeling. Despite the computing power currently available, more powerful simulation platforms and strategies are needed to cope with current and next generation neuron models.

This thesis attempts to develop methodologies aimed at better characterizing and understanding motoneuron models. The hypothesis presented is that relationships between model outputs in addition to the relationships between model inputs (parameters) and outputs (behaviors) provide a characteristic description of the model that describes the model in a more useful way than just model behaviors. This description can be used to compare a model to different implementations of the same motoneuron and to experiment data.

Data mining and data reduction techniques were used to process the data. Principal component analysis was used to indicate a significant, consistent reduction in dimensionality in an intermediate, mechanistic layer between the model inputs and outputs. This layer represents the non-linear relationships between input and output. This implies that if the non-linear relationships of a model were better understood and accessible, perhaps the model could be manipulated by varying the mechanism layer

members, or rather the model parameters that primarily affect a particular mechanism layer member.

Hierarchical cluster analysis was used to show similarity between data generated by performing sensitivity analyses on models with random parameter sets. A main cluster represented the main region of model behavior with outlying clusters representing non-physiological model behavior. This indicates that the data derived from sensitivity analysis is in fact a good candidate for a metric of model validation.

The Introduction and Background section explores the reasons for this thesis, its relation to previous work, the methods employed to analyze the experimental data, predicted conclusions, and the significance of the expected results. In the Method and Tools section, the model used to develop validation methodologies, the simulator on which the model was simulated, and the software/hardware approaches used to generate output are illustrated. The outcome of the application of the analysis techniques are presented and discussed in the Results and Analysis section.

The results demonstrate the usefulness of cluster analysis in demonstrating the similarities between data used as a model characterization metric or model signature. Its application is also valuable in identifying the main region of useful activity of a model, thus helping to identify a potential ‘average’ parameter set. Furthermore, factor analysis proves functional in identifying members of the mechanism layer as well as the degree to which model outputs are affected by these members.

# Introduction and Background

Hodgkin and Huxley published their seminal paper in 1952, detailing a mathematical model of the electric current flow through the surface membrane of the giant nerve fiber of a squid [1]. The numerical integration and other calculations required to produce just one action potential took them months to calculate. However, these equations, the basis of most neuron models since the 1952 paper, were only partly based on theoretical observations; empirical curve-fitting was used to generate the rest of the equations. This implies that the mechanisms that produce the neuronal output were not completely described, and thus any models which rely on these equations has underlying mechanisms which are not fully understood.

Due to this ignorance, modelers have long faced a difficult issue tuning model parameters to achieve specific output. Most qualitatively tune model parameters [2, 3]. The qualitative approach focuses on the minimal model necessary to study a particular aspect of neuron activity. Although encouraging results have been achieved with this approach, it suffers from parameter fragility; even slight deviations from base parameters can cause the model to malfunction. Moreover, qualitative models produce expected output but possibly as a result of incorrect mechanisms.

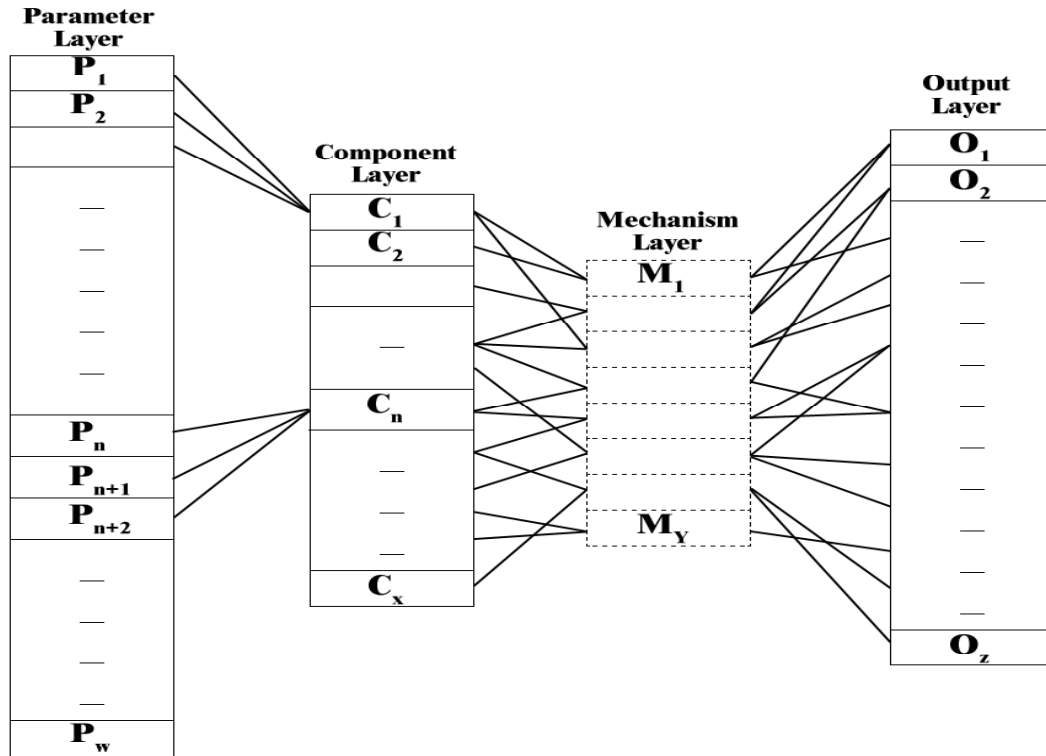
Different models of the same type of neuron, given the same input, produce similar output, but clearly do so in different ways [4-6]. Comparison of input and output is not enough to differentiate between such models. Reproducing experimental

phenomena, which is typically done as a validation method [7, 8], does not sufficiently validate the mechanisms responsible for output. There are few existing techniques to determine which models are physiologically correct. Reproduction of experimental data verifies that a model produces the desired output, but does not validate the mechanisms that generate the output. Insufficient validation methods limit the use of neuron models as cheaper and more accessible alternatives to the biological model. It is consequently apparent that tools for validating neuron models would be beneficial [4].

Yet another concern for modelers is the non-uniqueness of the parameter space. With the invention and rapid development of the electronic computer, much larger and more complex models can be created and simulated. The explosion in the number of parameters available for tuning, driven by the increase in model complexity that electronic computers facilitate, has necessitated automatic parameter search techniques, and prompted exploration of the parameter space. This exploration has exposed the problem of non-uniqueness; several different parameter sets can yield the same output [9]. This problem makes drawing conclusions based on differences in parameter sets difficult. Research has been conducted on comparing several approaches to automatic parameter search [10]. However these approaches focus on parameter generation from experimental data [11]. As a result, most neuron models are still hand-tuned [12].

This thesis examines a hypothetical dimension of model data in an attempt to formulate automatic validation tools for neuron models. The premise is that a mechanistic layer exists between the component layer (input) and the behavior layer (output) of a model that forms a restriction of activity (see Figure 1). The ineffectiveness of the parameter tuning methods stems from the ignorance of this layer, the inability to identify

this layer, but more tangibly the failure to identify the parameters that affect each member of this layer.



**Figure 1:** Block diagram of neuron model layers

Sensitivity analysis of the model yields a table of slopes, relating model inputs to model outputs. Because it relates inputs and outputs, it is reasonable to expect that slope data would provide a view of the mechanism layer when probed with suitable techniques.

Principal component analysis (PCA) is proposed as a technique to determine the number of members in the mechanism layer and the parameters that most affect a

particular mechanism layer member. PCA is an ideal analytical tool for probing the underlying factors that influence a set of data. PCA reduces dimensionality in data by assigning variance to hypothetical factors. Most of the variance in the data can be explained by fewer hypothetical factors than original measures. The influence of each original factor on the hypothetical principal components is also determined. PCA has previously been used to describe neural morphology [13-16] and to extend competitive learning neural models [17]. In this thesis, PCA is used to expose the number of members in the mechanism layer, and indicate which model outputs are affected by a particular principal component.

The slope data can also be used as a characterization metric. However, its dependence on model parameters, some of which are inaccessible experimentally, and the fact that its size increases as models and their parameter sets grow larger, make raw slope data unsuitable as part of the model character. A better candidate for a model character metric is the cross-correlation matrix of slope data, correlating on the outputs. Outputs that are affected by parameters to a comparable degree are highly correlated. Thus Models that claim to be alike, both in terms of output and mechanism should have similar cross-correlation matrices.

Cluster analysis encompasses numerous algorithms and methods for categorizing objects. These similar groupings are found such that the degree of similarity between two objects in the same group or cluster is maximal. Cluster analysis has been used to analyze microarray data [18], differentiate genes [19], to separate stained neurons into different clusters [20] and in neuroimaging [21]. Cluster analysis of slope data should show the degree of similarity between cross-correlation data from a single model. A high degree of

similarity would indicate that the notion of cross-correlation data as a characteristic of a model is valid.

Hierarchical cluster analysis will also be used to address the non-unique parameter space issue. Analyzing the cross-correlation matrices of model that gave the same output with different parameter sets should yield several clusters; it is believed that the main cluster is that in which the model is behaving 'physiologically' and that outlying clusters represent the model exhibiting non-physiological behavior.

The next section describes the test model, simulator, and hardware platform and software implementation used to generate the data used for cluster and principal component analysis.

# Method and Tools

## The Simulator

The simulator on which the test model was run implemented a finite-element approach to modeling. This approach involved dividing the problem domain into smaller sub-domains, in which the differential equations are approximately solved. Each sub-domain or region represented part of the biological neuron. The equations for each region are solved and assembled to give a solution for the entire domain. Other simulators like the popular NEURON implement a cable-theory approach to representing neurons [22].

The simulator implemented an adapted Powell conjugate directions method for searching [23]. The algorithm collected and used derivative information during the search and generally moved one parameter at a time, except in the case of a repeated parameter, where instead, a conjugate direction was created that would point to the minimum of a parabolic model space. Single parameter searching resumed once this conjugate direction was exhausted. The modification mainly involved the choice of parameters between conjugate directions; instead of serially searching parameters, a best parameter was used, based on stored derivative information.

An important concern when performing a search was the scale value of a behavior, typically a fraction of the behavior value. The smaller the scale was, that is, the finer the resolution, the smaller was the distance moved by the simulator per iteration. This resulted in a search that converged to a value with greater accuracy, or less error, but that consequently required more iteration for convergence. The precision of a behavior,

used to set the scale value, was basically a measure of the error involved in measuring the behavior value, essentially the smallest possible distance the simulator could move in any direction along that measure. Ideally the scale was set to the precision. However, the precision was usually several orders of magnitude smaller than the behavior base, and so if used to set the scale resulted in a search that did not converge within the given iterations of the simulator.

Convergence was determined by summing the sum of the square error of the behavior goals. By default, the error limit was the number of behaviors. To converge, the sum of the square of the error of behaviors had to be less than the number of behaviors.

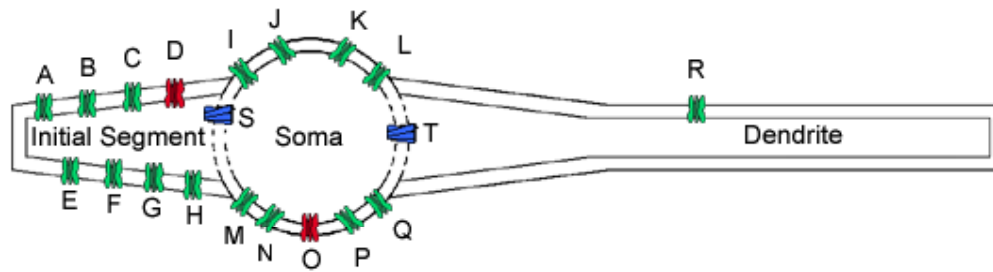
The simulator allowed one to perform sensitivity analysis on model parameters. Sensitivity analyses generated the data proposed as a metric of model character. After the calculation of sensitivity analysis for all parameters, the precision value for each behavior was available.

## **The Test Model**

A lot of early neuroscience modeling was done using motoneurons from the lumbar region of the spinal cord of the adult cat [2, 24, 25]. The availability of information on this type of neuron made it a good candidate for modeling studies, for comparison reasons. Any (neuron) model would have sufficed for this thesis, but because of the potential for comparison, a motoneuron model of the type described was used.

The test model had three compartments: the initial segment, soma and dendrite (see Figure 2) with 65 model parameters, shown in Tables A1, A2 and A3, and 42 model behaviors, shown in Table B1, available for manipulation and recording. The behaviors

were defined by the protocols run on the model (see Table B1). Each compartment had ionic channels that gave them characteristic properties of that specific region of the motoneuron (see Tables A1, A2 and A3).



**Figure 2:** 3 compartment model with structures: A - CA HVA channel; B – CA L-Type channel; C – Ca leak; D – Na-Ca pump; E – K Ca channel; F - Kf channel; G – K leak; H – Nav 1.6 channel; I – CA L-Type channel; J – Ca HVA channel; K – Ca leak channel; L – H channel; M – K(Ca) channel; N – Kf channel; O – Na-Ca Channel; P – K Leak; Q – Nav1.1 channel; R – K leak channel; S – Initial segment/Soma capacitance; T – Soma/Dendrite capacitance

The model had a default or base parameter set. This set was chosen so that the model produced the desired output after software experiment protocols were run on it. Thus the parameters were not obtained or derived from experimental data. However, since the model is of a motoneuron and was created to model a real motoneuron, the model can claim to be realistic enough that the base parameter set should have a strong similarity to actual biological model parameters. Running the protocols with these base parameter values generated a base set of behaviors. Each behavior had a precision value, representing the smallest possible distance that the simulator could move in any direction for that particular behavior. Each behavior also had a scale value that was the actual

smallest distance moved by the simulator for a particular behavior. This allowed the user to tune the accuracy with which the simulator conducted searches.

As in a real experiment in which an electrode is used to manipulate and read signals from the neuron, an electrode was used to implement and read data from the compartment in which it was defined according to a protocol. Protocols defined the characteristics of the electric current, electric voltage, signals recorded and other electrode or experiment conditions required to generate and record specific data [26, 27]. The protocols used in this thesis, listed in Table B1, were software protocols meant to duplicate protocols used in actual experiments, facilitating and simplifying comparison between model and experiment output. 7 protocols giving 42 behaviors were run to increase the output dimensionality thus giving the model several degrees of freedom within which to settle.

### **A Simulation Run**

A complete simulation run meant the following: The base case was run on the model with default parameters, resulting in initialized base behaviors. The behavior goals were set equal to the behavior bases, then the behaviors bases were set to zero. A random parameter set was then imported into the model. A sensitivity analysis was run, generating behavior precision values. The behavior scale was set equal to the precision for that behavior or 10% of the behavior goal if the precision was less than 10% of the behavior goal. A second run was conducted with greater precision; the scale was set equal to the precision or 5% of the behavior goal if the precision was less than 5% of the behavior goal. For both runs, a search was conducted in which the behavior base was

perturbed to approach the behavior goal, followed by a final sensitivity analysis. 90% of the searches conducted with the higher scale value (10% of behavior goal) were expected to converge. 125 simulation runs using random parameters were done for each precision level, a total of 250 simulation runs.

### **The necessity for a networked computer approach**

Performing just one simulation run took up to two days. Sequentially performing 250 simulation runs would take approximately 500 days. A more time efficient simulation method or platform had to be developed in order to collect data in a reasonable time frame.

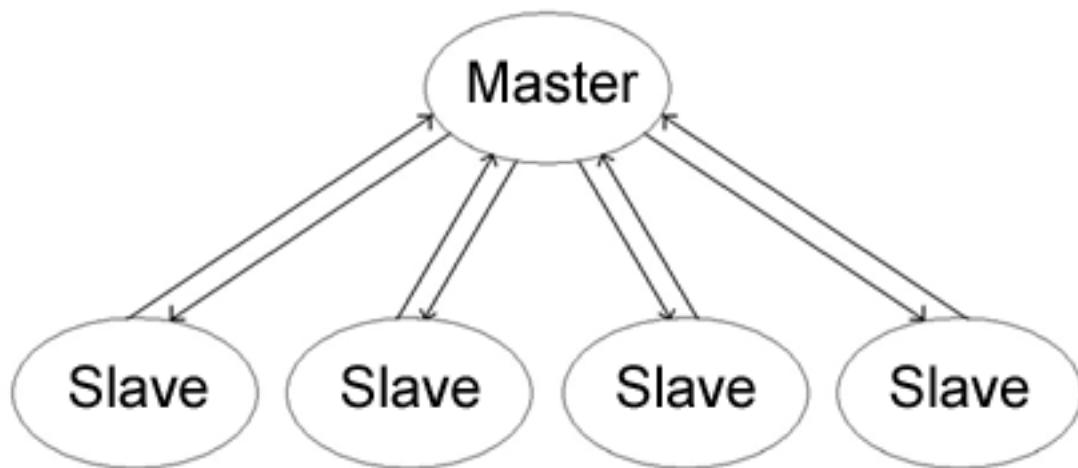
Previous research has shown that a current field programmable gate array (FPGA) is a quicker simulation platform than current computer general-purpose computers, also showing that several models can be run simultaneously on an FPGA by interleaving the different model simulations [28]. However, general-purpose computers are more readily available than FPGAs and are much more versatile in their usefulness. In addition, the conversion of the test model into code understandable by the FPGA is non-trivial. A Beowulf cluster was used to conduct simulation runs simultaneously.

A Beowulf cluster is a group of usually homogenous personal computers running a version of UNIX, connected by a TCP/IP network [29]. The cluster has software installed that allows for shared processing, that is, the splitting up of a task into sub-tasks among several processors, each running one or more processes committed to completing the sub-task in an effort to speed up the total execution time of the main task. Each

personal computer in the cluster is called a node, each task referred to as a job. A thread of code currently being executed on a node is called a process.

Rather than starting individual simulation runs on each node in the cluster, a more managed approach was taken involving some communication between processes performing simulation runs. This allowed for better control of the number of simulation runs conducted, the models used in each simulation run, special input that a model needed and better, consistent organization of the simulation run output.

The communication model implemented was the master/slave or client/server paradigm (see Figure 3). A dedicated master process sent data to a user-specified number of slave processes and was also responsible for deciding when all the required processing had been completed. The slave processes performed the simulation runs and communicated completion of a run to the master process.



**Figure 3:** Master/Server Paradigm

## **The Inter-Process Communication Method**

A crucial feature of this strategy was the communication method used between processes on different machines on the network. Candidate communication strategies were file based communication, shared memory, memory-mapped files and TCP communication.

The only acceptable method was TCP/IP communication. All the other methods suffered from deficiencies that eliminated them as communication choices: file based communication was too slow and required a shared-file system across a network which was not guaranteed; shared memory was quick and efficient but would not work across the distributed memory on the cluster; memory-mapped files suffer from both shortcomings of file-based and shared memory communication. Existing implementations of a TCP/IP communication method were evaluated.

## **LAM/MPI**

The Message Passing Interface (MPI) standard, developed by the MPI forum, is a library specification for message-passing. It was developed as a way to process parallel algorithms on computer clusters, allowing processes to communicate by sending messages among each other. The implementation chosen for this thesis was Local Area Multicomputer/Message Passing Interface (LAM/MPI) developed by and freely available from Indiana University [30, 31]. LAM/MPI had excellent support available through an informative website, useful FAQ list, well-patronized mailing-list with prompt responses, good documentation and numerous tutorials. LAM/MPI is compatible with Portable

Batch System (PBS), the batch queuing system through which all jobs on the Beowulf cluster had to be scheduled.

### **Porting from Windows to Linux**

The original simulator was written in Pascal and intended to run on a Microsoft Windows (Windows) platform. However to take advantage of LAM/MPI, the simulator had to be ported to a Linux platform. Though there are implementations of the MPI standard that run on the Windows platform, the decision was made to port the simulator to Linux because of its greater stability, and because the target cluster was a Beowulf cluster.

The Windows version of the simulator used a graphical user interface (GUI). The ported simulator, however, was converted to use a command line interface because some of the components used in the GUI code were available in Windows only. These and all graphical components were removed from the Linux version with no impairment to core simulator functionality. Without a GUI, the ability to visually track simulation progress was lost. Debugging code was added to all functions to display varying levels of information about the status of the running simulation. The quantity and type of information printed to the screen was controlled by a command line switch.

In the Windows version simulator, all parameter and output manipulation was done by the user, either by directly manipulating the values or by importing from a text editor or Microsoft Excel. All the data in the simulation runs in the Linux version simulator were to be generated automatically, requiring new functions to be written to the data generation and transfer. In addition, in the Windows version the user decided

whether or not to save output. Since the Linux version was designed to run unattended, all output was saved at various points during the simulation run and at the completion of a run. Unique names, based on process ID (PID) were automatically generated so that no data was overwritten accidentally.

C, C++ or Fortran code is required to make use of LAM/MPI libraries. However the Linux version simulator was written in Kylix. Unsuccessful attempts were made to reference the LAM/MPI libraries from the Kylix code. The dilemma was resolved by turning the simulator into a shared object, the Linux equivalent of a Windows dynamic-link library (DLL), exposing functions that allowed simulation to be performed. The C code that linked to and used the LAM/MPI libraries also linked to the simulator shared object making it possible to run the Kylix simulator to take advantage of LAM/MPI messaging. The simulator shared object, the LAM/MPI libraries and the executable C code that sent model and processing information via MPI all had to present on each machine on the cluster. Scripts were created that kept the source code, compiled code, and the simulator shared object up-to-date on each machine on the cluster.

### **Code Design Process**

Message passing between processes was managed using the master/slave model (see Figure 3). The implementation of this model required clearly defining the roles of master and slave processes.

In order to dispense data to the slaves, the master process required the number of simulations to run and the models to use. This information was supplied via an ASCII input file. The master also accepted command line arguments which controlled the

amount of debug information displayed on the screen, the percentage by which the random parameter set was perturbed from the base parameter set, and the minimum percentage of the behavior goal that the behavior scale had to be.

Using the base parameter set stored in the model, the master created random parameter sets. Each parameter was varied in both incrementing and decrementing directions by a supplied random percentage of the parameter base. A message structure was created, incorporating the original and perturbed parameter set, model behavior, model name, debug and tracking information, and also the action to be performed on the data. All the messages to be processed by slaves were placed in an array. The master process then sent the next message in the array to the next available slave until slave has data to process. The master then checked the status of each slave and sent the next available message to the next slave to acknowledge completion of processing. When all messages were sent and all expected acknowledgments had been received, the master signaled each slave to terminate, and then in turn self-terminated.

Upon receiving a message, a slave process would parse the message and identify the action to be taken. After completing the action, the slave would wait for the next message, performing actions until an action indicated an end of processing, upon receiving which the slave would terminate. Example tasks include starting a new simulation, continuing a previously started simulation, redoing a simulation with different tolerances, or terminating. Simulation tasks were carried out by making library calls to the simulator shared object.

LAM/MPI supports various modes of communication between processes. For the purpose of this thesis, the basic point-to-point communication mode was used. The

sending process would send a message only to a particular process. Likewise, the receiving process expected and would receive a message from only a specific source. A blocking send or receive call does not return until the call has completed. A non-blocking send or receive call returns right away, but requires a check of completion. The master process used a mixture of blocking and non-blocking sends and receives. To send messages to a slave process, a blocking send was used. Non-blocking receives were used to receive messages from slaves, freeing the master to send more messages to waiting slaves, and to check for completion from all processing slaves. Each slave used a blocking send to send messages to the master and a blocking receive to receive messages from the master ; outside of library calls to the simulator shared object, they had no other processing to tasks.

LAM/MPI separates the number of CPUs or processors from the number of processes running on these processors. Each processor is called a node, and each node can have any number of virtual CPUs assigned to them. This allows more than one process to run on a node, allowing advantage to be taken of faster or more powerful CPUs. It is up to the user to decide what the optimal number of processes is for each node. For this thesis, this optimum number was found to be processes per a 1GHz CPU. Since the master process spent the majority of time waiting for slaves to finish processing it ran in conjunction with 2 slaves.

### **The Beowulf Cluster**

Due to issues concerning PBS and LAM/MPI, the target Beowulf cluster was unusable. The development and test network of machines, converted into a heterogeneous

Beowulf cluster by LAM/MPI, was used to perform all of the simulation runs. A total of 7 machines were available, but no more than 5 were in use at any time as some of the machines were needed for other purposes. All machines ran the latest version of SUSE Linux, with the minimum required packages installed for LAM/MPI, Borland Kylix, SCP, SFTP and Mozilla Firefox to work. On this heterogeneous network, using the method outlined, total simulation time was about 38 days.

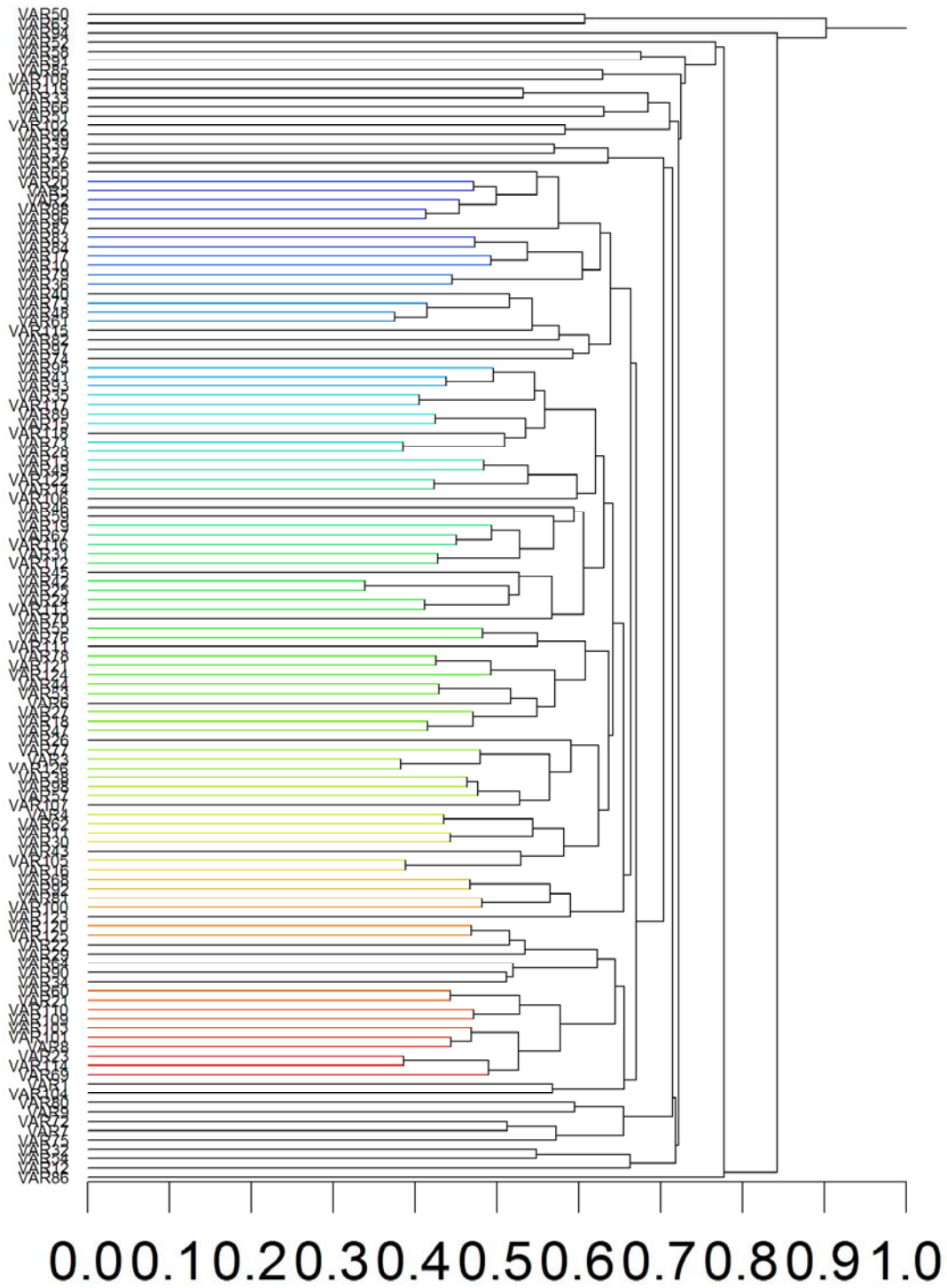
# Results and Analysis

A total of 125 simulation runs (batch 1) were performed with a behavior scale at least 10% of the behavior goal. Of the 125 simulation runs, 118 (94.4%) of these runs had successfully converging searches. Another 125 simulation runs (batch 2) were performed with a behavior scale at least 5% of the behavior goal. Of these runs, 55 (44%) of these runs had successfully converging searches. For each run the partial and final error, parameter, behavior, residual and slope values were recorded.

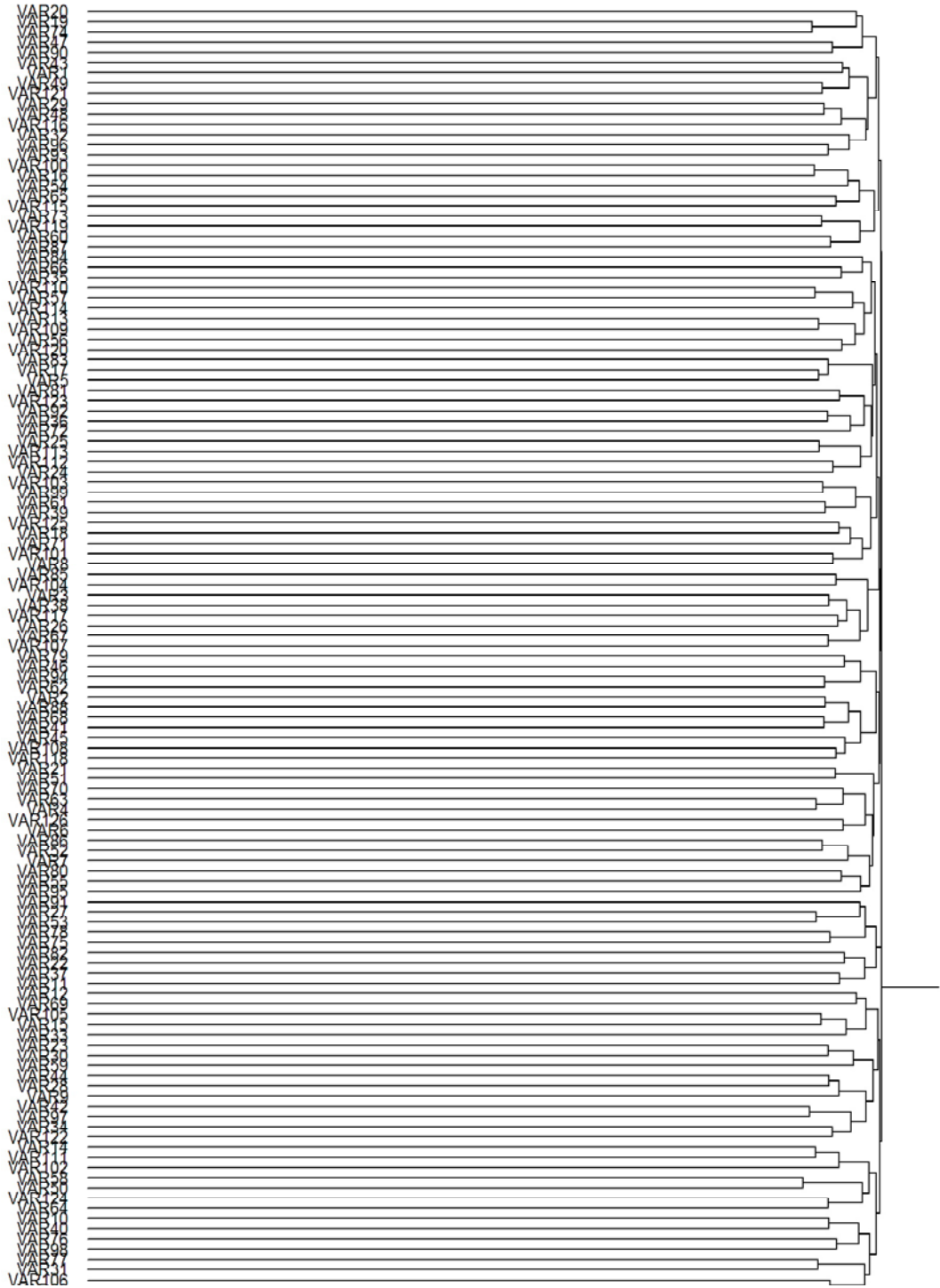
A cross-correlation matrix, correlating on the behaviors, was generated from each 42x65 slope matrix for each simulation run. Behaviors that resulted in null or undefined values were removed. Each of these matrices was unrolled and vertically concatenated to form a 1600x126 element array. The base case data was added to the randomly generated data as the 126<sup>th</sup> run.

## **CLUSTER ANALYSIS**

Cluster analysis was performed on each resultant matrix, with the clustering performed on the columns (simulation runs). The resultant cluster tree for batch 1, shown in Figure 4, was created using Pearson distance and the average linkage method. The tree does not show any clearly discernable clusters. However in comparison to randomly generated correlation data (Figure 5), there is some definite organization.



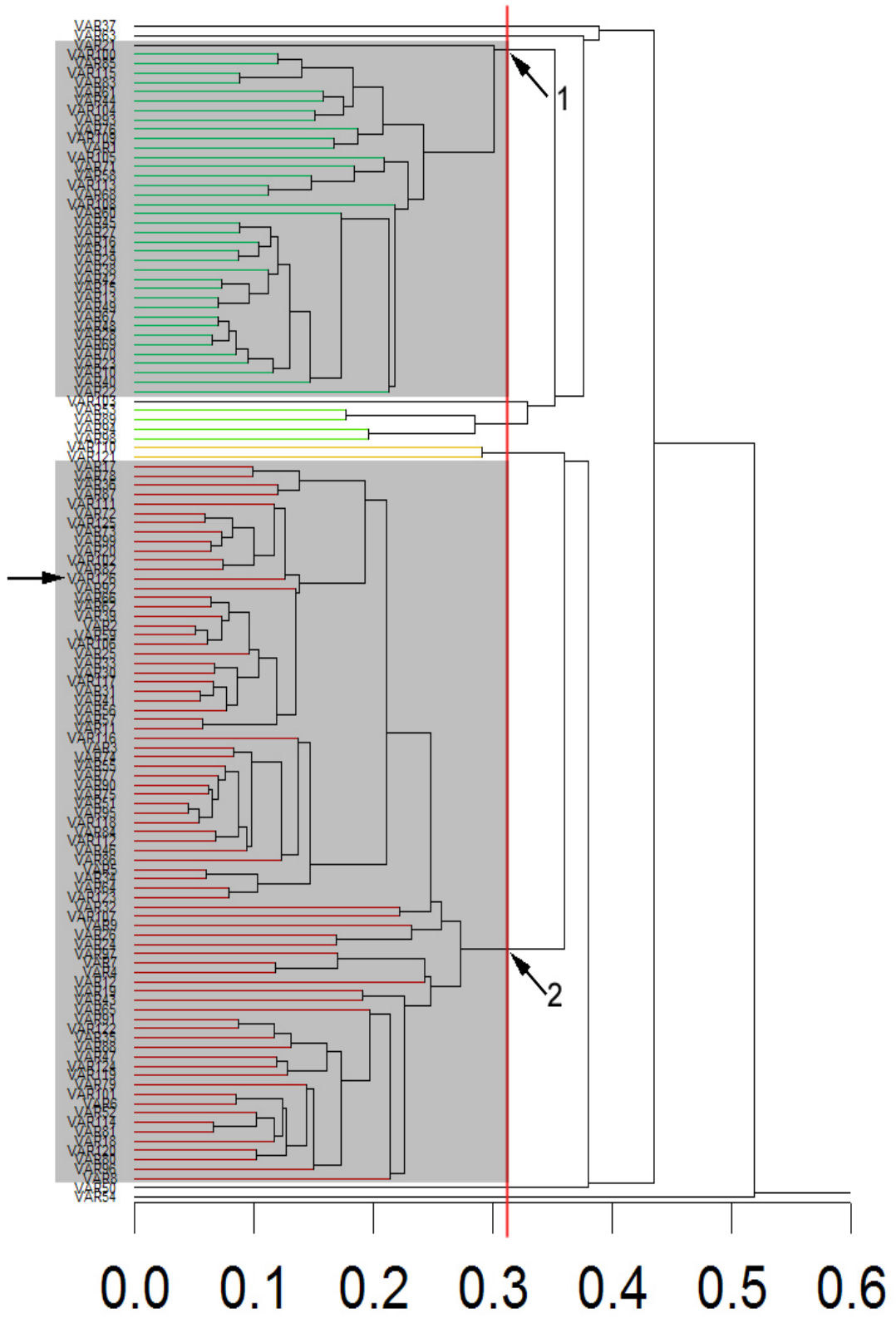
**Figure 4:** Cluster tree of batch 1 data using average linkage method and Pearson distance measure. Horizontal axis is distance (0 to 2); vertical axis is the run number (1 to 126)



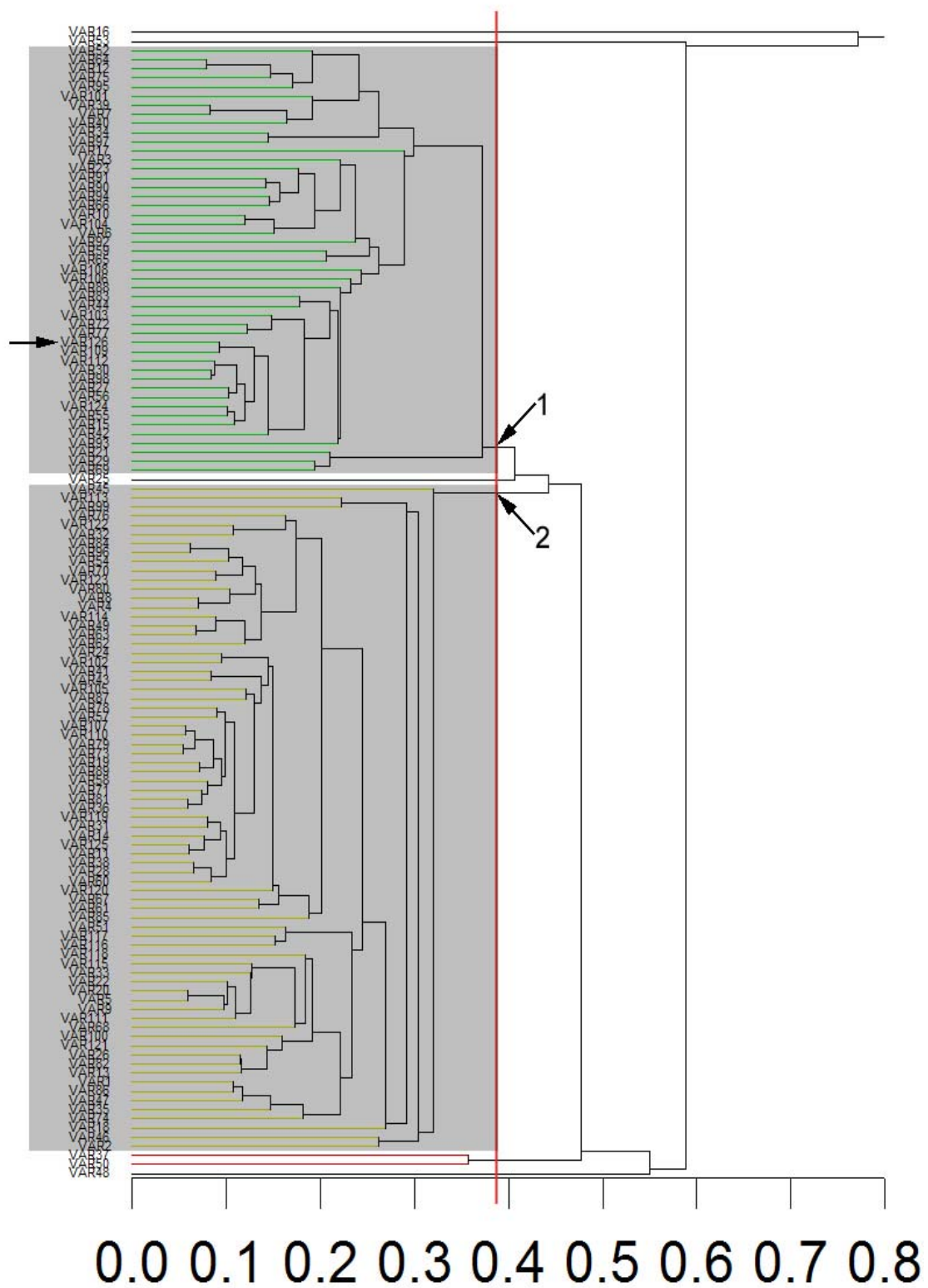
**Figure 5:** Cluster tree of uniform random correlation data using average linkage method and Pearson distance measure

To improve the disposition of the clusters, the data was normalized by the relevant behavior and parameter scale. From this data, cross-correlation matrices were created as previously described. The results for batch 1 are shown in Figures 6. The arrow on the horizontal axis points to the base case run. The red vertical line is the chosen line for cluster identification, yielding the 7 clusters indicated by the arrows within the figure. The cluster line was chosen to yield the minimum number of clusters yet simultaneously clearly identifying a main cluster. The base case run is located within cluster 5, the main cluster. The equivalent cluster tree for batch 2 is shown in Figure 7.

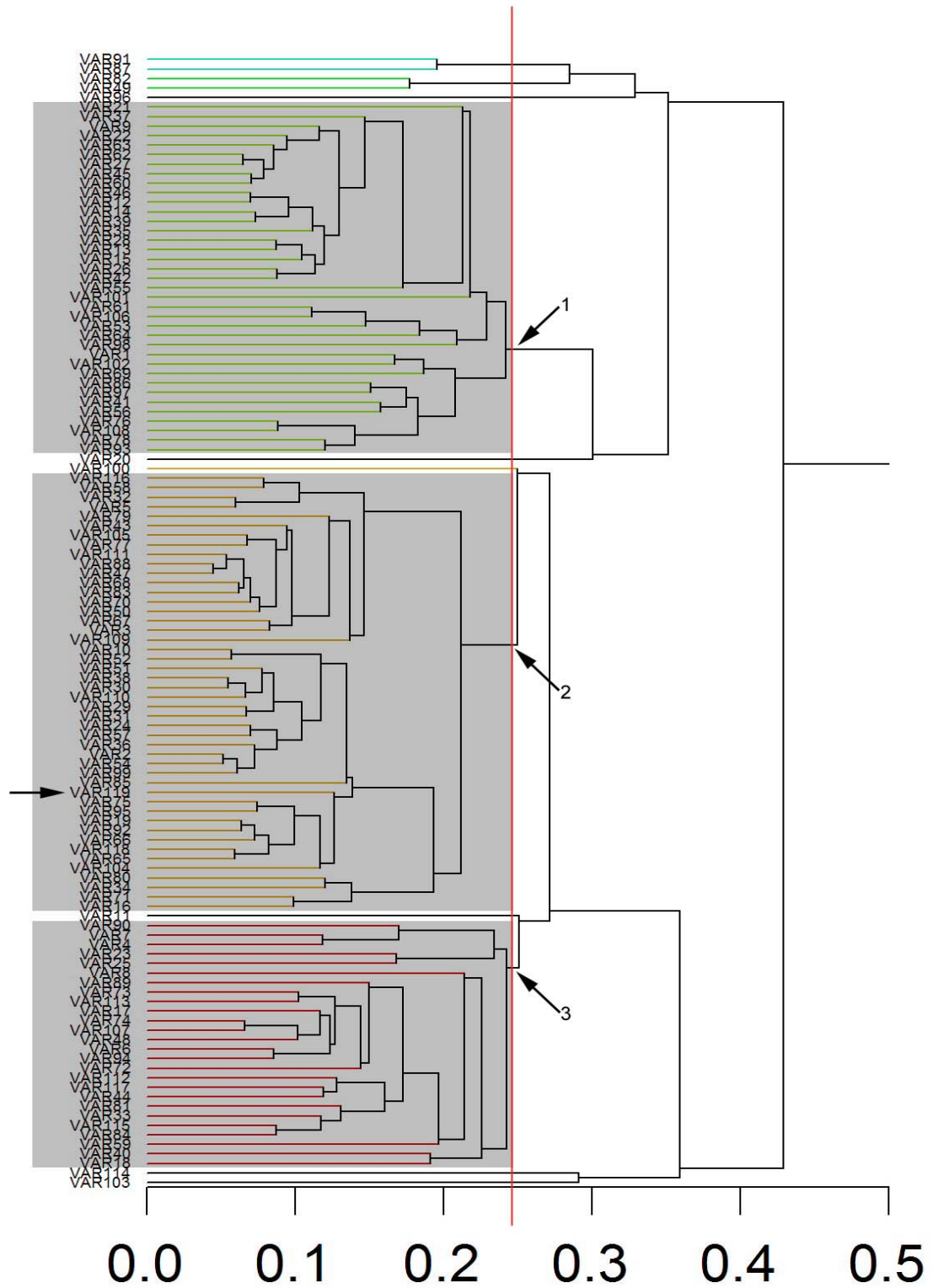
The data from simulation runs with non-converging searches were removed from consideration, resulting in Figure 8 for batch 1 and Figure 9 for batch 2. These trees show clear clusters with clustering starting at relatively short distances, indicating that simulation runs in each cluster are very similar. The base case run is approximately in the middle of a large cluster. As the base case parameters were chosen so that the resulting output resembled biological motoneuron output, this observation reinforces the hypothesis that the main cluster, by definition the one that contains the base case data, identifies cases where the model was exhibiting ‘physiological’ behavior. However, in Figure 8, there are three large clusters, and although the base case is in the largest cluster, the three clusters are all relatively the same size. In Figure 9, the largest cluster does not contain the base case. This means that there are several regions in which the model behaves ‘physiologically’, implying a non-convex parameter space. The manner in which the base parameter set was chosen, as described above, does not guarantee that it would lie in the largest cluster, but implies that it would lie in a cluster large enough to be identifiable as one of several main clusters, given a reasonable model.



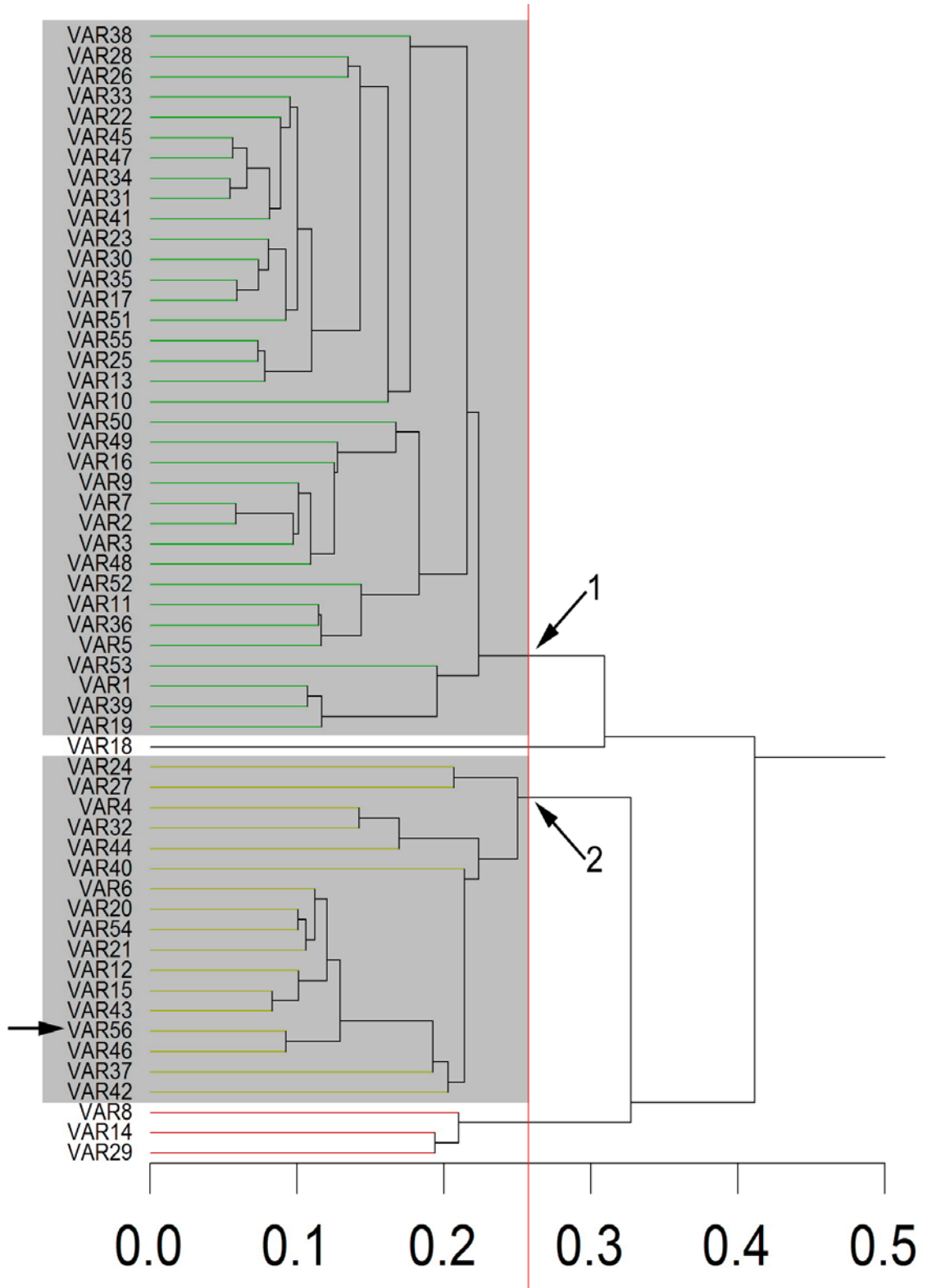
**Figure 6:** Cluster tree of batch 1 data normalized by parameter and behavior scales, clustered using Pearson distance measure and average linkage method



**Figure 7:** Cluster tree of batch 2 data normalized by parameter and behavior scales, clustered using Pearson distance measure and average linkage method

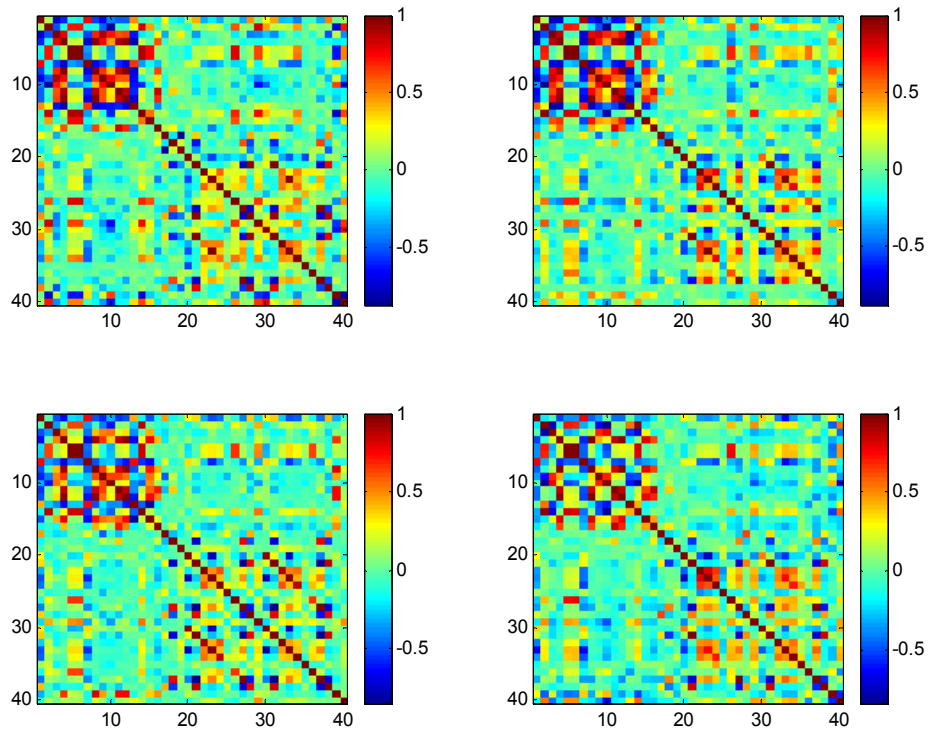


**Figure 8:** Cluster tree of converging batch 1 data normalized by parameter and behavior scales, clustered using Pearson distance measure and average linkage method



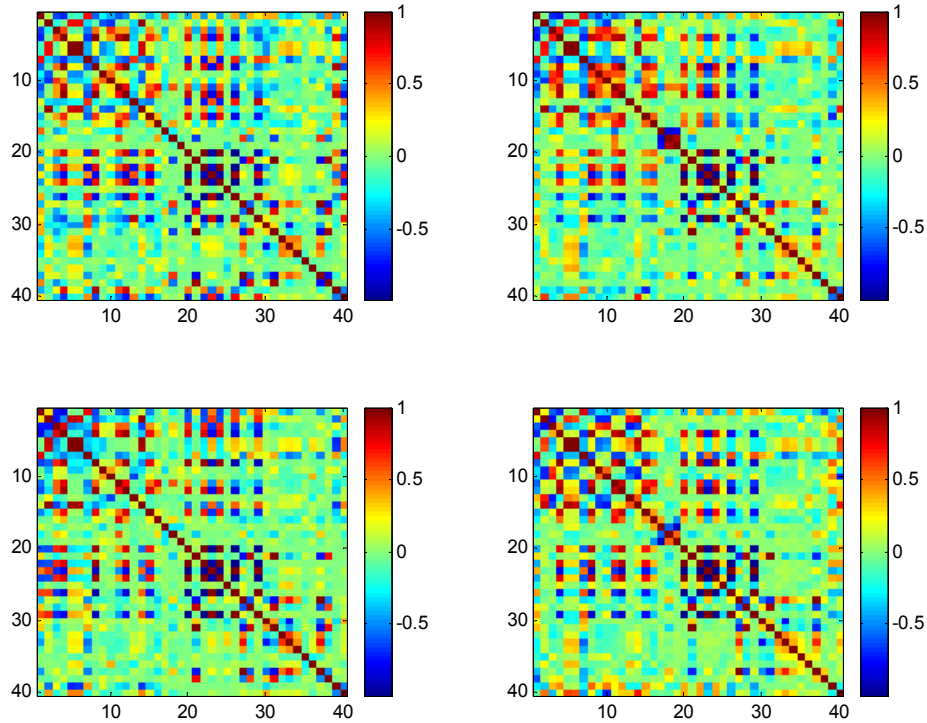
**Figure 9:** Cluster tree of converging batch 2 data normalized by parameter and behavior scales, clustered using Pearson distance measure and average linkage method

The emergence of clusters means that similar groups of cross-correlation data are being identified. Images of correlation matrices support this claim. Figure 10 shows 4 randomly picked correlation matrices from cluster 1 in Figure 8. It is obvious that the images are similar. The same color map was used to plot all images and is shown beside each image. Blue colors represent small values; red colors (e.g. the diagonal values) represent large numbers.



**Figure 10:** Random cross-correlation matrices from cluster 1 in Figure 8

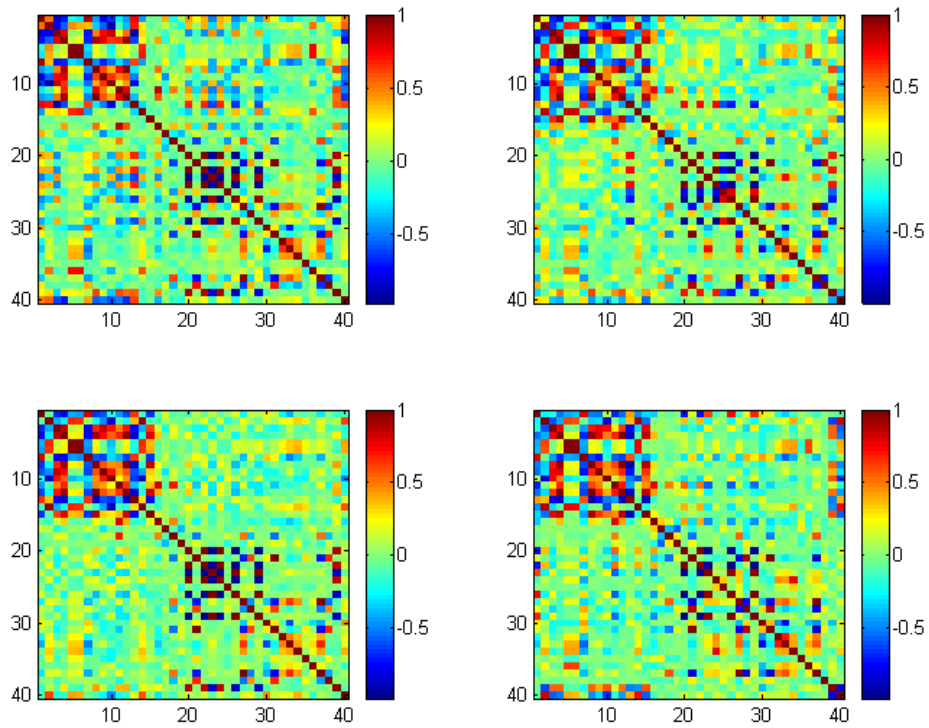
Figure 11 presents 4 random correlation matrices from cluster 2 in Figure 8, the main cluster. Again, there is readily apparent similarity in these images and it is equally clear that they are dissimilar from the images in Figure 10.



**Figure 11:** Random cross-correlation matrices from cluster 2 in Figure 8

Figure 12 portrays 4 random cross-correlation matrices from cluster 3 in Figure 8. These again are quite similar to each, and dissimilar from the images in Figure 10 and 11. However, all the cross-correlation matrices can be considered similar in the absolute sense. Ignoring outliers, in Figures 8 and 9, all the cross-correlation matrices look similar at a distance of 0.5 out of a maximum distance of 2. In general, the further out from the

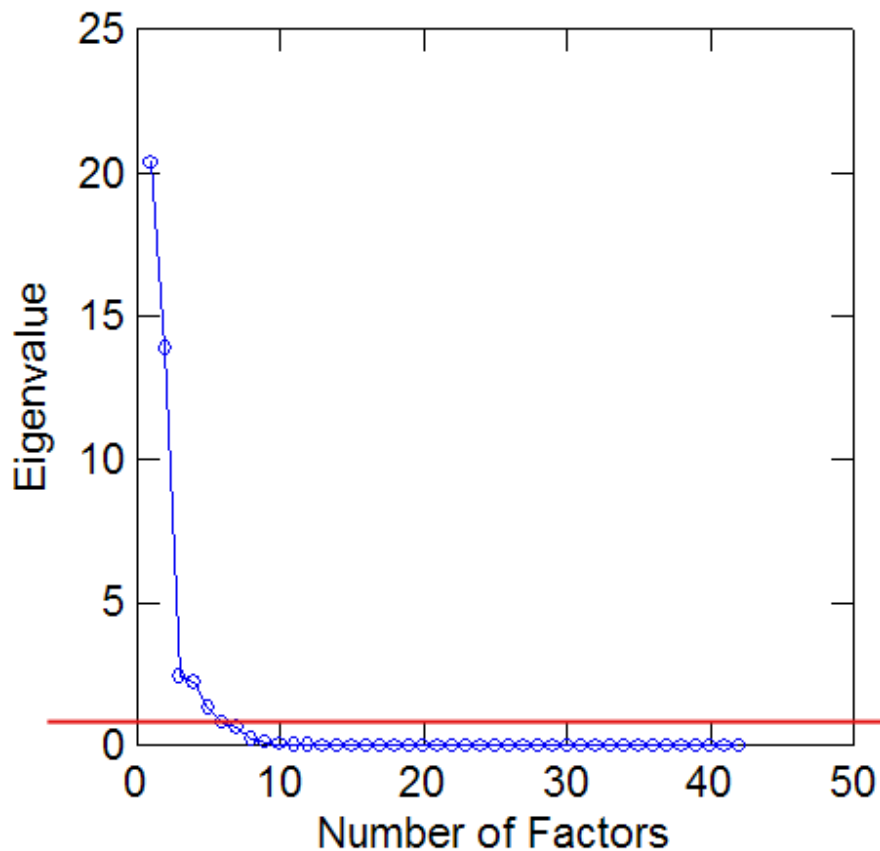
origin of the horizontal axis that clusters form, the more dissimilar are the members in converging clusters. The fact that all the clusters have all but converged at a distance of 0.5 imply that the correlation matrices were quite similar. The images in Figures 10, 11 and 12 support this statement.



**Figure 12:** Random cross-correlation matrices from cluster 3 in Figure 8

## PRINCIPAL COMPONENT ANALYSIS

PCA was performed on the slope data from the members of the main clusters in Figures 8 and 9 using varimax rotation, a perpendicular rotation method that minimizes the number of variables that have high loadings on each factor, simplifying the interpretation of the factors. Figure 13 shows the results of such an analysis. The number of factors is subjective as is the case with hierarchical cluster analysis. A minimum eigenvalue of 1 was used as the lower threshold for a factor. The number of factors ranged from 5 to 6. The base case slope data, shown in Figure 13, indicated 5 principal components. The red line is the eigenvalue threshold line.



**Figure 13:** Scree plot generated from PCA of base case parameters

The factor loadings for the base case are shown in Figure 14. Reds indicate high factor loadings and blues indicate low factor loadings. The absolute value of the slope data was used so that high negative and positive loadings are both red. The factors on the left are the model behaviors and are numbered in order of the behaviors in Table B.1. Data is sorted so that the behaviors that maximally affect a principal component (PC) (numbered at the bottom of the figure) are listed in descending order of effect. The variance represented by each PC is shown near the bottom of the table and shows that the first 2 PCs account for most of the variance in model output. If the PCs represent the members of the mechanism layer, the implication is that only two mechanisms account for most of the variation in model output.

Factor32					
Factor39					
Factor8					
Factor24					
Factor9					
Factor27					
Factor28					
Factor7					
Factor23					
Factor6					
Factor34					
Factor18					
Factor35					
Factor16					
Factor33					
Factor3					
Factor21					
Factor15					
Factor22					
Factor17					
Factor1					
Factor10					
Factor12					
Factor19					
Factor5					
Factor13					
Factor25					
Factor31					
Factor30					
Factor36					
Factor37					
Factor29					
Factor14					
Factor20					
Factor2					
Factor26					
Factor42					
Factor11					
Factor4					
Factor38					
Factor41					
Factor40					
Variance	20	14	3	2	1
Percent	48	33	6	5	4
PC	1	2	3	4	5

**Figure 14:** Principal Component Loadings from PCA on base run data

# Conclusion

A computer cluster was successfully used to generate large quantities of data in reasonable time, establishing their effective use as time efficient platform for large volume simulation.

Hierarchical cluster analysis was demonstrated to be a useful tool in comparing model signatures. The cross-correlation of slope data generated from sensitivity analysis is a good candidate for a model signature. Cluster analysis also helped to address the problem of a non-unique parameter space by identifying the regions of model behavior in which the model is believed to behave correctly.

Principal component analysis identified components that could represent the members of the mechanism layer. PCA also identified which model outputs were affected by individual components and to what degree model outputs are affected by these components. More research and analysis is needed to further describe the identified components and provide a physiological connection between model outputs and mechanism layers by which the model outputs are affected.

# **Appendix A**

## **Model Parameters and Base Values**

**Table A1: Global model parameters**

<b>Parameter Name</b>	<b>Units</b>	<b>Base Value</b>
Ca HVA Channel\hs	mV	9.5
Ca HVA Channel\htau	Msec	38
Ca HVA Channel\hVh	mV	37
Ca HVA Channel\ms	mV	6
Ca HVA Channel\mtau	Msec	1
Ca HVA Channel\mVh	mV	-10
Ca L-type Channel\ms	mV	7.1
Ca L-type Channel\mtau	Msec	38
Ca L-type Channel\mVh	mV	-25
K Ca Channel\mb		38
K Ca Channel\md		2500000
K Channel\ms	mV	23.8
K Channel\mVh	mV	-39
K Channel\tmax	Msec	9.8
K Channel\tmin	Msec	0.98
K Channel\ts	mV	6
K Channel\tVh	mV	-10.6
Na-Ca Pump\Tau	Msec	14.5
Na-K H2 Channel\ms	mV	-6.4
Na-K H2 Channel\mVh	mV	-74
Na-K H2 Channel>Selectivity		0.75
Nav1.1\h1ss		7
Nav1.1\h1Tau	Msec	0.2
Nav1.1\h2ss		-5
Nav1.1\h2Tau	Msec	0.19
Nav1.1\m1Vh	mV	-22
Nav1.1\m2Vh	mV	-117
Nav1.1\mtau	Msec	0.005
Nav1.1\mVs	mV	11.6
Nav1.6\h1ss		7
Nav1.6\h1Tau	msec	0.05
Nav1.6\h2ss		-5
Nav1.6\h2Tau	msec	0.05
Nav1.6\m1Vh	mV	-22
Nav1.6\m2Vh	mV	-117
Nav1.6\mtau	msec	0.005
Nav1.6\mVs	mV	11.6

**Table A2: Initial segment parameters**

Parameter Name	Units	Base Value
Capacitance	pF	100
G Leak	uS	0.303
Ca HVA Channel\GMax	uS	0.0061
Ca L-Type Channel\GMax	uS	0.005
CaLeak\GMax	uS	0.00005
K Ca Channel\GMax	uS	1.29
Kf\GMax	uS	0.55
KLeak\GMax	uS	0.079
Na-Ca Pump\IMax	nA	0.054
Na-Ca Pump\KConc\Ca	Mol/m3	0.00008
Nav1.6\GMax	uS	6

**Table A3: Soma parameters**

Parameter Name	Units	Base Value
Capacitance	pF	770
G Leak	uS	0.45
Ca L-Type Channel\GMax	uS	0.002
CaHVA\GMax	uS	0.0005
CaLeak\GMax	uS	0.00019
H\GMax	uS	0.227
K(Ca)\GMax	uS	1
Kf\GMax	uS	0.61
KLeak\GMax	uS	0.4
Na-Ca\IMax	nA	0.096
Na-Ca\KConc\Ca	Mol/m3	0.00262
Nav1.1\GMax	uS	10

**Table A4: Dendrite parameters**

Parameter Name	Units	Base Value
Capacitance	pF	10000
G Leak	uS	0.41
KLeak\GMax	uS	0.033

**Table A5: Inter-compartment parameters**

Parameter Name	Units	Base Value
Initial segment- Soma Interface\G	uS	4
Soma – Dendrite Interface\G	uS	1.12

# **Appendix B**

## **Model Behaviors and Base Values**

**Table B1: Model behaviors**

<b>Behavior Name</b>	<b>Protocol</b>	<b>Base Value</b>
FI.Curvature\ (Hz)	Ramp Frequency-Current	0.05007
FI.First Spike Level\ (mV)	Ramp Frequency-Current	4.79569
FI.Fo\ (Hz)	Ramp Frequency-Current	10.1498
FI.Io\ (Hz)	Ramp Frequency-Current	11.2765
FI.Last Spike Level\ (mV)	Ramp Frequency-Current	14.5288
FI.Max Spike Level\ (mV)	Ramp Frequency-Current	14.7887
FI.SPGain\ (Hz/nA)	Ramp Frequency-Current	0.72251
IV.Fast Onset\ (mV)	Current-Voltage	-53.9319
IV.G Min\ ( $\mu$ s)	Current-Voltage	0.69497
IV.GFast Min\ ( $\mu$ s)	Current-Voltage	-1.66499
IV.Gn\ ( $\mu$ s)	Current-Voltage	1.22554
Rheobase\ (nA)	Rheobase	9.348
Rheobase.Spike Time\ (msec)	Rheobase	49.9144
Rheobase.SpikeLevel\ (mV)	Rheobase	7.13904
Rheobase.SpikeThreshold\ (mV)	Rheobase	-52.958
Sag.Gn\ ( $\mu$ s)	Sag	1.09962
Sag.HalfDecay\ (msec)	Sag	106.998
Sag.SR	Sag	0.15633
Sag.TTP\ (msec)	Sag	39.1107
Spike.ADPWidth\ (msec)	Spike	4.55897
Spike.AHPDur\ (msec)	Spike	133.634
Spike.AHPHalfDur\ (msec)	Spike	23.642
Spike.AHPMag\ (mV)	Spike	11.1423
Spike.AHPTTP\ (msec)	Spike	23.0548
Spike.fAHPLLevel\ (mV)	Spike	-63.1217
Spike.Height\ (mV)	Spike	86.5771
Spike.LateADPMag	Spike	0.1795
Spike.LateADPTTP\ (mV)	Spike	208.535
Spike.Width\ (msec)	Spike	1.8404
SpikeRamp.CurrentSlope\ (nA/s)	Spike Ramp	12
SpikeThresh.ADPWidth\ (msec)	Spike Threshold	4.70393
SpikeThresh.AHPDur\ (msec)	Spike Threshold	133.989
SpikeThresh.AHPHalfDur\ (msec)	Spike Threshold	23.9241
SpikeThresh.AHPMag\ (mV)	Spike Threshold	11.1541
SpikeThresh.AHPTTP\ (msec)	Spike Threshold	22.985
SpikeThresh.fAHPLLevel\ (mV)	Spike Threshold	-63.5187
SpikeThresh.Height\ (mV)	Spike Threshold	85.4358
SpikeThresh.I\ (nA)	Spike Threshold	54.6875
SpikeThresh.LateADPMag\ (mV)	Spike Threshold	0.17948
SpikeThresh.LateADPTTP\ (msec)	Spike Threshold	207.836
SpikeThresh.V\ (mV)	Spike Threshold	-50.1842
SpikeThresh.Width\ (msec)	Spike Threshold	1.97584

# References

1. Hodgkin, A.L. and A.F. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve*. J Physiol, 1952. **117**(4): p. 500-44.
2. Baldissera, F., B. Gustafsson, and F. Parmiggiani, *Adaptation in a simple neurone model compared to that of spinal motoneurons*. Brain Research, 1973. **52**: p. 382-4.
3. Booth, V. and J. Rinzel, *A minimal, compartmental model for a dendritic origin of bistability of motoneuron firing patterns*. Journal of Computational Neuroscience, 1995. **2**(4): p. 299-312.
4. Bower, J.M. and C. Koch, *Experimentalists and modelers: can we all just get along?* Trends Neurosci, 1992. **15**(11): p. 458-61.
5. Foster, W.R., L.H. Ungar, and J.S. Schwaber, *Significance of conductances in Hodgkin-Huxley models*. Journal of Neurophysiology, 1993. **70**(6): p. 2502-18.
6. Golowasch, J., et al., *Failure of averaging in the construction of a conductance-based neuron model*. J Neurophysiol, 2002. **87**(2): p. 1129-31.
7. Gruber, A.J., et al., *Modulation of striatal single units by expected reward: a spiny neuron model displaying dopamine-induced bistability*. J Neurophysiol, 2003. **90**(2): p. 1095-114.
8. Schild, J.H., et al., *Afferent synaptic drive of rat medial nucleus tractus solitarius neurons: dynamic simulation of graded vesicular mobilization, release, and non-NMDA receptor kinetics*. J Neurophysiol, 1995. **74**(4): p. 1529-48.
9. Bhalla, U.S. and J.M. Bower, *Exploring parameter space in detailed single neuron models: simulations of the mitral and granule cells of the olfactory bulb*. J Neurophysiol, 1993. **69**(6): p. 1948-65.

10. Vanier, M.C. and J.M. Bower, *A comparative survey of automated parameter-search methods for compartmental neural models*. J Comput Neurosci, 1999. **7**(2): p. 149-71.
11. Ali-Hassan, W.A., G.M. Saidel, and D. Durand, *Estimation of electrotonic parameters of neurons using an inverse Fourier transform technique*. IEEE Trans Biomed Eng, 1992. **39**(5): p. 493-501.
12. LeMasson, G. and R. Maex, *Introduction to Equation Solving and Parameter Fitting*, in *Computational Neuroscience*, E. De Schutter, Editor. 2001, CRC Press: Boca Raton. p. 1-22.
13. Sultan, F. and J.M. Bower, *Quantitative Golgi study of the rat cerebellar molecular layer interneurons using principal component analysis*. J Comp Neurol, 1998. **393**(3): p. 353-73.
14. Fenelon, G., et al., *Central complex of the primate thalamus: a quantitative analysis of neuronal morphology*. J Comp Neurol, 1994. **342**(3): p. 463-79.
15. Pearson, J.C., J.R. Norris, and C.H. Phelps, *Subclassification of neurons in the subthalamic nucleus of the lesser bushbaby (*Galago senegalensis*): a quantitative Golgi study using principal components analysis*. J Comp Neurol, 1985. **238**(3): p. 323-39.
16. Yelnik, J., et al., *Golgi study of the primate substantia nigra. I. Quantitative morphology and typology of nigral neurons*. J Comp Neurol, 1987. **265**(4): p. 455-72.
17. Lopez-Rubio, E., et al., *Principal components analysis competitive learning*. Neural Comput, 2004. **16**(11): p. 2459-81.
18. Matsumoto, I., et al., *DNA microarray cluster analysis reveals tissue similarity and potential neuron-specific genes expressed in cranial sensory ganglia*. J Neurosci Res, 2003. **74**(6): p. 818-28.

19. Plescia, C., C. Rogler, and L. Rogler, *Genomic expression analysis implicates Wnt signaling pathway and extracellular matrix alterations in hepatic specification and differentiation of murine hepatic stem cells*. *Differentiation*, 2001. **68**(4-5): p. 254-69.
20. Alekseenko, S.V., et al., *Structure of internal interneuronal connections in field 17 of the cat cerebral cortex*. *Neurosci Behav Physiol*, 2004. **34**(6): p. 617-20.
21. Simon, O., et al., *Automatized clustering and functional geometry of human parietofrontal networks for language, space, and number*. *Neuroimage*, 2004. **23**(3): p. 1192-202.
22. Hines, M., *A program for simulation of nerve equations with branching geometries*, in *Int J Biomed Comput*. 1989. p. 55-68.
23. Powell, M., *An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives*. *Computer Journal*, 1964. **7**(2): p. 155-162.
24. Kernell, D., *The repetitive impulse discharge of a simple neurone model compared to that of spinal motoneurons*. *Brain Res*, 1968. **11**(3): p. 685-7.
25. Rall, W., et al., *Dendritic location of synapses and possible mechanisms for the monosynaptic EPSP in motoneurons*. *Journal of Neurophysiology*, 1967. **30**(5): p. 1169-93.
26. Lin, P.X., R.D. Fields, and D. v Agoston, *Effects of electrical stimulation on GAP-43 expression in mouse sensory neurons*. *Brain Res Dev Brain Res*, 1993. **76**(1): p. 95-103.
27. Carp, J.S., R.K. Powers, and W.Z. Rymer, *Alterations in motoneuron properties induced by acute dorsal spinal hemisection in the decerebrate cat*. p. 539-48.
28. Graas, E.L., *Exploration of alternatives to general-purpose computers in neural simulation / by Estelle Laure Graas.*, in *School of Electrical and Computer Engineering*. 2003, Georgia Institute of Technology: Atlanta. p. 55.

29. Bitner, M. and G. Skelton, *Low cost, highly effective parallel computing achieved through a Beowulf cluster*. p. 42-7.
30. Squyres, J.M. and A. Lumsdaine, *A Component Architecture for LAM/MPI*, in *Proceedings, 10th European PVM/MPI Users' Group Meeting*. 2003, Springer-Verlag: Venice, Italy. p. 379-387 %L squyres03:\_compon\_archit\_lam\_mpi.
31. Burns, G., R. Daoud, and J. Vaigl, *LAM: An Open Cluster Environment for MPI*, in *Proceedings of Supercomputing Symposium*. 1994. p. 379-386 %L burns94:\_lam.