2006-05-05

Nonlinear Optimization of a Stochastic Function in a Cell Migration Model

Dorothy M. Branco
Worcester Polytechnic Institute

Nonlinear Optimization of a Stochastic Function in a Cell Migration Model

by

Dorothy Marie Branco

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Applied Mathematics

by

May 2006

APPROVED:

Professor Roger Lui, Thesis Advisor

Professor Bogdan Vernescu, Department Head

Abstract

The basis for many biological processes such as cell division and differentiation, immune responses, and tumor metastasis depends upon the cell's ability to migrate effectively. A mathematical model for simulating cell migration can be useful in identifying the underlying contributing factors to the crawling motions observed in different types of cells. We present a cell migration model that simulates the 2D motion of amoeba, fibroblasts, keratocytes, and neurons according to a set of input parameters. In the absence of external stimuli the pattern of cell migration follows a persistent random walk which necessitates for several stochastic components in the mathematical model. Consequently, the cell metrics which provide a quantitative description of the cell motion varies between simulations. First we examine different methods for computing the error observed between the output metrics generated by our model and a set of target cell metrics. We also investigate ways of minimizing the variability of the output by varying the number of iterations within a simulation. Finally we apply finite differences, Hooke and Jeeves, and Nelder-Mead minimization methods to our nonlinear stochastic function to search for optimal input values.

Acknowledgments

I would like to express special thanks to my advisor, Professor Roger Lui, whose guidance, suggestions, and encouragement have made this thesis possible. I am very grateful for his endless time and patience, and for giving me the opportunity to work on this project.

Second, I would like to thank Professor Yu-Li Wang from the University of Massachusetts Medical School for allowing us to study his cell migration program and for his close assistance throughout the project.

In addition, I have greatly appreciated the prompt assistance from Joshua Brandt, Allan Johannesen, and Michael Malone on all the computing aspects of this project including knowledge of the Unix systems and parallel processing at WPI.

I am also very thankful for all the cordial and knowledgeable faculty and staff in the Department of Mathematics whose instruction and support have contributed to the value of my graduate education. I also want to thank my fellow classmates in the mathematics graduate program who have been more than helpful and supportive during my time here.

Finally, I want to thank my parents and friends for their unending love and support. I cannot end without thanking my dear friend, Michael Sylvia, who has assisted me with his tremendous knowledge in medicine and has provided me with everlasting support and inspiration throughout this process.

Contents

1 Introduction **1**

2 Cell Migration Model **4**

2.1 Computational Model of Cell Migration 4

2.1.1 *SimMigration* Input Parameters 4

2.1.2 Ten Steps Governing Cell Migration 5

2.2 Cell Metrics 13

2.3 Four Cell Types Generated by *SimMigration* 15

2.3.1 Amoeba 16

2.3.2 Fibroblast 17

2.3.3 Keratocyte 18

2.3.4 Neuron 19

3 Evaluating Error in *SimMigration* **21**

3.1 Computing Error in *SimMigration* 22

3.2 Assessing Fluctuations in Repeated Simulations of *SimMigration* . . . 25

3.3 Choosing Best Subset of Cell Metrics to Compute *L2Error* 28

3.3.1 Correlation Matrices 35

4 Minimizing a Nonlinear Stochastic Function **38**

4.1 Minimization with Derivatives 39
4.1.1 Finite Difference Method 39
4.2 Minimization without Derivatives 41
4.2.1 Hooke and Jeeves Method 41
4.2.2 Nelder-Mead Method 45

A Nelder-Mead Implementation in C **49**
A.1 main.c code 49

List of Figures

2.1 Flow chart for one iteration of the *nCycle* loop in *SimMigration*. . . . 12

2.2 *SimMigration* Images of an Amoeba During Migration 16

2.3 *SimMigration* Images of a Fibroblast During Migration 17

2.4 *SimMigration* Images of a Keratocyte During Migration 18

2.5 *SimMigration* Images of a Neuron During Migration 19

3.1 Best Subsets of Cell Metrics 36

3.2 Second Best Subsets of Cell Metrics 36

3.3 Third Best Subsets of Cell Metrics 37

List of Tables

2.1 *SimMigration* Input Parameters 5

2.2 *SimMigration* Input Values for the Four Cell Types 15

2.3 *SimMigration* Cell Metrics 20

3.1 Variance of *L2Error* With and Without *nTimes* 26

3.2 Variance of *L2Error* for Different *nTimes*, *nRound*, and *nCycle* Values 28

3.3 Metric Identification Numbers 29

3.4 Maximum Eigenvalues of Subsets of Size 2, 3, 4, and 5 for Amoeba Simulations 31

3.5 Maximum Eigenvalues of Subsets of Size 2, 3, 4, and 5 for Keratocyte Simulations 33

3.6 Maximum Eigenvalues of Subsets of Size 2, 3, 4, and 5 for Fibroblast Simulations 34

3.7 Maximum Eigenvalues of Subsets of Size 2, 3, 4, and 5 for Neuron Simulations 35

4.1 Amoeba Target Value Input Parameters and Step Sizes for Finite Difference Methods 40

4.2 *L2Error* Function Derivative Approximations using Finite Difference Methods for Amoeba 41

4.3 Results of Hooke and Jeeves Minimization Method on Amoeba Input Parameters 44
4.4 Results from Nelder-Mead Minimization on each Amoeba Input Parameter 48
4.5 Results from Nelder-Mead Minimization on 4 Amoeba Input Parameters 48
4.6 Results from Nelder-Mead Minimization on 5 Amoeba Input Parameters 48
4.7 Results from Nelder-Mead Minimization on 5 Amoeba Input Parameters 48

Chapter 1

Introduction

Cell migration has been implicated in a variety of medical conditions, for example: cancer, immune dysfunction, and inflammatory disease such as multiple sclerosis, rheumatoid arthritis, and atherosclerosis [3]. In cancer one of the differences between benign and malignant cells is the property of motility. When a cancerous cell gains the ability to migrate, what is known as cancer metastasis, the severity of the condition increases. On the other hand, in immune diseases such as Wiskott-Aldrich syndrome and HIV infection, normal cell migratory function is impaired[13, 1]. Understanding the complex dynamic of cell motility will help both scientists and physicians develop methods to modulate cell movement by either inhibiting migration, as in cancer, or provoking migration, as in immune dysfunctions [3].

The underlying mechanisms of cell movement are governed by a complex molecular process. Actin, a protein found in the cytoskeleton of eukaryotic cells, plays a major role in cell locomotion [2, pg. 821]. The stages of cell movement are classified into three main steps: protrusion, adhesion, and traction [2, pg. 845]. During protrusion, tightly bound bundles of actin filaments extend away from the cell in the direction of its motion. The site of protrusion constitutes what is known as the

leading edge forming extensions which may be classified as filopodia or pseudopodia. In the adhesion stage, actin attaches to certain sites on the surface beneath the cell. In fibroblast cells, these attachments are made with stress fibers, a particular bundle of actin filaments containing myosin-II, and form focal adhesions on the cell surface [2, pg. 840]. These stress fibers add a tension component between the cell surface and its surroundings and are continually being assembled and disassembled during cell migration. Traction is the final stage that involves the forward motion of the cell. Cell movement takes place over many repetitions of the three stage process of protrusion, adhesion, and traction.

An attempt to explain the inherent causes of the force invoking cell protrusion has led to multiple theories and may not be standard for all types of cells. Current theories include actin polymerization at the leading edge, internal hydrostatic pressure, and the ability of motor proteins to convert chemical energy into a force [9, 11]. Less is known about traction forces. One theory suggests that certain extensions of the cell's membrane, such as filopodia, generate a layer of actin which allows the cell to move forward [9].

Although all cells follow the basic steps of movement, different types of cells will have variations in qualities, such as shape, speed, and persistence, in their movements. For example, the unicellular organism *Dictyostelium discoideum* forms short rounded protrusions called pseudopods as it crawls across a surface [4]. The fibroblast, however, forms longer thinner protrusions named lamellipodia which extend and retract at greater speeds [2, pg. 827]. These unique characteristics are attributed to each cell's specific function.

Advancements in cell imaging have enabled scientists to measure certain characteristics of cell motion. As a result scientists are currently able to phenotypically identify a specific cell according to a set of quantitative measurements based

on past experiments. Consequently, new knowledge supporting the complex mechanisms behind cell movement is directly obtained (experimentally) by comparing a normal cell to a genetically mutated cell targeting specific proteins. However, identifying precisely at which step in the process of cell motion that changes are occurring is experimentally challenging.

A mathematical model based on internally driven cell mechanisms can be a valuable tool in future cell research. In this thesis we examine a computer simulation of cell migration based on a set of input parameters that govern the processes of cell protrusion, adhesion and traction. After the simulation is repeated a number of times, a set of cell metrics is computed which quantitatively describe the cell's motion. Upon examining the output of this computer simulation, we observed that repeated simulations were highly variable due to the stochastic components incorporated in modeling the random nature of cell motion. We have applied several numerical techniques in order to examine the program's inconsistency and to minimize the fluctuations in the program's output. First we devise a method for computing the error between the program's output cell metrics and a set of target metrics which can be obtained from observing live cells in the laboratory. We refer to this as *L2Error* and compute it using all or a subset of the cell metrics. In addition, we perform a correlation analysis of all possible subsets of the cell metrics to test for independency between the metrics.

Finally, we examine ways of minimizing our nonlinear stochastic *L2Error* function. First we apply finite difference methods by approximating the function's derivatives. Since finding the function's derivatives was unsuccessful, we proceeded with the *Hooke and Jeeves* and *Nelder-Mead* direct search methods which do not rely on derivative approximations.

Chapter 2

Cell Migration Model

In this chapter we describe the computer model for cell migration. The computer program that simulates cell migration was developed by Yu-Li Wang, Ph.D., a professor in the Department of Physiology at the University of Massachusetts Medical School. We will refer to his program as *SimMigration*. The computer program is written in C/C++. There are ten steps in the simulation program of cell migration that are directed by eleven input parameters. After one complete simulation a set of cell metrics are produced that identify a particular cell type. We aim to investigate the behavior of the cell metrics over repeated simulations for different types of cells.

2.1 Computational Model of Cell Migration

2.1.1 *SimMigration* Input Parameters

SimMigration simulates cell movement for four types of cells: amoeba, fibroblast, keratocyte, and neuron. Each cell type has different characteristics and displays a unique set of cell metrics in the laboratory. *SimMigration* uses stimulating signals to induce cell protrusion and inhibiting signals to drive cell retraction. There are

eleven input parameters which are set by the user in *SimMigration*. Each cell type has a different set of values for these input parameters which guides its distinctive features of motility. The following table lists the eleven input parameters followed by a number which indicates the step in the *SimMigration* program where it appears. A brief description is also given for each input parameter.

Table 2.1: *SimMigration* Input Parameters

Parameter, Step	**Description**
fDiffuse, 3	diffusion rate of the stimulating signal
fDecay, 3	fraction of stimulating signal left after each iteration
fBurst, 2	probability of gaining additional stimulating signal
nBurst, 2	amount of stimulating signal added to the boundary points
dlnhibitorConc, 5	amount of inhibiting signal
fProtrusion, 8	rate at which radius increases upon protrusion
fRetraction, 7	rate at which radius decreases upon retraction
fFeedback, 4	probability of signals receiving additional bursts of signal
nOffPoint, 4	threshold of Feedback curve
bFocalAdhesion, 6	ability to form focal adhesions
fFAAssembly, 6	probability of forming a focal adhesion during protrusion
fFAHalflife, 6	amount of focal adhesions lost

2.1.2 Ten Steps Governing Cell Migration

Step 1: Initialization

One iteration of *SimMigration* consists of two nested loops. The outer loop is repeated the number of times given by the variable *nRound*. We have set *nRound* to be 20 for all the simulations unless indicated otherwise. The first step is the initialization step, which begins in this outer loop and initializes each of the cell metrics. The location of the center of the cell is expressed in Cartesian coordinates as (x_c, y_c). The cell has 360 boundary points which are given in polar coordinates (r_j, θ_j) with respect to the cell center established at the origin. The initial radius

from each boundary point to the cell's center is set in *SimMigration* as *fRadius* and equals 10 units in all of our simulations.

Once the boundary points are defined, each is assigned a value for the presence of stimulating signal. At each time point t_i, the stimulating signal for the j^{th} boundary point is expressed as $S_j^+(t_i)$. Stimulating signals are initially assigned using a uniformly distributed random number generator. First, the random number generator assigns to each boundary point a random number R_j between 0 and 1. If $R_j < fBurst$, then $S_j^+(t_0) = nBurst$. Otherwise $S_j^+(t_0) = 0$. A positive value indicates the presence of an initial stimulating signal while a value of zero indicates its absence. Once the cell's initial position, boundary points, and stimulating signals are determined the initialization step is complete.

Step 2: Display Cell

The second step begins inside the inner loop, which is executed a set number of times given by the variable *nCycle*. We have defined *nCycle* to be 1200 for all simulations except where we have noted otherwise. Each iteration in the inner loop is comprised of steps 2 through 10 and produces a new set of boundary points and a new cell center based upon the cell's position in the previous iteration. The cell is displayed by connecting the new 360 boundary points in order of their angles.

Step 3: Diffuse and Decay of Stimulating Signals

The following step calculates the diffusion and decay of existing stimulating signals along the boundary points. The two input parameters *fDiffuse* and *fDecay* give the diffusion rate and the decay constant of the cell. At boundary points where the cell is protruding, the stimulating signal is diffused to nearby locations. Simultaneously all stimulating signals that are present must decay uniformly. Thus stim-

ulating signals are dependent on signals present at previous time points. Modeling stimulating signals in cells is complex and requires both a deterministic and stochastic component. First the deterministic component models the change in stimulating signal based on *fDiffuse* and *fDecay*. In this step, a diffusion function D is calculated as $D(S^+_{j-1}, S^+_j, S^+_{j+1}) = \left[\frac{S^+_{j-1} - S^+_j}{2d_{j-1,j}} + \frac{S^+_{j+1} - S^+_j}{2d_{j+1,j}}\right] fDiffuse$ where $d_{j-1,j}$ is the distance between the $(j-1)^{th}$ and j^{th} boundary points. Based on diffusion and decay of existing stimulating signals, the stimulating signal at the end of this step is given by $S^+_j(t_i) = \left[S^+_j(t_{i-1}) + D(S^+_{j-1}(t_{i-1}), S^+_j(t_{i-1}), S^+_{j+1}(t_{i-1}))\right] fDecay$. The stochastic component of stimulating signals appears in the next step.

Step 4: Generation of New Pulses of Stimulating Signals

Each boundary point has a probability of gaining a set quantity *nBurst* of additional stimulating signals. This probability not only depends on the difference of a boundary point's stimulating and inhibiting signals, but also increases linearly according to the cell's positive feedback behavior. The net signal at each boundary point is given by $x = S^+_j(t_i) - S^-(t_i)$ where S^+_j is the boundary point's stimulating signal and S^- is the total inhibiting signal which will be explained in Step 5. The function $f(x)$ defines the probability of gaining additional stimulating signal according to the positive feedback so that boundary points with a greater net signal have an increased probability. Since $f(x)$ is a cumulative distribution function, then $0 \leq f(x) \leq 1$. The input parameter *nOffPoint* provides the threshold value for positive feedback such that if $x < nOffPoint$, then $f(x) = 0$ indicating that net signals below the threshold have zero probability of gaining additional stimulating signals. If $nOffPoint \leq x < \frac{1}{fFeedback} + nOffPoint$, then $f(x) = fFeedback(x - nOffPoint)$ indicating that the probability of gaining additional signal for these boundary points increases in proportion to *fFeedback*. Finally,

if $x \geq 1/fFeedback + nOffPoint$, then $f(x) = 1$ indicating that boundary points with very large net signals have the maximum probability of gaining additional bursts of signal.

The probability of stimulating signals gaining an additional bursts of signal given by $f(x)$ is then added to the probability of gaining additional stimulating signals set by *fBurst*. Let $\alpha = f(x) + fBurst$, then $\varphi(\alpha)$ is defined to be a stochastic function where the probability of $\varphi(\alpha) = 1$ is α and the probability of $\varphi(\alpha) = 0$ is $(1 - \alpha)$. This stochastic function is implemented in the *SimMigration* program by using a uniformly distributed random number generator that assigns a random number R_j, between 0 and 100 to each boundary point. If $R_j < \alpha$, then *nBurst* is added to the boundary point's stimulating signal. Each boundary point can attain a minimum stimulating signal equal to 0 and a maximum stimulating signal equal to 100.

The fourth step completes the calculation of stimulating signals. The final stimulating signal for the j^{th} boundary point is given by

$S_j^+(t_i) = \left[S_j^+(t_{i-1}) + D(S_{j-1}^+(t_{i-1}), S_j^+(t_{i-1}), S_{j+1}^+(t_{i-1}))\right] fDecay + nBurst\ \varphi(\alpha)$

where the first component was calculated in the third step.

Step 5: Calculation of Inhibiting Signals

Unlike stimulating signals, inhibiting signals are defined globally for the cell. The inhibiting signal at the i^{th} time point is expressed as

$$S^-(t_i) = \left(\sum_{j=1}^{360} S_j^+\right) dInhibitorConc \left(\frac{\pi}{360} \sum_{j=1}^{360} r_j^2\right).$$

The input parameter *dInhibitorConc* is the inhibition constant for the cell. $\sum_{j=1}^{360} S_j^+$ is the sum of the stimulating signals at all the boundary points and $\frac{\pi}{360} \sum_{j=1}^{360} r_j^2$ is the

area of the cell. The inhibiting signal prevents the boundary points from protruding all at once and aids in maintaining the cell's size and direction.

Step 6: Focal Adhesion Turnover

Fibroblasts are the only types of cells in *SimMigration* that form focal adhesions. When simulating fibroblast cells, the input parameter *bFocalAdhesion* is set to TRUE to turn on focal adhesion assembly. The formation and removal of focal adhesions are controlled by two additional input parameters *fFAAssembly* and *fFAHalflife*. Focal adhesion assembly occurs with a probability defined by *fFAAssembly* in protruding areas of the cell. Each protruding boundary point is assigned a number between 0 and 1 using a uniform random number generator. If the value assigned to the protruding boundary point is less than *fFAAssembly*, then a focal adhesion point is formed. Once a focal adhesion point is defined, its location and age are recorded. During cell movement, focal adhesions points determined in previous iterations do not change location. Over time focal adhesions are removed with a probability equal to *fFADisassembly*. Since focal adhesions exhibit exponential decay, their half-life is given by $fFAHalflife = \dfrac{-\ln(2)}{\ln(1 - fFADisassembly)}$. Thus $fFADisassembly = 1 - [\exp(-\ln 2)]^{1/fFAHalflife} = 1 - 0.5^{1/fFAHalflife}$. A focal adhesion is removed if the uniformly distributed random number between 0 and 1 assigned to the focal adhesion is less than *fFADisassembly*. When simulating amoeba, keratocytes, and neurons *bFocalAdhesion* is set to FALSE, and the steps involving focal adhesions are skipped.

Step 7: Retraction

In the absence of focal adhesions retraction occurs if the stimulating signal at each boundary point S_j^+ is less than or equal to the net inhibiting signal S^-. The

radius of the retracting boundary point becomes $r_j(1 - fRetraction)$. However, if the retracted radius is less than the minimum radius allowed which is set to 10 units, then the boundary point is not retracted to ensure that the cell does not collapse at any point.

In the presence of focal adhesions retraction may be restricted at some boundary points. The retracting boundary point does not retract beyond a focal adhesion during retraction. Also, the retracting boundary point does not retract beyond any line connecting two other focal adhesion points that lie on opposite sides of the boundary point. If none of these situations occur, then the boundary point's retracted radius is equal to $r_j(1 - fRetraction)$.

Step 8: Protrusion and New Focal Adhesion Assembly

Protrusion occurs at the j^{th} boundary point if $S_j^+ > S^-$. In the protrusion step, the radius is increased at these boundary points by a normal random variable with mean set by the input parameter *fProtrusion* and variance 0.1**fProtrusion*. For each boundary point that protrudes there is an associated probability of forming a focal adhesion point in fibroblast cells as discussed in step 7. In all types of cells, a site consisting of sequentially protruding boundary points forms a pseudopod. The number of pseudopods are recorded during migration.

Step 9: Reinitialization

Once cell protrusion, adhesion, and retraction have been determined for all boundary points, the next step is to construct the cell with its new position and shape. First location of the new centroid is calculated in Cartesian coordinates. The new cell center after each iteration is given by $x_c(t_{i+1}) = x_c(t_i) + \frac{1}{360}\sum_{j=1}^{360} r_j \cos(\theta_j)$

and $y_c(t_{i+1}) = y_c(t_i) + \frac{1}{360}\sum_{j=1}^{360} r_j \sin(\theta_j)$, where r_j is the new radius calculated for the j^{th} boundary point with angle θ_j at the end of the i^{th} iteration. Then the polar coordinates for the new boundary points are calculated with respect to $(x_c(t_{i+1}), y_c(t_{i+1}))$ as follows. Let $x_j(t_{i+1}) = r_j \cos(\theta_j)$ and $y_j(t_{i+1}) = r_j \sin(\theta_j)$. Then the new radius r_j and angle θ_j are calculated as $r_j = \sqrt{(x_j(t_{i+1}))^2 + (y_j(t_{i+1}))^2}$ and $\theta_j = \arctan(y_j(t_{i+1})/x_j(t_{i+1}))$. This completes one cycle through the *nCycle* loop of *SimMigration*.

Step 10: Morphometry

The final step in *SimMigration* is the calculation of the cell metrics which describe the characteristics of its motion. Details of the metrics and their calculations will be given in the following section. The cell metrics are computed inside the second loop and are averaged over *nCycle* times. This is repeated within the outer loop *nRound* number of times and again the average value for each metric is computed. The final output of *SimMigration* is the average value of each cell metric over *nRound* $\times$ *nCycle* times. Since the computer simulation of cell movement includes some random components, it is best to take the average values of the metrics over repeated simulations. We will investigate this issue further in the next chapter.

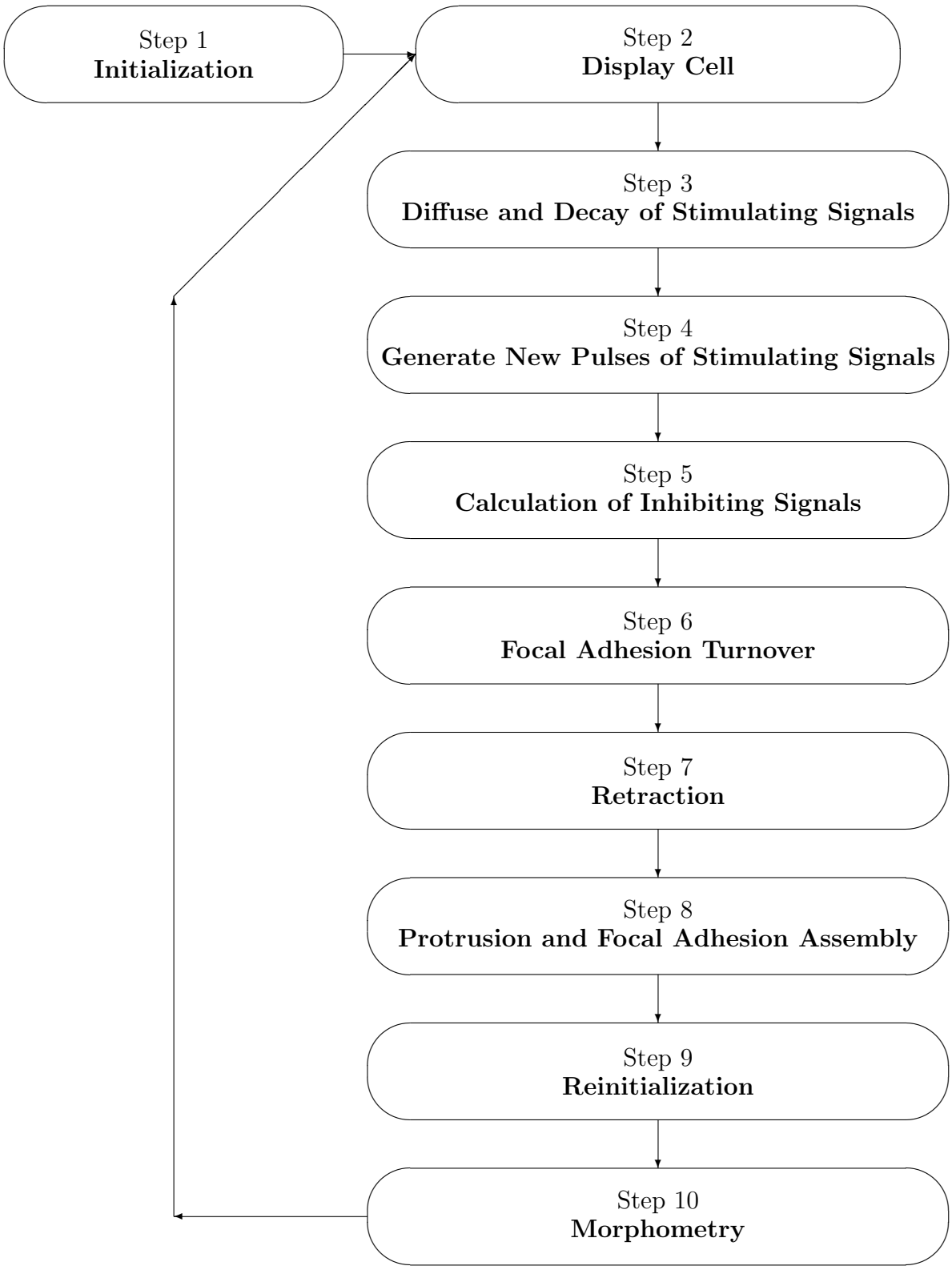

Figure 2.1: Flow chart for one iteration of the *nCycle* loop in *SimMigration*.

2.2 Cell Metrics

Cell metrics offer a means of quantitatively describing the properties of cell motility such as size, shape, and persistence. Current technology is available to measure these quantities in digital movies of live cells. The ultimate goal of the simulation program is to try to closely match experimental metrics of different types of cells. The cell metrics include area, speed, perimeter, maximum radius, persistence, percent extension, number of pseudopods, and roundness.

The following are the formulas we use to compute the cell metrics.

Area

Area is calculated as the average space contained inside the 360 boundary points.

$$Area = \frac{\pi}{360} \sum_{i=1}^{360} r_i^2$$

Speed

Speed is measured by calculating the distance between the cell center at time i, $(x_c(t_i), y_c(t_i))$, and at time $i+1$, $(x_c(t_{i+1}), y_c(t_{i+1}))$ per unit time.

$$Speed = \sqrt{(x_c(t_{i+1}) - x_c(t_i))^2 + (y_c(t_{i+1}) - y_c(t_i))^2}$$

Perimeter

Perimeter is the distance connecting all the boundary points of the cell.

$$Perimeter = \sum_{i=1}^{360} \sqrt{[r_i \cos(\theta_i) - r_{i+1} \cos(\theta_{i+1})]^2 + [r_i \sin(\theta_i) - r_{i+1} \sin(\theta_{i+1})]^2}$$

Maximum Radius

The maximum radius is the largest radius, r_i, obtained over all the boundary points where $i = 1, .., 360$.

$$RadMax = \max_i r_i$$

Persistence

Persistence is a measure of the cell's tendency to travel in the same direction for a sustained period of time. It is calculated based on the coordinates of three successive cell centers c_1, c_2, and c_3 and the difference in the angles θ_1 and θ_2. θ_1 and θ_2 are formed between the positive x-axis and the lines joining the points c_1 and c_2, and c_2 and c_3, respectively. In particular, the lines joining two cell centers form the hypotenuse of a right triangle and the angles are calculated by $\theta = \arctan(\frac{\Delta y}{\Delta x})$, where Δy and Δx are the change in the coordinates of two consecutive cell centers. In the following formula for persistence, $Turnangle = \theta_2 - \theta_1$.

$$Persistence = \left| \frac{Speed}{1 + \frac{100}{360} \times TurnAngle} \right|$$

Percent Extension

Extension is the percentage of the perimeter of the cell that is connected by protruding boundary points.

$$Extension = \frac{ExtendingPerimeter}{TotalPerimeter} \times 100$$

Pseudopods

The number of pseudopods is determined by counting the number of groups of successively protruding boundary points.

Roundness

Roundness is given by the area of the cell divided by the area of a circle with the cell's maximum radius. As a cell attains a rounder or more circular shape, roundness tends toward a value of 1.

$$Roundness = \frac{Area}{\pi Radmax^2}$$

2.3 Four Cell Types Generated by *SimMigration*

The following table shows the values for the input parameters used to generate the four cell types. Amoeba, keratocytes, and neurons require nine input values, while fibroblasts require eleven input values due to its focal adhesions. In this section we will see how the migration of each cell type is affected, despite the small changes among the input values between the different cells.

Table 2.2: *SimMigration* Input Values for the Four Cell Types

Parameter	**Amoeba**	**Fibroblast**	**Keratocyte**	**Neuron**
fDiffuse	0.2	0.2	0.6	0
fDecay	0.999	0.999	0.98	0.9999
fBurst	0.003	0.002	0.002	0.001
nBurst	4	4	5	4
dlnhibitorConc	0.000003	0.000003	0.000003	0.000003
fProtrusion	0.2	0.15	0.08	0.08
fRetraction	0.005	0.003	0.008	0.0001
fFeedback	1	1	10	10
nOffPoint	5	5	1	0
bFocalAdhesion	FALSE	TRUE	FALSE	FALSE
fFAAssembly	N/A	0.02	N/A	N/A
fFAHalflife	N/A	50	N/A	N/A

2.3.1 Amoeba

The amoeba is a single-celled organism that thrives in soil, fresh water, and oceans [5]. During movement, the amoeba takes on many different shapes. It is most distinguished by its projections, called pseudopods, which aid in its motion. In Figure 2.2 the images taken from *SimMigration* show the amoeba's irregular shape and its ability to forms both short and long protrusions. Its leading edge tends to extend a quarter to a third of its circumference. The amoeba moves about rapidly and frequently changes direction.

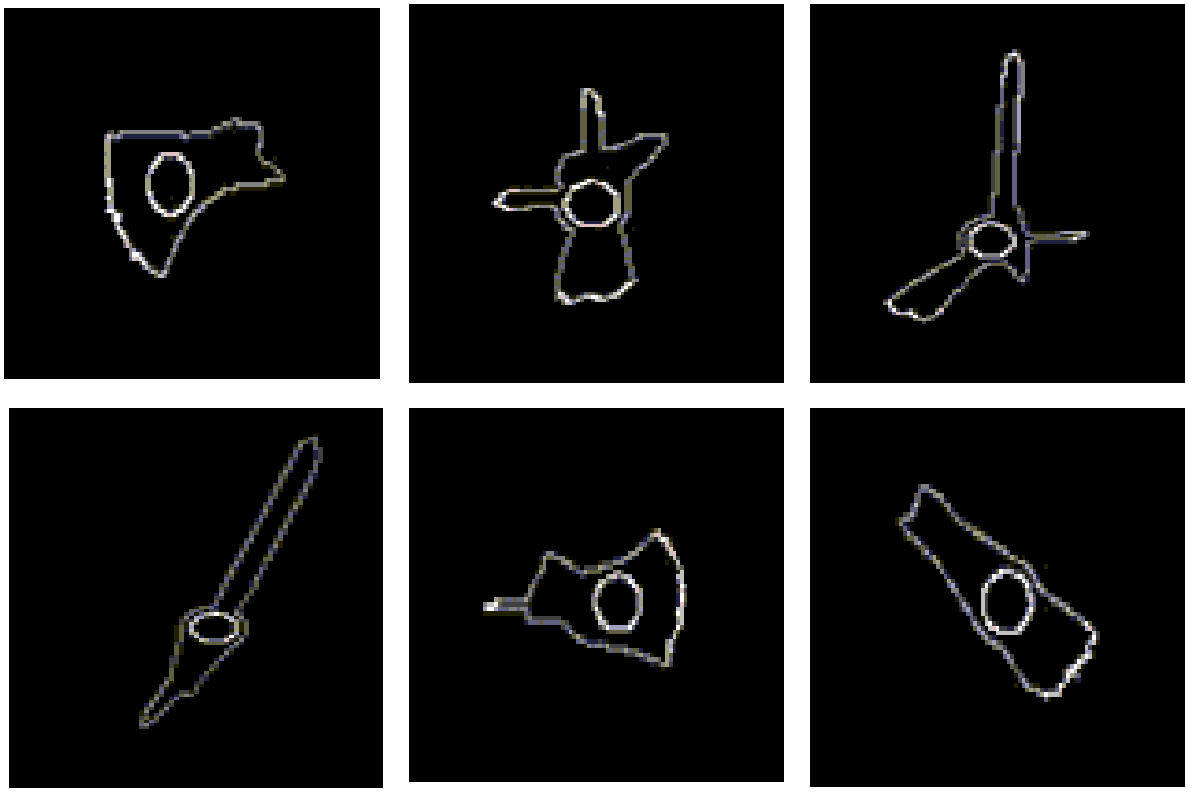

Figure 2.2: *SimMigration* Images of an Amoeba During Migration

2.3.2 Fibroblast

Fibroblasts are found in connective tissue and are important in scar formation and wound healing. These cells have the ability to form focal adhesions which aid in cell migration. These focal adhesions are marked by the white dots in Figure 2.3. Focal adhesion points inhibit retraction and protrusion at particular boundary points. A moving fibroblast forms a broad leading edge covering about a third of the cell's circumference and changes direction infrequently since retraction is limited by focal adhesions.

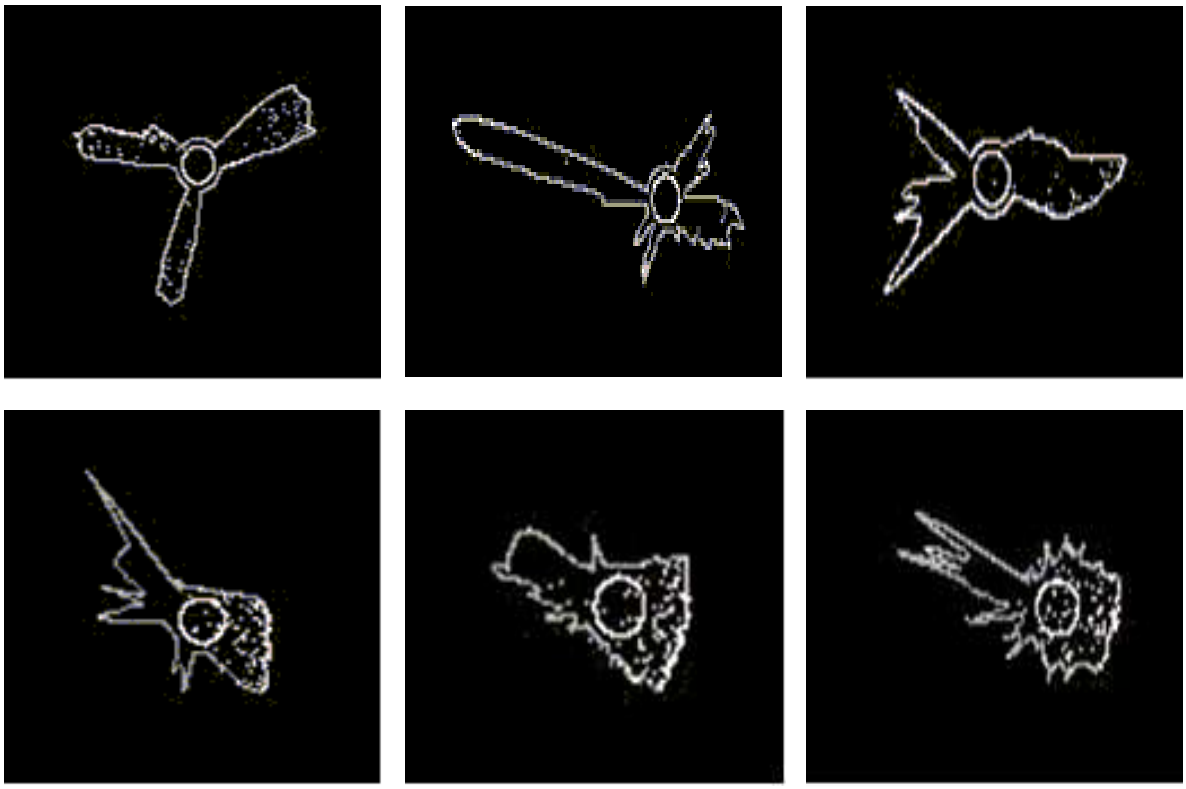

Figure 2.3: *SimMigration* Images of a Fibroblast During Migration

2.3.3 Keratocyte

Keratocytes refer to a certain class of abnormally shaped red blood cells that have two prominent cytoplasm projections. Red blood cells are normally shaped like round disks, but keratocytes shown in Figure 2.4 have a semicircular shape that is induced by projections on opposite sides of the cell. In the *SimMigration* program keratocytes form the broadest leading edge which cover half of its perimeter. Once the leading edge has been established, the keratocyte tends to move in the same direction exhibiting persistency.

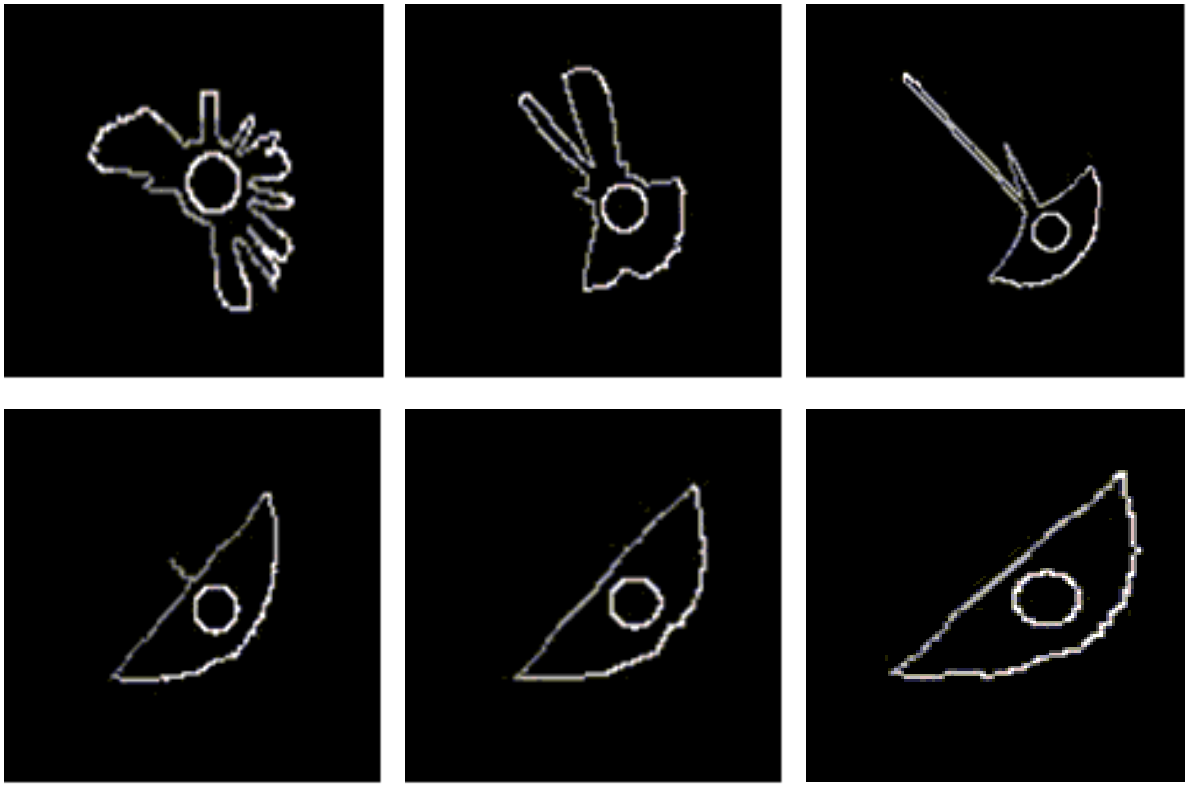

Figure 2.4: *SimMigration* Images of a Keratocyte During Migration

2.3.4 Neuron

Neurons play a fundamental role in quickly transmitting signals to and from the brain. Their primary means of signal transmission is through a very long and this protrusion called an axon. Neurons typically show one leading edge extending to form an axon while the cell body remains virtually immobile. Figure 2.5 shows a neuron during migration whose cell body remains unchanged except for one protrusion that continues to extend further.

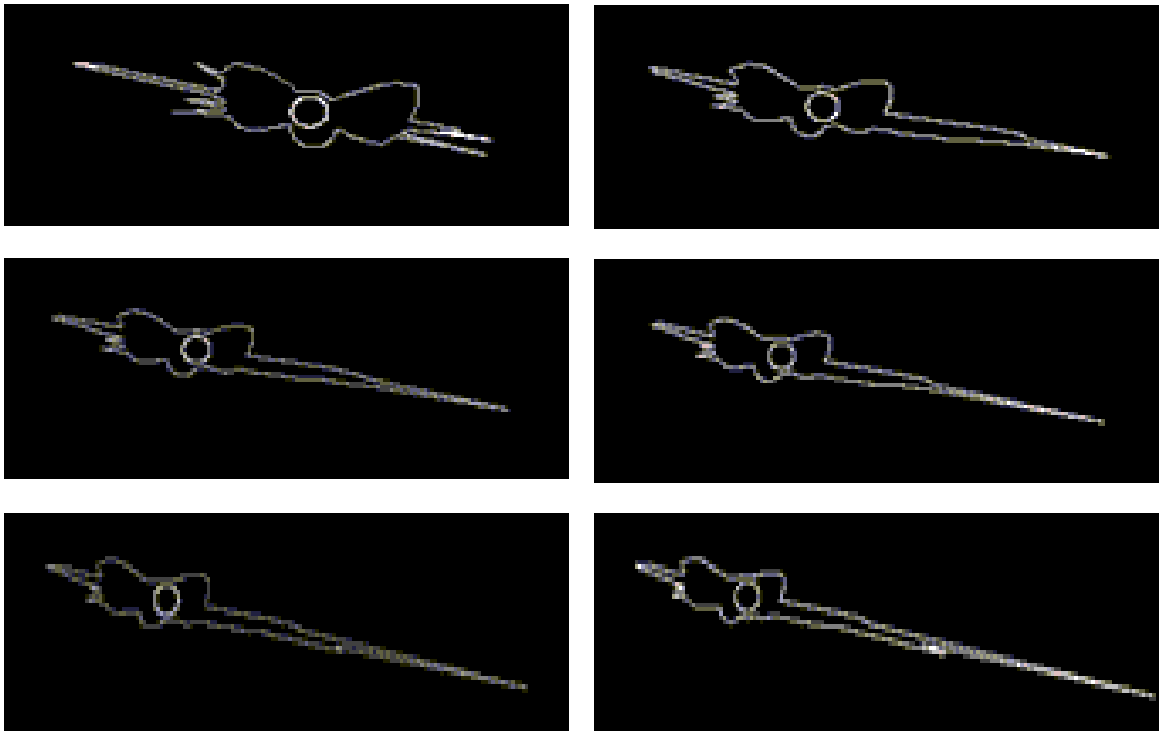

Figure 2.5: *SimMigration* Images of a Neuron During Migration

Table 2.3 shows some typical values of cell metrics produced by the *SimMigration* program for the four types of cells . Amoebas exhibit large areas, fast speeds, and average about two pseudopods. Keratocytes show the smallest area and greatest percent extension. Neurons have the slowest speed which reflects their tendency to remain stationary while extending protrusions. Fibroblasts have the largest area, perimeter, and maximum radius along with a fast speed indicative of their importance in quickly migrating to injured sites.

Table 2.3: *SimMigration* Cell Metrics

Metrics	**Amoeba**	**Fibroblast**	**Keratocyte**	**Neuron**
Area	1144.459464	1243.510226	969.673176	1164.201204
AreaSD	1.429039	2.059442	2.549485	3.422075
Speed	0.165226	0.098488	0.074442	0.016005
SpeedSD	0.000520	0.000373	0.000260	0.000162
Perimeter	224.759095	300.125728	210.172332	211.186254
PerimeterSD	0.426630	0.606931	0.718930	0.631655
RadMax	49.557879	52.623142	37.384989	37.687814
RadMaxSD	0.100619	0.104838	0.129259	0.087860
Persistence	0.152457	0.064733	0.068065	0.014930
PersistenceSD	0.000528	0.000353	0.000248	0.000156
Extension	21.866658	15.402795	40.419852	20.452028
ExtensionSD	0.050698	0.063789	0.126664	0.169116
Pseudopod	2.162125	1.961715	3.105110	2.177340
PseudopodSD	0.006897	0.006131	0.018323	0.018558
Roundness	0.190823	0.204614	0.362587	0.325892
RoundnessSD	0.000670	0.000946	0.001418	0.001051

Chapter 3

Evaluating Error in *SimMigration*

In this chapter, we discuss the different methods used for computing error between the *SimMigration* output metrics and a set of target metrics. Error was computed using both the L^1-norm, denoted by *L1Error*, and the L^2-norm, denoted by *L2Error*. In the following formulas for *L1Error* and *L2Error*, x_i represents the i^{th} output metric value and $\overline{x_i}$ represents the i^{th} target metric value. Since the number of metrics used in computing *L1Error* and *L2Error* may vary, we express the summation from $i = 1$ to $i = n$.

$$L1Error = \sum_{i=1}^{n} \left| \frac{x_i - \overline{x_i}}{\overline{x_i}} \right|$$

$$L2Error = \sqrt{\sum_{i=1}^{n} \left(\frac{x_i - \overline{x_i}}{\overline{x_i}} \right)^2}$$

Since the L^2-norm yielded better results than the L^1-norm in the Nelder-Mead minimization, we proceeded with only *L2Error* computations. The following section of this chapter will discuss the three methods for computing *L2Error* and its behavior over repeated runs of *SimMigration*.

3.1 Computing Error in *SimMigration*

Method 1 ***L2Error with calibration***

In this first method we compute *L2Error* using the metrics Area, Speed, SpeedSD, RadMax, RadMaxSD, Persistence, Pseudopod, PseudopodSD, and Roundness. The cell metrics outputted by the *SimMigration* program are compared to a set of metrics taken from experimental data. When introducing a set of experimental metrics, there is a discrepancy in the units of measurement. In order to overcome this, it is necessary to introduce calibration. Two metrics, Area and Speed, are used to calibrate length and time by the following equations.

$$dell = \sqrt{\frac{ExperimentalArea}{SimMigrationArea}}$$

$$dtau = \frac{SimMigrationSpeed}{ExperimentalSpeed} \times dell$$

Dell is the ratio relating experimental length to computational length. Similarly, *dtau* is the ratio relating experimental time to computational time. *Dell* and *dtau* are used to calibrate metrics with units. For example, in order to compare the computed value of RadMax, whose unit is computational length, to the experimental value of RadMax, whose unit is experimental length, we must multiply computational RadMax by *dell* to convert its units to experimental length. After calibration, the term used in computing *L2Error* is

$$\left(\frac{RadMax \times dell - \overline{RadMax}}{\overline{RadMax}}\right)^2$$

where RadMax is the output metric from *SimMigration* and $\overline{RadMax}$ is the experimental value. Similarly, the following are the terms for Persistence, RadMaxSD,

and SpeedSD in the computation of *L2Error* with units and calibration.

$$\left(\frac{Persistence \times dell - \overline{Persistence}}{\overline{Persistence}}\right)^2$$

$$\left(\frac{RadMaxSD \times dell - \overline{RadMaxSD}}{\overline{RadMaxSD}}\right)^2$$

$$\left(\frac{SpeedSD \times \frac{dell}{dtau} - \overline{SpeedSD}}{\overline{SpeedSD}}\right)^2$$

Pseudopod and Roundness have no units, thus calibration is not necessary. *L2Error* is then computed with all cell metrics except Area and Speed, which are used only for calibration purposes. This method is of limited use since it cannot compare Area and Speed to the experimental metrics. Area and Speed are two fundamental metrics which may be important in classifying a particular cell type. The set of metrics we used to compute *L2Error* in Method 1 consists of the nine metrics: RadMax, Perimeter, Pseudopod, Extension, Persistence, PseudopodSD, ExtensionSD, SpeedSD, and PersistenceSD. To study the variance of *L2Error* computed with Method 1, we ran *SimMigration* 20 times and recorded *L2Error* each time. The mean and standard deviation of *L2Error* was 3.963 and 0.028, respectively. Since these results still show a 1% error in repeated runs, we seek an alternative method that will reduce the standard deviation in repeated simulations.

Method 2 ***L2Error without calibration***

Calibration may introduce some additional error into the final computation of *L2Error*. An alternative way to compute *L2Error* and avoid calibration is to eliminate all units in the metrics. One way to do this is to use the coefficient of variance of each metric instead of the mean. The coefficient of variance of a metric is its standard deviation divided by its mean.

Let $CV_i = \frac{\sigma_i}{x_i}$ for the i^{th} computational metric with units, and let $\overline{CV_i} = \frac{\overline{\sigma_i}}{\overline{x_i}}$ for the i^{th} experimental metric with units, where σ_i and $\overline{\sigma_i}$ are the standard deviations of the i^{th} computational and experimental metrics, respectively. Then for each metric with units, the term used in computing *L2Error* is $\left(\frac{CV_i - \overline{CV_i}}{\overline{CV_i}}\right)^2$. Terms for metrics without units are computed as before, $\left(\frac{x_i - \overline{x_i}}{\overline{x_i}}\right)^2$.

This method is useful when comparing two sets of data with different units of measurement. We are able to avoid calibration and are not restricted from using the metrics Area and Speed used for calibration in *Method 1*. In the computations of *L2Error* in Method 2, we included the 9 metrics: RadMax, Perimeter, Area, Speed, Pseudopod, Extension, Persistence, PseudopodSD, and ExtensionSD. Analyzing *L2Error* over 20 runs resulted in a mean of 2.703 and a standard deviation of 0.0035. Method 2 resulted in a standard deviation eight times smaller than in Method 1. Although Method 2 provides a sufficiently low variance in *L2Error* over repeated runs, the metrics SpeedSD and PersistenceSD cannot be used as additional metrics because they are needed to calculate the coefficients of variance for speed and persistence.

Method 3 ***L2Error without units***

In this method, we eliminate the need for calibration and coefficients of variance in our *L2Error* computations. For the purpose of analyzing the consistency of this program, we have produced a set of target output metrics from the *SimMigration* program. This set of target output metrics is shown in Table 2.3. The *SimMigration* program is run ten times with $nRound = 20$ and $nCycle = 1200$ and then the output data is averaged over the ten simulations and set as the target output data. Using these computer simulated metrics as the target data eliminates the need for calibration since the same units apply to both the target and output data.

In addition, this method allows all of the cell metrics and their respective standard deviations to be considered in the computation of *L2Error*. We will use Method 3 for computing *L2Error* in our Hooke and Jeeves and Nelder-Mead minimization searches.

3.2 Assessing Fluctuations in Repeated Simulations of *SimMigration*

One computer simulation of *SimMigration* performs *nRound* $\times$ *nCycle* iterations. Another input parameter, *nStartAnalysis*, determines which iteration in the *nCycle* loop to start collecting data for morphometry. At iteration steps less than *nStartAnalysis*, the cell metrics calculated are not included in the final cell metrics output of *SimMigration*. Let $nCycle1 = nCycle - nStartAnalysis$, then the cell metrics are recorded over *nRound* $\times$ *nCycle1* iterations. For one simulation of *SimMigration*, the output for each metric is computed as follows, where $\overline{X}[i]$ denotes the i^{th} output metric and $x[i]_{m,n}$ denotes the i^{th} metric computed during the n^{th} *nRound* iteration and m^{th} *nCycle1* iteration.

Let $R = nRound$ and $C1 = nCycle1$.

$$\overline{X}[i] = \frac{1}{R \times C1} \sum_{n=1}^{R} \sum_{m=1}^{C1} x[i]_{m,n}$$

Similarly, the variance of the i^{th} metric is defined as follows:

$$V[i] = \frac{1}{R \times C1} \sum_{n=1}^{R} \sum_{m=1}^{C1} \frac{\left(x[i]_{m,n} - \overline{X}[i]\right)^2}{R \times C1 - 1}$$

Let $\overline{X}[i]_n = \frac{1}{C1}\sum_{m=1}^{C1} x[i]_{m,n}$. Then the variance can also be expressed as:

$$V[i] = \frac{1}{(R \times C1)(R \times C1 - 1)} \left\{ \sum_{n=1}^{R} (C1 - 1) S_n^2 + C1 \sum_{n=1}^{R} \overline{X}[i]_n^2 - R \times C1 \; \overline{X}[i]^2 \right\}$$

where $S_n^2 = \left\{ \sum_{m=1}^{C1} \left(x[i]_{m,n} - \overline{X}[i]_n \right)^2 \right\} / (C1 - 1)$. We then calculate the standard deviation of each metric as $SD[i] = \sqrt{Var[i]}$. Repeated simulations of *SimMigration* produce slightly different output values for each $\overline{X}[i]$ and $SD[i]$. To minimize the variance in repeated simulations, we have added an outer loop which performs *nTimes* number of simulations. We collect the output metrics computed for each simulation and then compute the mean given by $\overline{\mathbf{X}}[i]$, and the standard deviation given by $\overline{SD}[i] = \sqrt{\sum_{k=1}^{nTimes} \left(\overline{X}[i]_k - \overline{\mathbf{X}}[i] \right)^2 / (nTimes - 1)}$. When we compared these new output values, $\overline{\mathbf{X}}[i]$ and $\overline{SD}[i]$, over repeated runs, the standard deviation between simulations was reduced by 75%. These results are given in Table 3.1.

Table 3.1: Variance of *L2Error* With and Without *nTimes*

Simulation	*L2Error* Without *nTimes*	*L2Error* With *nTimes*
1	2.44296	2.44460
2	2.44720	2.44392
3	2.43790	2.44494
4	2.44346	2.44323
5	2.44626	2.44470
6	2.44221	2.44535
7	2.44856	2.44309
8	2.44734	2.44505
9	2.44535	2.44386
10	2.44571	2.44388
Mean	2.44470	2.44426
Standard Deviation	0.00314	0.00078

Although the addition of the *nTimes* loop decreases the variance of the

L2Error function, we must also consider the time and expense of running the simulations multiple times. A single run with $nTimes = 10$, $nRound = 20$, and $nCycle = 1200$ may only take about an hour on a UNIX server; however, in the next chapter we will perform searches that require twenty or more simulations. These searches would take approximately one day on the UNIX server. To reduce the run times we will use parallel computing which has twenty processors and can run twenty runs of *SimMigration* with these settings in less than an hour.

The variability among successive simulations of *SimMigration* may also be attributed to some random components used in modeling cell migration. For example, a random component is introduced when assigning stimulating signals to the cell's boundary points. Since each boundary point has the probability of gaining a stimulating signal, then a random number is assigned to each boundary point with a Gaussian distribution. The Gaussian distribution for this random component has mean $nBurst$ and variance $0.3 \times nBurst$. Therefore, it is unlikely that a simulation of *SimMigration* will assign the same stimulating signal to each boundary point as in another simulation.

In analyzing the variance of repeated simulations, we expect less variation in *L2Error* as the values of *nTimes*, *nRound*, and *nCycle* increase. We ran *SimMigration* for $nTimes = 10$, 20, and 30, $nRound = 10$ and 20, and $nCycle = 1200$ and 2400. We found that repeated simulations with $nTimes = 10$, $nRound = 20$, and $nCycle = 1200$ give a reasonably low standard deviation and coefficient of variance without too great an increase in the number of iterations. The results are reported in Table 3.2.

Table 3.2: Variance of *L2Error* for Different *nTimes*, *nRound*, and *nCycle* Values

nTimes	nRound	nCycle	*L2Error* Mean	*L2Error* SD	*L2Error* CV
10	10	1200	2.6968034	0.00557	0.002064
10	20	1200	2.7025071	0.00263	0.000973
10	10	2400	2.697345	0.00455	0.001688
10	20	2400	2.7031501	0.00199	0.000737
20	10	1200	2.6961705	0.00583	0.002161
20	20	1200	2.7042991	0.00410	0.001516
20	10	2400	2.6964116	0.00322	0.001195
20	20	2400	2.7031450	0.00354	0.001339
30	10	1200	2.6965091	0.00603	0.002236
30	20	1200	2.7034867	0.00328	0.001214
30	10	2400	2.6969542	0.00317	0.001177
30	20	2400	2.7029223	0.00243	0.000900

3.3 Choosing Best Subset of Cell Metrics to Compute *L2Error*

Whether *L2Error* is computed using method 1, 2, or 3, variability is inevitable. Nevertheless, we question whether a certain subset of the cell metrics produces greater consistency in *L2Error*. It is evident that since the cell metrics can only measure size, shape, and persistence of the cell's movement, some metrics may behave similarly, therefore, any two metrics have the possibility of being highly correlated. This adds redundancy to the model and can contribute to its inconsistency. In this section we search for a particular subset of metrics which may decrease the variability in *L2Error*. Finding the best subsets requires both the computation of correlation matrices and finding the eigenvalues of these matrices.

To investigate the interdependency of the eleven metrics, we proceeded with a correlation analysis. First 10,000 simulations of *SimMigration* were performed and the 11 metrics were recorded. This resulted in a $10,000 \times 11$ matrix which we denote by A. For simplification purposes we have numbered the metrics from 1 through

11 corresponding to the column in matrix A in which the metric appears. We will refer to each metric by its assigned number as indicated in Table 3.3. For example, we will refer to Speed as metric 6 and its values are collected in column 6 of matrix A. Next we choose a subset of size $k \in \{2, 3, ..., 11\}$. There are $\sum_{i=2}^{11} \binom{11}{i} = 2,036$ possible choices where $\binom{11}{i} = \frac{11!}{i!(11-i)!}$. Then the 2,036 subsets are grouped by size which refers to the number of metrics in the subset. For example, the group of size 2 will consist of 55 subsets, the group of size 3 will consist of 165 subsets, and so on.

Table 3.3: Metric Identification Numbers

ID Number	Metric
1	RadMax
2	Perimeter
3	Pseudopod
4	Extension
5	Area
6	Speed
7	Persistence
8	PseudopodSD
9	ExtensionSD
10	SpeedSD
11	PersistenceSD

For each subset in a group we obtain the corresponding simulations matrix. For example, we begin with the group of size 2 and consider the 55 subsets. For each subset containing the metrics i and j, we choose the i^{th} and j^{th} columns and form a $10,000 \times 2$ submatrix A' with these columns. The MATLAB function *corrcoef(A')* is called to compute the correlation matrix of the metrics i and j which we denote by R. Section 3.3.1 describes correlation matrices in more detail. The same procedure is applied to the remaining groups of sizes 3 through 11. This completes the correlation component of obtaining the best subsets.

The next step is to find the maximum eigenvalue of each correlation matrix. To do this we use MATLAB's *eig(R)* function. In some cases negative eigenvalues were obtained from *eig(R)*, therefore, we took the absolute value of each eigenvalue. We then chose the maximum eigenvalue and saved it with its corresponding subset. Now each of our subsets is associated with exactly one eigenvalue. This allowed us to sort the subsets by their eigenvalues from least to greatest. Minimum eigenvalues within a group were closest to 1 in their value. For example, Table 3.4 lists the smallest five eigenvalues obtained for groups of size 2, 3, 4, and 5 from 10,000 simulations of the amoeba cell type.

From these results we see that the subset of size 2 with the smallest eigenvalue of 1.0061 consists of metrics 7 and 10. This is the subset of size 2 with an eigenvalue closest to 1. We choose 1 as an ideal eigenvalue because it is the eigenvalue corresponding to the identity matrix which contains linearly independent columns. Therefore the subset containing metrics 7 and 10 are highly uncorrelated. We consider the subset {7, 10} to be the best subset of size two. Since the second least eigenvalue in the group of size 2 is very close to the minimum eigenvalue, we also consider this as a good subset. Thus the metrics 6 and 10 which produce an eigenvalue of 1.0063 are also highly uncorrelated metrics. We refer to this subset as the second best subset. The third smallest eigenvalue in this group is 1.0452 for metrics 1 and 8 which is still reasonably close to 1. We consider this the third best subset of size 2. We also look at the smallest five eigenvalues found for the groups of size 3, 4, and 5. Notice that as the size of the subsets gets larger, the minimum eigenvalues also increase. For groups larger than size 5, the eigenvalues are much larger than 1 which would show that some of the metrics within the subset are correlated. For our purposes we are interested in finding subsets of metrics that are independent. The larger the size of the subset with a small enough eigenvalue, the more metrics

Table 3.4: Maximum Eigenvalues of Subsets of Size 2, 3, 4, and 5 for Amoeba Simulations

Size	Max Eigenvalue	Subset of Metrics
2	1.0061	7, 10
	1.0063	6, 10
	1.0452	1, 8
	1.0512	4, 5
	1.0797	6, 11
3	1.2291	3, 4, 11
	1.2512	1, 7, 11
	1.2565	3, 4, 10
	1.2761	1, 6, 11
	1.2762	1, 7, 10
4	1.4228	1, 7, 8, 11
	1.4329	1, 6, 8, 11
	1.4439	4, 5, 7, 11
	1.4673	1, 7, 8, 10
	1.4706	4, 5, 6, 11
5	1.6961	4, 5, 7, 8, 11
	1.7018	1, 4, 7, 8, 11
	1.7125	4, 5, 6, 8, 11
	1.7332	1, 4, 6, 8, 11
	1.7349	4, 5, 7, 8, 10

we can show to be independent.

We performed the same analysis for 10,000 simulations of *SimMigration* also for keratocytes, fibroblasts, and neurons. The smallest five eigenvalues obtained for groups of size 2, 3, 4, and 5 for these cells are shown in Tables 3.5 for keratocytes, 3.6 for fibroblasts, and 3.7 for neurons. Figure 3.1 shows the best subsets obtained for each cell type by taking the subset in each group with the smallest eigenvalue. For example, the best subsets for amoeba in order of increasing size are {7, 10}, {3, 4, 11}, {1, 7, 8, 11}, and {4, 5, 7, 8, 11}. We do the same to obtain the second best subsets by taking the subsets with the second smallest eigenvalue in each group. For example, the second best subsets for amoeba are {6, 10}, {1, 7, 11}, {1, 6, 8, 11}, and {1, 4, 7, 8, 11}. The second best subsets for all the cell types are shown in

Figure 3.2. Similarly, the third best subsets are obtained for each cell type and are shown in Figure 3.3.

We now try to identify metrics that are uncorrelated to include in our *L2Error*. The cell metrics may be classified into three categories: *size*, *speed*, and *shape*. The *size* group consists of the metrics RadMax (1), Perimeter (2), and Area (5). The *speed* group contains the metrics Speed (6) and Persistence (7), and the *shape* group consists of Pseudopod (3) and Extension (4).

First we look at the *size* group. The metrics RadMax (1), Perimeter (2), and Area (5) are not collectively contained in any of the best, second best, or third best subsets except in the case of neuron which contains Perimeter (2) and Area (5). However, these subsets have eigenvalues approximately equal to 2 (see Table 3.7, size 5) and therefore, are correlated. We can conclude that the metrics RadMax (1), Perimeter (2), and Area (5) are highly correlated. Since these are all measures of size, we expect these metrics to be correlated.

Next we look at the *speed* group. In the best (Fig. 3.1), second best (Fig. 3.2), and third best (Fig. 3.3) subsets, the metrics Speed (6) and Persistence (7) never appear in the same subset. This indicates that these two metrics are highly correlated which we expect to be the case since persistence is calculated based on speed.

Finally we look at the *shape* group. Pseudopod (3) and Extension (4) appear in the best subset (Fig. 3.1) of size 3 for amoeba and in the second best subset (Fig. 3.2) of size 4 for neuron. In the case of amoeba, the eigenvalue for the subset {3, 4, 11} is 1.2291 which is moderately close to 1. In the case of neuron, the eigenvalue for the subset {3, 4, 5, 8} is 1.6190 showing less independence among of the two metrics. If we look at the third best subsets (Fig. 3.3), there are three subsets containing Pseudopod (3) and Extension (4). One subset is {3, 4, 10} in the case of

Table 3.5: Maximum Eigenvalues of Subsets of Size 2, 3, 4, and 5 for Keratocyte Simulations

Size	Max Eigenvalue	Subset of Metrics
2	1.0026	2, 6
	1.0032	1, 9
	1.0134	1, 11
	1.0194	6, 10
	1.0305	1, 7
3	1.0916	1, 7, 10
	1.1361	5, 6, 9
	1.1408	2, 6, 9
	1.1528	5, 6, 11
	1.1606	3, 6, 9
4	1.1723	5, 6, 9, 11
	1.2053	1, 7, 9, 11
	1.2060	2, 6, 9, 11
	1.2167	1, 6, 9, 11
	1.2258	3, 6, 9, 11
5	1.7127	1, 6, 8, 9, 11
	1.7290	1, 6, 8, 9, 10
	1.8016	1, 7, 8, 9, 11
	1.8055	1, 7, 8, 9, 10
	1.8102	1, 4, 7, 9, 11

amoeba with an eigenvalue of 1.2565 (see Table 3.4, size 3). Another subset is {3, 4, 5} in the case of neuron with an eigenvalue of 1.1344 (see Table 3.7, size 3). The third subset is {3, 4, 8, 9, 11} in the case of fibroblast with an eigenvalue of 1.5683 (see Table 3.6, size 5). While the eigenvalues in the cases of amoeba and neuron are moderately close to 1, the eigenvalue in the case of fibroblast is rather large. It is not clear whether the metrics Pseudopod (3) and Extension (4) are correlated. Although both these metrics result from cell protrusion they may not necessarily be correlated since some cells may form one long and thin protrusion and other cells may form many short protrusions.

The results from the best subsets analysis suggest that only three or four metrics may be necessary to form *L2Error*. The correlation analysis shows that

Table 3.6: Maximum Eigenvalues of Subsets of Size 2, 3, 4, and 5 for Fibroblast Simulations

Size	Max Eigenvalue	Subset of Metrics
2	1.0045	7, 9
	1.0073	8, 11
	1.0152	6, 10
	1.0239	2, 6
	1.0288	1, 9
3	1.0600	1, 4, 8
	1.0769	1, 4, 9
	1.0855	1, 6, 9
	1.0992	5, 6, 9
	1.1229	4, 8, 10
4	1.2807	1, 4, 8, 10
	1.2986	1, 6, 8, 10
	1.3122	2, 4, 8, 9
	1.3131	1, 4, 8, 9
	1.3155	4, 5, 8, 9
5	1.5350	1, 6, 8, 9, 11
	1.5668	1, 4, 6, 9, 10
	1.5683	3, 4, 8, 9, 11
	1.5716	1, 4, 8, 9, 11
	1.5878	3, 4, 6, 9, 10

subsets containing one metric from each group are more likely to be independent and still retain the three main properties of cell motility. Including more than three or four metrics in our calculation of *L2Error* may be unnecessary and contribute to its variability.

Table 3.7: Maximum Eigenvalues of Subsets of Size 2, 3, 4, and 5 for Neuron Simulations

Size	Max Eigenvalue	Subset of Metrics
2	1.0383	5, 7
	1.0399	3, 7
	1.0426	3, 5
	1.0660	5, 6
	1.0667	5, 8
3	1.0428	3, 5, 7
	1.1006	3, 5, 6
	1.1344	3, 4, 5
	1.1731	3, 5, 9
	1.2316	3, 5, 11
4	1.5717	3, 5, 7, 8
	1.6190	3, 4, 5, 8
	1.6212	3, 5, 8, 11
	1.6237	3, 5, 6, 8
	1.6466	3, 5, 8, 10
5	1.9033	2, 3, 5, 7, 8
	1.9251	2, 3, 5, 6, 8
	1.9255	2, 3, 5, 8, 9
	1.9490	2, 3, 4, 5, 8
	1.9994	2, 3, 5, 8, 11

3.3.1 Correlation Matrices

The correlation matrix of n metrics is an $n \times n$ matrix whose i, j entry is the correlation coefficient of metrics i and j denoted by $\rho_{i,j}$. The correlation matrix is symmetrical since $\rho_{i,j} = \rho_{j,i}$. The correlation coefficient of metrics i and j is defined as $\rho_{i,j} = \dfrac{COV(i,j)}{\sigma_i \sigma_j}$, where $COV(i,j)$ is the covariance of metrics i and j. σ_i is the standard deviation of metric i and σ_j is the standard deviation of metric j. Covariance is defined as $COV(i,j) = \dfrac{1}{n-1} \sum_{k=1}^{n} (i_k - \bar{i})(j_k - \bar{j})$, where i_k is the k^{th} observation of metric i and $\bar{i}$ is the mean value of i over all observations.

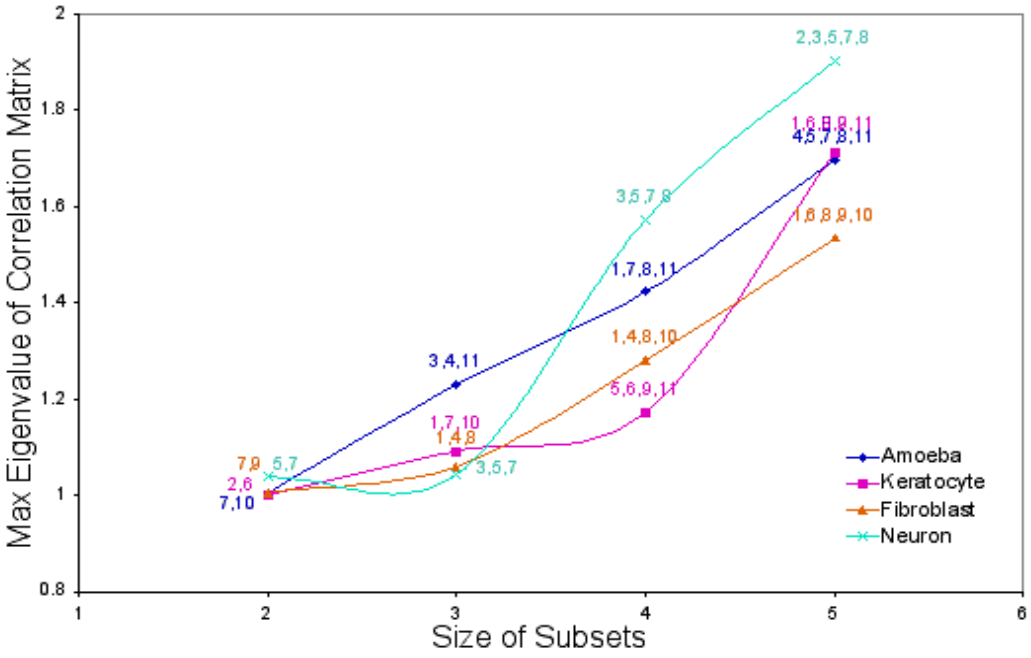

Figure 3.1: Best Subsets of Cell Metrics

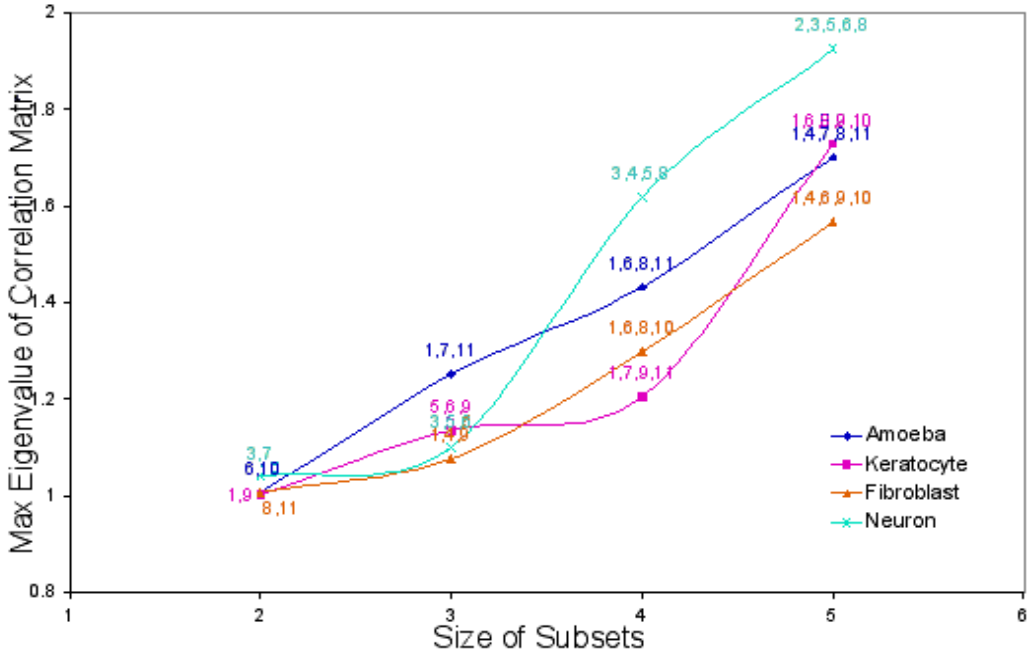

Figure 3.2: Second Best Subsets of Cell Metrics

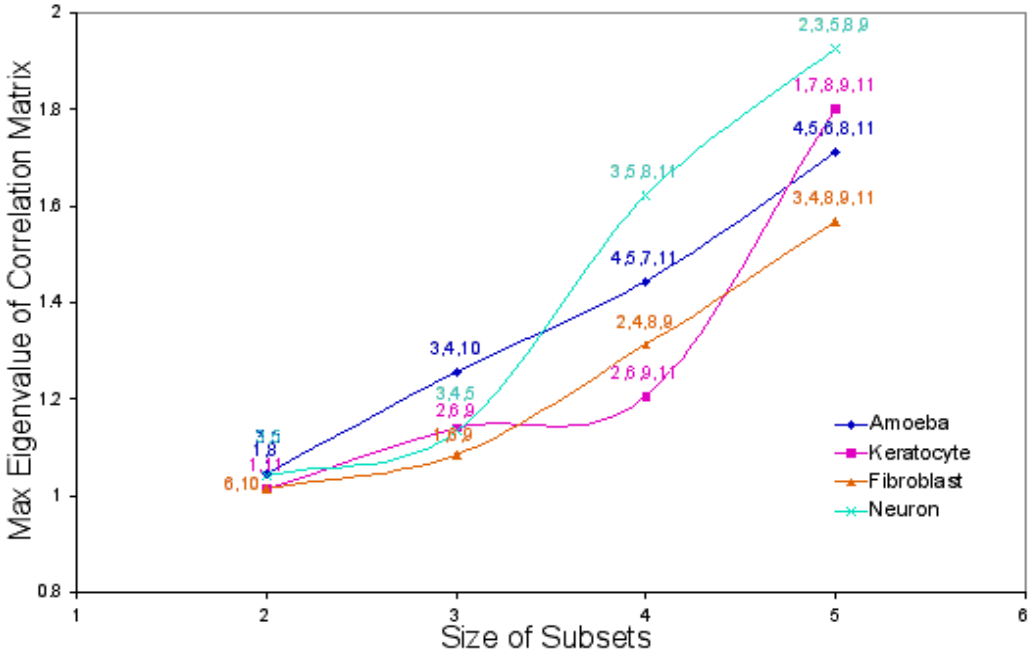

Figure 3.3: Third Best Subsets of Cell Metrics

Chapter 4

Minimizing a Nonlinear Stochastic Function

In unconstrained optimization, the goal is to find the point $\mathbf{x}^* = (x_1^*, x_2^*, ..., x_n^*)$ which minimizes or maximizes the objective function $f(x_1, x_2, ..., x_n)$ where there is no constraint on any of the variables $x_1, ..., x_n$. Algorithms for unconstrained optimization problems require an initial starting point, $\mathbf{x^s}$, that is a reasonable approximation to the solution. Although the function may have many local minima, we seek a method that will be able to converge to the global minimum. There are many effective algorithms available for unconstrained optimization problems; however, the speed and accuracy of the algorithm depend on the type of function being optimized. The number of variables, smoothness of the function, and expense of function evaluations can greatly increase the difficulty in optimization. In this chapter we will discuss unconstrained optimization methods which require evaluation of the function's derivatives and those that use a direct search strategy.

4.1 Minimization with Derivatives

Two main approaches in unconstrained minimization algorithms are *line search* and *trust region methods* [10]. *Line search methods* first choose a direction and then determine a step length that will produce a new iterate which decreases the function. *Trust region methods* use a model function, usually a quadratic, that behaves like the objective function in a region around the current iterate. First a step length is chosen within the trust region, and then a direction is determined by minimizing the function over all directions.

Among the *line search methods* there are many ways of obtaining the search direction. For example, the steepest-descent direction is given by $- \bigtriangledown f_k$ which requires calculation of the function's gradient. On more complex functions, the Newton direction, $-\bigtriangledown^2 f_k^{-1} \bigtriangledown f_k$, may be of interest. The Newton direction can be found only if the Hessian of the function's second derivatives can be computed. Alternatively, there are *Quasi-Newton* search directions which approximate the Hessian based on information from the previous iterate. Both the Newton and *Quasi-Newton* methods require that the Hessian or approximate Hessian matrix be positive definite.

4.1.1 Finite Difference Method

Line search methods rely on information about the function's first derivatives and sometimes its second derivatives. In order to approximate the derivatives of our *L2Error* function in *SimMigration*, we use finite differences which only require two function evaluations at each iteration. For each input parameter, we compute *L2Error* at its initial value, x_i, then at its initial value increased by its step size, $x_i + h_i$, and at its initial value decreased by its step size, $x_i - h_i$. The forward, backward, and central difference formulas were used to approximate the derivatives

for each input parameter as follows.

Let x_i be the i^{th} input parameter and h_i be its step size. Let $f_{j-1} = f(x_i - h_i)$, $f_j = f(x_i)$, and $f_{j+1} = f(x_i + h_i)$. Then the three finite difference formulas are:

Forward Difference

$$f' \approx \frac{f_{j+1} - f_j}{h_i}$$

Backward Difference

$$f' \approx \frac{f_j - f_{j-1}}{h_i}$$

Central Difference

$$f' \approx \frac{f_{j+1} - f_{j-1}}{2h_i}$$

The finite difference methods were performed on the *L2Error* function for Amoeba input data. Table 4.1 shows the target values and step sizes for each input parameter. The derivatives obtained from the forward, backward, and central difference formulas for each input parameter are shown in Table 4.2.

Table 4.1: Amoeba Target Value Input Parameters and Step Sizes for Finite Difference Methods

Parameter	**Target Value**	**Step Size**
fDiffuse	0.2	0.002
fDecay	0.999	0.01
fBurst	0.003	0.00003
nBurst	4	0.04
dlnhibitorConc	3.0e-6	3.0e-8
fProtrusion	0.2	0.002
fRetraction	0.005	0.00005
fFeedback	1	0.01
nOffPoint	5	0.05

Table 4.2: *L2Error* Function Derivative Approximations using Finite Difference Methods for Amoeba

Parameter	**Forward**	**Backward**	**Central**
fDiffuse	1153.3678 ± 1.1770	-1152.2804 ± 0.6261	0.5437 ± 0.4010
fDecay	234.1970 ± 0.0796	-230.2079 ± 0.1735	1.9946 ± 0.1265
fBurst	7.6822e4 ± 3.1723e1	-7.6845e4 ± 8.8070	-11.5232 ± 15.5262
nBurst	57.6036 ± 0.0342	-57.6845 ± 0.0148	-0.0404 ± 0.0238
dInhibitorConc	7.6720e7 ± 6.3715e4	-7.6811e7 ± 1.0974e4	-4.5259e4 ± 3.3585e4
fProtrusion	1152.4893 ± 0.5684	-1151.2146 ± 1.7516	0.6373 ± 0.9127
fRetraction	46062.6504 ± 35.0438	-46049.4650 ± 55.1431	6.5927 ± 10.0640
fFeedback	230.3985 ± 0.0683	-230.1535 ± 0.1793	0.1225 ± 0.1110
nOffPoint	46.1140 ± 0.0181	-46.1374 ± 0.0320	-0.0117 ± 0.0130

The derivative approximations for each input parameter are inconsistent among the forward, backward, and central difference calculations. Thus computing derivatives of the *L2Error* function is not practicable and we must seek an alternative way to minimize the *L2Error* function which does not rely on derivative calculations.

4.2 Minimization without Derivatives

In certain cases it is more efficient to find the minimum of a function by direct evaluation when the approximation of derivatives fails. Two direct search methods for unconstrained optimization problems are the Hooke and Jeeves, and Nelder-Mead methods. Both methods can conduct searches in n dimensions for nonlinear functions.

4.2.1 Hooke and Jeeves Method

We first take a look at the Hooke and Jeeves method for finding the minimum of a nonlinear function. The algorithm we used to perform the Hooke and Jeeves method

was provided by the Netlib Library [7]. This algorithm requires the user to provide the number of the function's dependent variables, an initial guess to the minimum point, the step size, termination criteria for the step size, and the maximum number of iterations allowed.

There are two types of moves in the Hooke and Jeeves iterates [12]. The first move is an *exploratory* move that searches for a value near the initial guess which reduces the function's value. This is considered the *best point nearby*. The second move is a *pattern* move that is based on the direction established by the exploratory move. Once the *best* point is found, the search continues in its direction.

Hooke and Jeeves Algorithm

Step 1: Find a point nearby which minimizes the function. For each variable, let $x_i = x_i + \delta_i$, where δ_i is a given step length for the i^{th} variable.

Step 2: Evaluate the function at each of the x_i's. If $f(x_i + \delta_i) < f(x_i)$ then replace x_i with $x_i + \delta_i$. Otherwise evaluate $f(x_i - \delta_i)$. If $f(x_i - \delta_i) < f(x_i)$ then replace x_i with $x_i - \delta_i$. If neither of these moves produces a reduction in the function, then do not alter x_i.

Step 3: Begin Pattern Moves Calculate the new step length: $\delta_i = |x_p * \rho|$, where x_p is the starting point obtained from the initial guess or from the previous iteration. After each iteration the step length is reduced by the parameter ρ.

Stopping Conditions

- Stop when the step length is smaller than ϵ, the minimum step length allowed.
- Stop when the maximum number of iterations is reached.

Table 4.3 shows the results from applying the Hooke and Jeeves minimization algorithm to the input parameters for amoeba. First the Hooke and Jeeves method

was performed on the nine input parameters. The method failed to find any of the target values within a 10% relative error. In fact, the smallest error was 20%. Then we narrowed down the search to four parameters. The first four parameter search was performed on *fDecay*, *fBurst*, *fProtrusion*, and *fFeedback*. This search still failed to find reasonable estimates of the target values with the lowest error at 10% for *fDecay* which was our original starting point. A second four parameter search on *fBurst*, *fProtrusion*, *fRetraction*, and *fFeedback* performed better. The method was able to find the target values for *fBurst*, *fProtrusion*, and *fRetraction* within 23% of the target values. However, this search was still not able to attain a reasonable value for *fFeedback*. We then narrowed the search to only three variables *fBurst*, *fProtrusion*, and *fRetraction*. The largest error observed among these three variables was 37% in the three variable search compared to 23% in the four variable search. Although we reduced the number of variables in the search to simplify the minimization problem, the Hooke and Jeeves method was still not able to attain target values for all variables with at least a 90% accuracy. As a result we pursued an alternative minimization method.

Table 4.3: Results of Hooke and Jeeves Minimization Method on Amoeba Input Parameters

Parameter	**Target Value**	**Start Point**	**Minimum Found**	**Relative Error**
fDiffuse	0.2	0.1	0.525	163 %
fDecay	0.999	1.0	3.55	255 %
fBurst	0.003	0.004	0.0176	487 %
nBurst	4	3	3	25 %
dInhibitorConc	0.000003	0.000003	0.0000056	87%
fProtrusion	0.2	0.15	0.5325	166%
fRetraction	0.005	0.004	0.004	20%
fFeedback	1	1.7	1.7	70%
nOffPoint	5	4	4	20%
fDecay	0.999	0.90	0.90	10 %
fBurst	0.003	0.0025	0.013	333 %
fProtrusion	0.2	0.1	0.68	240 %
fFeedback	1	1.2	1.2	20 %
fBurst	0.003	0.002	0.0037	23%
fProtrusion	0.2	0.05	0.1997	0.15%
fRetraction	0.005	0.002	0.0054	16%
fFeedback	1	2	2	100%
fBurst	0.003	0.01	0.0019	37%
fProtrusion	0.2	0.08	0.205	2.5%
fRetraction	0.005	0.001	0.00399	20%

4.2.2 Nelder-Mead Method

Nelder and Mead proposed a widely used direct search method for minimizing a non-linear function in multidimensions. The Nelder-Mead method performs relatively well on functions that contain some noise. This method searches for a minimum by forming a *simplex* in n dimensions, or a convex hull of $n+1$ points in R^n [8]. The algorithm we used to perform the Nelder-Mead method was supplied by the GNU Scientific Library and is shown in Appendix A [6]. The user must supply an initial vector of estimates to the solution, the step sizes for each variable, the number of variables, and the function to minimize. From the initial vector, the algorithm constructs $n+1$ vectors in which the i^{th} vector increases the $(i-1)^{th}$ variable by its step size. These $n+1$ vectors form the vertices of the simplex. Each iteration consists of the following steps:

Nelder-Mead Algorithm

Step 1: Construct n+1 vectors from the initial vector

Let the initial guess be $\mathbf{x} = (x_0, x_1, ..., x_{n-1})$ and the respective step sizes be given by $\mathbf{s} = (s_0, s_1, ..., s_{n-1})$.

Then $v_0 = (x_0, x_1, ..., x_{n-1})$

$v_1 = (x_0 + s_0, x_1, ..., x_{n-1})$

...

$v_i = (x_0, x_1, ..., x_{i-1} + s_{i-1}, ..., x_{n-1})$

...

$v_n = (x_0, x_1, ..., x_{n-1} + s_{n-1})$

Step 2: Evaluate the function at each vertex and order the vertices by their function values

Step 3: Reflect the vector, v_i, that produces the highest function value about the

centroid of the remaining n points. Evaluate the function at the reflection vector, v_r. If $f(v_r)$ is less than the maximum function value and greater than the minimum function value, then stop. v_r becomes the new vertex.

Step 4: If $f(v_r) < f(v_l)$, where $f(v_l)$ is the minimum function value over all vertices, then expand the simplex by doubling the distance of v_r to the centroid. Evaluate the function at the expansion point v_e.

If $f(v_e) < f(v_r)$, then v_e is the new vertex. Otherwise v_r is the new vertex.

Step 5: If $f(v_r) > f(v_h)$, where $f(v_h)$ is the maximum function value over all vertices, then contract the simplex by halving the distance of v_r to the centroid. Evaluate the function at the contraction point v_c.

If $f(v_c) < f(v_r)$, then v_c is the new vertex.

Step 6: If $f(v_c) > f(v_r)$, then shrink the simplex by halving the distance of each vertex to the vertex which produces the smallest function value.

Step 7: Repeat until the termination criteria is met.

The Nelder-Mead search was performed to minimize *L2Error* with the input parameters for amoeba. In addition, all of the parameters were scaled to a number between 1 and 10, to allow the step size to be a value of 1. The initial guess for the input parameters was set to within 20% error of the target value.

First, we performed the Nelder-Mead search on one input parameter at a time while keeping the rest of the input parameters set to their target values. This reduces the simplex to only one dimension and requires less run time, about thirty minutes on a 3CPU parallel processor. Table 4.4 shows the results obtained from the one dimensional Nelder-Mead search on each of the input parameters. By searching for one parameter at a time we can differentiate between those parameters whose target values were achieved with at least 90% accuracy and those who did not achieve sufficient accuracy. Based on the information obtained from the one dimensional

Nelder-Mead search, we classified five input parameters as "good" variables and three as "bad" variables. Parameters that had a relative error of 6% or less were classified as "good" variables, otherwise they were considered "bad" variables. The five good variables were *fDecay*, *fBurst*, *fProtrusion*, *fRetraction*, and *dInhibitorConc* and the four bad variables were *fDiffuse*, *nBurst*, *nOffPoint*, and *fFeedback*.

Following the one dimensional searches we then selected subsets of the five "good" variables to conduct searches in larger dimensions. Table 4.5 shows the results from the Nelder-Mead search on *fDecay*, *fBurst*, *fProtrusion*, and *fRetraction*. All of the four variables except for *fProtrusion* converged to the their target value within a 5% relative error. Table 4.6 and 4.7 show the Nelder-Mead minimization results for five input parameters. The starting point for each input parameter is at 20% below the target value in Table 4.6, and at 10% below the target value in Table 4.7. For starting points at 20% below the target value, the method converged to the target value for *fDecay* within 1% error. For the remaining variables, *fBurst*, *fProtrusion*, *fRetraction*, and *dInhibitorConc*, the method made no progress in obtaining the target values. Reducing the starting points to only 10% below the target value improved the results for the five variable search with a convergence of *fDecay*, *fBurst*, and *fProtrusion* to within 7% of the target value.

Although the five dimensional search with starting points at 10% below the target values proved to be successful, the range of values for the starting points may be too restricting for practical use. In addition, we are not able to include more than five variables in the Nelder-Mead search due to the poor performance of *fDiffuse*, *nBurst*, *nOffPoint*, and *fFeedback*. These "bad" input parameters may have multiple local minima causing our Nelder-Mead search to fail when these parameters are added. A deeper investigation in the behavior of these input parameters may offer further valuable insight into the complications we encountered.

Table 4.4: Results from Nelder-Mead Minimization on each Amoeba Input Parameter

Parameter	Target Value	Start Point	Minimum Point	Relative Error
fDiffuse	0.2	0.24	0.2463	23.15 %
fDecay	0.999	0.80	0.9996	0.06 %
fBurst	0.003	0.004	0.0029	4.83 %
nBurst	4	4.8	6.55	63.75 %
dlnhibitorConc	0.000003	0.0000036	0.000003162	5.40 %
fProtrusion	0.2	0.24	0.2025	1.25 %
fRetraction	0.005	0.004	0.005	0.06 %
fFeedback	1	0.8	1.79	79.30 %
nOffPoint	5	4	3.996	20.08 %

Table 4.5: Results from Nelder-Mead Minimization on 4 Amoeba Input Parameters

Parameter	Target Value	Start Point	Minimum Point	Relative Error
fDecay	0.999	0.799	0.9985	0.05 %
fBurst	0.003	0.0024	0.003021	0.70 %
fProtrusion	0.2	0.16	0.1811	9.45 %
fRetraction	0.005	0.004	0.005165	3.30 %

Table 4.6: Results from Nelder-Mead Minimization on 5 Amoeba Input Parameters

Parameter	Target Value	Start Point	Minimum Point	Relative Error
fDecay	0.999	0.799	0.9911	0.79 %
fBurst	0.003	0.0024	0.002159	28.03 %
fProtrusion	0.2	0.16	0.1635	18.25 %
fRetraction	0.005	0.004	0.003782	24.36 %
dlnhibitorConc	0.000003	0.0000024	0.000002024	32.53 %

Table 4.7: Results from Nelder-Mead Minimization on 5 Amoeba Input Parameters

Parameter	Target Value	Start Point	Minimum Point	Relative Error
fDecay	0.999	0.8991	1.0006	0.16 %
fBurst	0.003	0.0027	0.002807	6.45 %
fProtrusion	0.2	0.18	0.1866	6.71 %
fRetraction	0.005	0.0045	0.004403	11.93 %
dlnhibitorConc	0.000003	0.0000027	0.000002653	11.58 %

Appendix A

Nelder-Mead Implementation in C

A.1 main.c code

```
#include "SimMigration.h"
#include "stdio.h"
#include <gsl/gsl_vector.h>
#include <gsl/gsl_multimin.h>
int
main(void)
{
        FLOAT fDiffuse;
        double L2Error;
        FLOAT relerr, tar;
        FILE *fp;
        fp = fopen("Results","w");

        // STARTING POINT //
        fDiffuse = (FLOAT) 2.20;
        tar = 2.00;
        size_t np = 1;
        double par[1] = {2.00}; //scaled from 0.20

        const gsl_multimin_fminimizer_type *T = gsl_multimin_fminimizer_nmsimplex;
        gsl_vector *ss, *x;
        gsl_multimin_function minex_func;
        size_t iter = 0, i;
        int status;
        double size;

        /* Initial vertex size vector */
        ss = gsl_vector_alloc (np);
```

```
/* Set all step sizes to 1 */
gsl_vector_set_all (ss, 1.0);

/* Starting point */
x = gsl_vector_alloc (np);

gsl_vector_set (x, 0, fDiffuse);

/* Initialize method and iterate */
minex_func.f = &MigrationSimulation_f;
minex_func.n = np;
minex_func.params = (void *)&par;

s = gsl_multimin_fminimizer_alloc (T, np);
gsl_multimin_fminimizer_set (s, &minex_func, x, ss);

do
{
  iter++;
  status = gsl_multimin_fminimizer_iterate(s);

  if (status)
  break;

  size = gsl_multimin_fminimizer_size (s);
  status = gsl_multimin_test_size (size, 1e-4);

  if (status == GSL_SUCCESS)
  {
   fprintf (fp,"converged to minimum at %d iterations \n", iter);

   for (i = 0; i < np; i++)
   {
    fprintf (fp, "Parameter value: %10.3e ", gsl_vector_get (s->x, i));
    relerr = (tar - gsl_vector_get (s->x, i))/tar*100;
    fprintf (fp, "Relative Error: %.3f \n", relerr);
   }
   fprintf (fp, "f() = %7.3f size = %.3f \n", s->fval, size);
  }
}
while (status == GSL_CONTINUE && iter < 100);
```

```
        gsl_vector_free(x);
        gsl_vector_free(ss);
        gsl_multimin_fminimizer_free (s);
        fclose(fp);

return status;
}
```

Bibliography

[1] Janardhan A., Swigut T., and Hill B. et al. HIV-1 Nef Binds the DOCK2-ELMO1 Complex to Activate Rac and Inhibit Lymphocyte Chemotaxis. *PLOS Biology Vol. 2*, pages 65–76, January 2004.

[2] Alberts B., Bray D., and Lewis J. et al. *Molecular Biology of the Cell*. Garland Publishing, New York, NY, 1994.

[3] Luster A. D., Alon R., and von Andrian U. H. Immune cell migration in inflammation: present and future therapeutic targets. *Nature Immunology Vol. 6*, pages 1182–1190, December 2005.

[4] Palsson E. and Othmer H. G. A model for individual and collective cell movement in *Dictyostelium discoideum*. *PNAS Vol. 97*.

[5] Solomon E.P., Berg L.R., and Martin D.W. et al. *Biology, Fourth Edition*. Saunders College Publishing, Fort Worth, Texas, 1996.

[6] Galassi M. et al. GNU Scientific Library Reference Manual, Second Edition.

[7] Johnson M. G. Hooke and Jeeve's direct search. *CACM*, June 1963.

[8] Press W. H., Teukolsky S. A., Vetterling W. T., and Flannery B. P. *Numerical Recipes in C: The Art of Scientific Computing Second Edition*. Cambridge University Press, New York, NY, 1992.

[9] Mitchison T. J. and Cramer L. P. Actin-Based Cell Motility and Cell Locomotion. *Cell Vol. 84*, pages 371–379, February 1996.

[10] Nocedal J. and Wright S. J. *Numerical Optimization*. Springer, New York, NY, 1999.

[11] Yanai M., Kenyon C. M., and Butler J. P. et al. Intracellular pressure is a motive force for cell motion in Amoeba proteus. *Cell Motility and the Cytoskeleton Vol. 33*, pages 22–29, Decmber 1998.

[12] Hooke R. and Jeeves T. A. Direct search solution of numerical and statistical problems. *Journal of the Association for Computing Machinery (ACM)*, pages 212–219, 1961.

[13] Chen X., Huang B. Q., and Splinter P. L. et al. Cdc42 and the Actin-Related Protein/Neural Wiskott-Aldrich Syndrome Protein Network Mediate Cellular Invasion by *Cryptosporidium parvum*. *Infection and Immunity Vol. 72*, pages 3011–3021, May 2004.