Investigation of a Simulated Annealing Cooling Schedule used to Optimize the Estimation of the Fiber Diameter Distribution in a Peripheral Nerve Trunk

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Engineering,

Specialization in Biomedical Engineering

by

Arya Vigeh

May 2011

COMMITTEE MEMBERSHIP

TITLE:	Investigation of a Simulated Annealing Cooling Schedule used to Optimize the Estimation of the Fiber Diameter Distribution in a Peripheral Nerve Trunk
AUTHOR:	Arya Vigeh
DATE SUBMITTED:	May 2011
COMMITTEE CHAIR:	Dr. Robert Szlavik, Associate Professor, Biomedical and General Engineering
COMMITTEE MEMBER:	Dr. David Clague, Associate Professor, Biomedical and General Engineering
COMMITTEE MEMBER:	Dr. Scott Hazelwood, Associate Professor, Biomedical and General Engineering

ABSTRACT

Investigation of a Simulated Annealing Cooling Schedule used to Optimize the Estimation of the Fiber Diameter Distribution in a Peripheral Nerve Trunk

Arya Vigeh

In previous studies it was determined that the fiber diameter distribution in a peripheral nerve could be estimated by a simulation technique known as group delay. These results could be further improved using a combinatorial optimization algorithm called simulated annealing. This paper explores the structure and behavior of simulated annealing for the application of optimizing the group delay estimated fiber diameter distribution. Specifically, a set of parameters known as the cooling schedule is investigated to determine its effectiveness in the optimization process.

Simulated annealing is a technique for finding the global minimum (or maximum) of a cost function which may have many local minima. The set of parameters which comprise the cooling schedule dictate the rate at which simulated annealing reaches its final solution. Converging too quickly can result in sub-optimal solutions while taking too long to determine a solution can result in an unnecessarily large computational effort that would be impractical in a real-world setting.

The goal of this study is to minimize the computational effort of simulated annealing without sacrificing its effectiveness at minimizing the cost function. The cost function for this application is an error value computed as the difference in the maximum compound evoked potentials between an empirically-determined template distribution of fiber diameters and an optimized set of fiber diameters. The resulting information will be useful when developing the group delay estimation and subsequent simulated annealing optimization in an experimental laboratory setting.

Keywords: simulated annealing, group delay, fiber diameter distribution, cooling schedule, optimization algorithm

Acknowledgments

First and foremost I would like to thank my advisor Dr. Robert Szlavik for the countless hours spent helping me throughout the duration of this project. His guidance and enthusiasm for this project were an important motivating factor for me to complete this work. Thanks also to my committee members Dr. David Clague and Dr. Scott Hazelwood for taking the time out of their busy schedules to review my work and participate in my thesis defense.

To all the friends I have made during my time in San Luis Obispo, thank you for the amazing memories that I will never forget. I can’t wait to see what lies ahead for all of us.

Being fortunate enough to come from such a large family, I must also acknowledge all my cousins, aunts, uncles, grandparents, my brother-in-law and all our close family friends who have always been there for me no matter what the situation may be. Each and every one of you has had a positive impact on my life that has shaped me into the person I am today and for that I am forever grateful.

To my sister for always looking out for me whenever I asked for help but more importantly, whenever I didn’t. There is no way I could have made it this far without your guidance. Also, thank you for bringing my beautiful niece into this world. She has become a new light in my life.

Last and certainly not least, thank you to my parents for their unconditional love and support throughout my life and for helping me understand all the things that matter most in this world.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 The Nervous System

The nervous system is known as the communication and control system of the body. Its three primary functions are to receive sensory input, interpret the information received (known as integration) and finally perform some action in response to that signal. Possible responses include storing the signal for later processing, ignoring the signal or generating a motor response. An example of a simple reflex circuit is revealed when a person touches something that is extremely hot. Receptors on the skin respond to the heat by sending a signal to interneurons in the spinal cord where it is interpreted as pain. These interneurons project further into motor neurons that control muscle fibers in the arm which can cause an involuntary contraction that moves the hand away from the source of heat. More sophisticated examples of neural circuits involve processing the sensory input in the brain where a response can be sent back out to motor neurons innervating muscle fibers [1]. A description of what these signals look like and how they travel across the body will be provided in a later section.

The nervous system is divided into two parts: the central nervous system (CNS) and peripheral nervous system (PNS). The CNS consists of the brain and spinal cord and is essentially the control center of the entire nervous system. Signals transmitted from the CNS are transferred to other parts of the body through the PNS which consists of branches of nerve fibers extending out from the spinal cord. These fibers are also known as axons and are a part of the fundamental unit of the nervous system, the neuron.

1.1.1 Structure of a Neuron

Neurons, or nerve cells, are the primary structural unit of the nervous system. The structure of a neuron can be seen in Figure 1. A key property of this type of cell is its ability to be electrically excited. Neurons can communicate with one another using electrochemical signaling in the synapse which is the region between the terminal branch of one neuron (pre-synaptic cell) and the dendrites of a nearby neuron (post-synaptic cell) [1]. The signals travelling along neural pathways during excitation can be represented as electrical changes in the transmembrane potential (or just membrane potential). This membrane potential is defined as the voltage difference between the inside and outside of the nerve cell [2]. More information on this potential will be presented later.

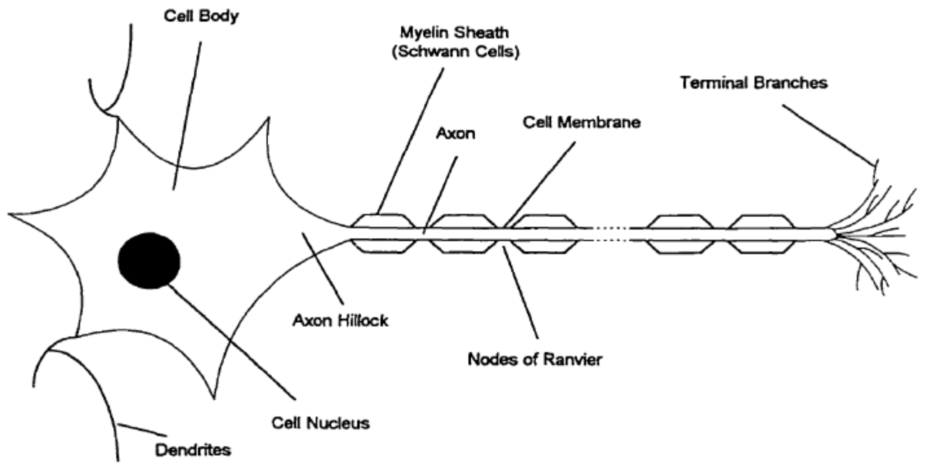


Figure 1 Structure of a single neuron (nerve cell) [3]

1.1.1.1 Dendrites

An important structure of nerve cells is the dendrites. These are branch-like structures that extend out from the cell body and receive input passed on from nearby neurons. A typical neuron may contain hundreds of these projections providing a greater surface area for receiving signals from other neurons. The dendrites role in signaling is to forward the received signal to the cell body [1].

1.1.1.2 Cell Body

The cell body, or soma, houses the nucleus of the neuron as well as other organelles necessary for proper function. The size of the soma ranges from 5 to 140 µm [1]. An important feature of the soma is the axon hillock where action potentials are generated. Action potentials refer to the rapid change in the transmembrane potential along a neuron during excitation. When the voltage at the axon hillock relative to the extracellular region reaches a particular threshold, an action potential is generated which travels along the length of the axon. This transient change in membrane potential is the basis for cell signaling in the nervous system [2].

1.1.1.3 Axon

The axon’s primary role is to transmit action potentials over long distances [2]. Axons can be very short or as long as several feet. A long axon is also referred to as a nerve fiber [1]. A typical nerve fiber is surrounded with a whitish, fatty covering called a myelin sheath. These coverings are produced by Schwann cells in the peripheral nervous system and help insulate the axon from surrounding tissue as well as allowing for faster signal conduction [1].

1.2 Communication between Neurons: The Action Potential

The propagation of an action potential along networks of nerve fibers is seen through changes in cell membrane potential of the contributing neurons. As previously stated, the membrane potential is defined as the voltage inside the cell relative to the voltage outside the cell. This value can be found experimentally by placing a voltage-sensing microelectrode inside a cell and a reference electrode somewhere in the nearby extracellular fluid. If the cell is not being activated by a propagating action potential, the resting membrane potential can be measured using a voltmeter [2].

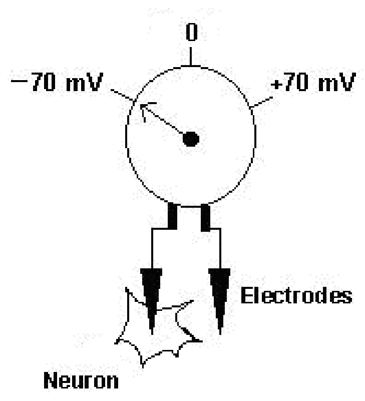


Figure 2 Measuring Cell Membrane Potential

1.2.1 Resting Membrane Potential

There are two primary factors that determine the value of the resting membrane potential. The first is the concentration of specific ions inside and outside the cell membrane (sodium, potassium and chloride being the most important). The resting membrane potential lies somewhere between the Nernst equilibrium potential for sodium (about 58 mV) and the Nernst equilibrium potential for potassium and chloride (about -80 mV). The second factor which determines the resting membrane potential is the relative ionic permeability for these three ions. In other words, the ease with which sodium, potassium, or chloride can cross through the membrane can sway the value of the resting membrane potential for a given cell. The permeability of a membrane to a specific ion is a function of the number of channels that allow the ion to cross and the ease with which the ion can pass through a single channel [2].

A formula for calculating the resting membrane potential for a given cell is known as the Goldman equation and can be simplified into the following:

$$E_m = (58mV)log(\frac{[K^+]_o + b[Na^+]_o}{[K^+]_i + b[Na^+]_i}) \tag{1}$$

where $[K^+]_o$ is the extracellular concentration of potassium,
$[K^+]_i$ is the intracellular concentration of potassium,
$[Na^+]_o$ is the extracellular concentration of sodium,
$[Na^+]_i$ is the intracellular concentration of sodium
and b is the ratio of sodium to potassium permeability

The resting transmembrane potential for a neuron is approximately -70 mV when $[K^+]_o$ = 5 mM, $[K^+]_i$ = 125 mM, $[Na^+]_o$ = 120 mM, $[Na^+]_i$ = 12 mM, and b = 0.02 [2].

1.2.2 Action Potential Propagation

The permeability of ions such as sodium and potassium across the cell membrane of a neuron are not static quantities. Instead, they are functions of both the transmembrane potential and time. Action potentials are initiated by a stimulus which causes a reduction in the membrane potential. In other words, the value of the membrane potential becomes more positive. This is known as depolarization and can be caused either by a pre-synaptic neuron releasing neurotransmitters at the synapse or through artificial current injection [2].

A small depolarization is usually not sufficient for generating an action potential. Instead, a depolarization of about 10-20 mV is required to initiate the action potential response. For a typically neuron (with resting membrane potential of -70 mV), this threshold is around -60 to -50 mV. This threshold is especially important at the axon hillock of the neuron because it is at this location of the neuron where action potentials are initiated [2].

When a neuron has reached the depolarization threshold, voltage-gated sodium channels open which results in an influx of sodium ions across the cell membrane. Sodium ions travel inside the cell because of the concentration gradient that exists across the membrane. This causes further depolarization due to the increase of positive ions entering the cell. Soon after, when the membrane potential reaches approximately +30 mV, sodium channels close and voltage-gated potassium channels open. Because potassium is at a higher concentration inside the cell, the opening of these channels causes and efflux of ions from the cell which returns the membrane potential back to its original resting value (after a brief hyperpolarization period). Initial ion concentrations are restored after the action potential cycle is completed by the Na-K pump which requires ATP. The opening and closing of voltage-gated sodium and potassium channels occurs along the length of the axon which is how the signal is propagated. When depolarization reaches the end of a neuron (at the terminal branches), the neuron releases neurotransmitters into the synapse which depolarize the post-synaptic cell and the action potential continues along the new neuron [2].

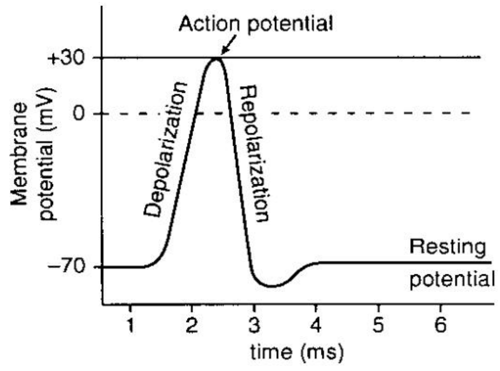


Figure 3 Action Potential Waveform

There are several key characteristics of action potentials in nerve fibers. First, in a typically mammalian nerve fiber, an action potential travels between 10-20 m/s. The amplitude of the action potential is independent of the strength of the stimulus. A strong stimulus will produce the same wave shape as a weak stimulus (assuming they are both sufficient enough to initiate an action potential). The only difference here is that the stronger stimulus will generate a larger frequency of action potentials along the axon. An upper bound limit exists for the state where they can be activated again [1]. This is known as the refractory period and typically lasts about 1 millisecond. During this time another action potential cannot be initiated [2].

1.3 Peripheral Nerve Disease

Peripheral nerve disease, also known as peripheral neuropathy, is estimated to affect nearly 20 million Americans [4]. It is commonly diagnosed near the hands and feet but can also affect autonomic nerves which innervate internal organs. Symptoms include but are not limited to tingling, pain, weakness, or numbness in the affected areas [5]. Approximately 30 percent of neuropathies are due to unknown causes while another 30 percent are caused by symptoms of diabetes. Autoimmune diseases, infections, heredity (e.g. Charcot-Marie-Tooth disease) and nutrition imbalance such as vitamin B12 deficiency are among the other etiologies [5].

A common tool for diagnosing peripheral neuropathies is the nerve conduction velocity (NCV) test. The idea behind NCV studies is to measure the speed at which electrical impulses travel along the tested nerve. A stimulating electrode that can either be placed on the surface of the skin or under the skin using a needle activates the nerve fibers while recording electrodes are placed further along the nerve path. The distance between the electrodes and the time it takes for electrical signals to travel between the electrodes are used to calculate the conduction velocity. Slower than normal transmission can be attributed to axonal loss or damage to the myelin sheath which are symptoms of certain neuropathies [6].

Peripheral nerve diseases can be further classified by the size and type of nerve fibers they affect. Nerve trunks are composed of large myelinated axons, small myelinated axons, and small unmyelinated axons. Table 1 describes some differences between the three.

Table 1 Nerve Fiber Sizes and Function [7]

Fiber Diameter Size	**Axon Type**	**Function**
Large, Myelinated	Motor and Sensory	Motor functions, vibration sense, proprioception, light touch
Small, Myelinated	Autonomic and Sensory	Light touch, pain, temperature
Small, Unmyelinated	Sensory	Pain and temperature

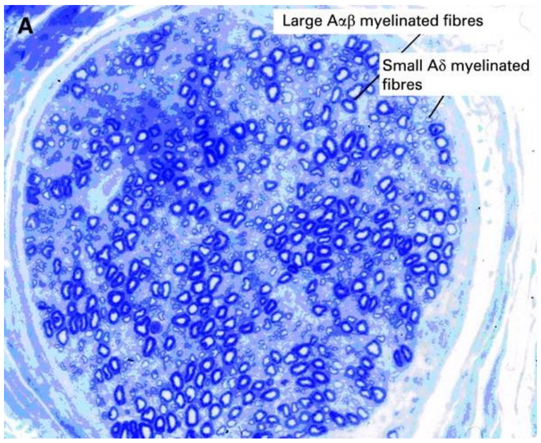


Figure 4 Cross-section of a Peripheral Nerve [8]

One of the more common neuropathies targeting large nerve fibers is Chronic Inflammatory Demyelinating Polyneuropathy (CIDP) which is an autoimmune disease that destroys the myelin sheaths within peripheral nerves. The prevalence of CIDP is underestimated due to the limitations of electrophysiological techniques to diagnose the disease. The heterogeneity of this particular neuropathy prevents the standard conduction velocity test from providing an accurate diagnosis. As a result, there have been a growing number of patients left undiagnosed despite the progression of their symptoms [9].

In contrast to CIDP which primarily impacts large nerve fibers, certain diseases have been reported to impact smaller diameter fibers. For example, neuropathies related to diabetes have been found to affect small fibers as a result of high blood glucose levels (hyperglycemia) [10]. It is estimated that over half of patients suffering from diabetes will develop a related neuropathy within 25 years after diagnosis [11].

While the nerve conduction velocity test is useful for determining the functionality of a nerve trunk, it does not provide any additional information about the individually affected fibers which together contribute to conducting the compound evoked potential in the nerve. A method that determines the fiber diameter distribution of activated fibers in a nerve trunk could give clinicians a better diagnostic tool against neuropathies [12].

1.4 Group Delay

A minimally invasive electrodiagnostic tool for determining the nerve fiber (axon) diameter distribution in a peripheral nerve was proposed by R.B. Szlavik in 2008 using a technique called group delay. This method is based on the decomposition of the maximal compound evoked potential using a setup similar to the standard nerve conduction velocity test.

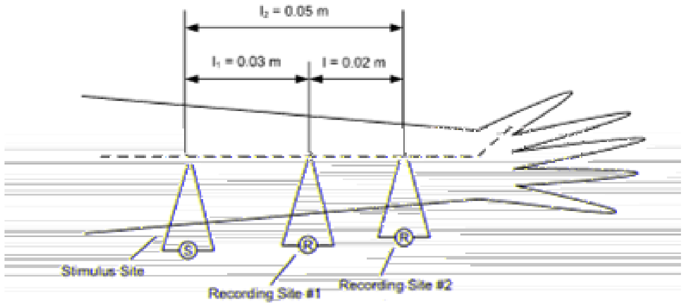


Figure 5 Group Delay Setup [12]

1.4.1 Group Delay Method

The setup for the group delay method is shown in Figure 5. A stimulating electrode is placed over a subcutaneous nerve trunk with two additional electrodes placed distally along the median nerve used for recording the evoked potentials. The distances shown in the figure are merely for simulation purposes and are not intended to be to scale. Starting with a very small current that gradually increases at the stimulus site, evoked potentials are recorded at both sites. Assuming the increase in current at each step is small enough, each successively recorded compound evoked potential is broken down into single fiber action potentials based on the previously recorded compound evoked potential. Once all possible fibers are stimulated at a sufficiently high stimulus current, the estimated group delay between the two recording sites at each current step where a new fiber was recruited are used to obtain an estimation of the diameters of each activated fiber within the nerve trunk [12]. The size distribution of nerve fibers is linearly related to the conduction velocity distribution [13].

The theoretical effectiveness of this proposed technique was determined by R.B. Szlavik using a population of randomly generated fiber diameters described by the following distribution:

$$p_k(d_k) = \sum_{h=1}^{4} \frac{\beta_h}{\sigma_h \sqrt{2\pi}} \exp\left[-\frac{(d_k - \mu_h)^2}{2\sigma_h^2} \right] \tag{2}$$

The values for each variable are provided in [12] and the distribution formed the template fiber population. A flowchart of the group delay algorithm can be seen in Appendix A.

1.4.2 Initial Group Delay Results

The template fiber diameter distribution was compared to the distribution determined by the group delay estimation and it was determined that the technique could retrieve the fiber size distribution with reasonable accuracy even in the presence of noise. Results were quantified using two different methods. The first compared the maximum compound evoked potential generated by the template distribution of nerve fiber diameters with the maximum compound evoked potential generated by the fiber diameter distribution estimated by group delay. This value will be further referred to as the final error and its calculation will be covered in a later section. The second method for quantifying results compares the actual fiber diameter distributions between the template and the group delay estimation and is called the chi-squared test [12, 14].

1.4.3 Modification to the Group Delay Technique

An improvement was made to the original results of the group delay technique by R.B. Szlavik and G.E. Turner using an optimization algorithm known as simulated annealing. Optimization algorithms such as simulated annealing are typically used to solve problems where the computational effort grows exponentially with the size of the problem. A detailed explanation of simulated annealing is provided in the next section. This particular algorithm is used to randomly vary the diameters found using group delay which as a result changes the associated single fiber evoked waveform for the purposes of obtaining a better fit between the template maximum compound evoked potential and the maximum compound evoked potential determined through group delay [14].

1.5 Optimization of Group Delay Data

1.5.1 Combinatorial Optimization

Combinatorial Optimization problems are characterized as having a finite set of solutions where one (sometimes multiple) solution is considered the best or optimal. Although the set of solutions is finite, it is typically large enough to be unable to explore exhaustively [15]. Optimality is determined by a cost function for which every solution in the set can be assigned a value. The goal therefore of an optimization problem is to determine the minimum cost and its corresponding solution [16]. Possibly the most commonly discussed example of a combinatorial optimization problem is the Traveling Salesman Problem (TSP). In this problem, a salesman starts from his home city and visits N number of cities on a prescribed list one time before returning home. The idea then is to determine a solution which describes the shortest path he can take.

The fundamental problem when attempting to find the optimal path to the Traveling Salesman Problem is that the computational effort increases exponentially with the number of cities on the tour. Consequently, accurate solutions can only be found when N is in the range of several hundred cities or less [17]. The TSP falls into a specific class of problems known as NP-complete (non-deterministic polynomial time complete). An NP-complete problem has two important characteristics. The first is that an optimal solution cannot be found in a reasonable amount of computational time. The second feature is that an algorithm for solving one NP-complete problem can be used to solve any other problem that is also NP-complete [17].

One of two methods is typically employed for solving large NP-complete problems though they do not have to be mutually exclusive. One option is to find the most optimal solution at the risk of large computational time. Optimization algorithms typically fall into this category. The second option is to find a reasonable solution as quickly as possible at the risk of sub-optimality. This particular technique is done by approximation algorithms. Furthermore, combinatorial optimization algorithms can be either general or tailored. General algorithms can be applied to a wide range of problems which makes them problem independent. Tailored algorithms on the other hand use problem-specific details which restrict the set of problems to which they can be applied. An example of a low quality general approximation algorithm is local optimization (or local search) which is briefly discussed in the next section. Simulated annealing is also a general algorithm but considered to be of much higher quality. Its performance is asymptotically viewed as an optimization algorithm but in practical implementations, it can behave as an approximation algorithm [15].

1.5.2 Local optimization

Local optimization is one technique used to solve combinatorial optimization problems, albeit not very well. It begins with an arbitrary determined solution to the given problem and a neighborhood structure consisting of solutions that are similar to the initial one. As mentioned before, solutions can be evaluated according to a cost function related to the problem being optimized. The algorithm then makes a small change to the solution (also referred to as exploring a neighboring solution) and determines what the new cost is. If the new value is an improvement from the original, the change is accepted and the procedure repeats from the new location in the solution set. This is known as a downhill movement because the algorithm is moving towards a lower value of the cost function. If the new value is larger than the previous, the change is ignored and the original solution is maintained. The algorithm terminates when no more improvements can be made to the cost function. This technique is sufficient for finding the local minimum of a problem but not for determining the global minimum. The reason why is because once the algorithm reaches a local minimum, it is essentially trapped in a neighborhood of solutions unless it accepts a change which increases the cost function. This type of movement is referred to as an uphill change and is not allowed by the local optimization algorithm. Simulated annealing, by contrast, performs similarly in that it will accept all changes to the system that reduce its cost function but also accepts some changes which increase the cost function. The decision on whether or not an increasing cost change is accepted will be discussed later. Simulated annealing's ability to make uphill changes to the system's cost function drastically improves the performance of the algorithm and makes finding the global minimum of a solution set possible [15].

1.5.3 Simulated Annealing

The simulated annealing algorithm was first derived by Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983 however it is based off of the Monte Carlo method first described by N. Metropolis in 1953.

The name of this algorithm comes from the analogous process of a liquid freezing and crystallizing or a metal cooling and annealing. At high temperatures, a liquid consists of randomly dispersed molecules resulting in a high energy state. When carefully decreasing the temperature from this point, the particles slowly arrange themselves into a highly structured lattice (solid phase). It is crucial throughout this process that the system reaches a steady state before decreasing the temperature to the next level. When the temperature is sufficiently low enough, the system reaches its ground state or the point at which the energy of the solid is minimized. If cooling is not performed slowly enough, the system is no longer at its minimal energy state, analogous to the process of quenching [16].

It is important before moving forward to bridge the gap between the physical analogy and the simulated annealing optimization algorithm. Table 2 compares the physical annealing process with the associated terminology used in simulated annealing.

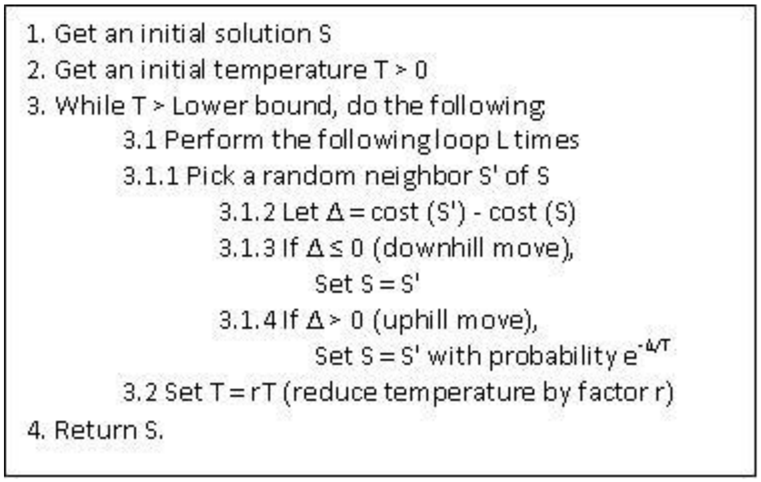
Table 2 Annealing Analogy to Optimization Problem

Physical Annealing	**Simulated Annealing Equivalent**
Arrangement of particles in a system	Solution to optimization problem
Changes in the system (i.e. a new configuration of particles)	Neighboring solution (i.e. making a small change to current solution)
Energy of the system at current configuration	Cost Function of current solution
Temperature of the system	Control Parameter or Annealing schedule

Thus the key idea behind simulated annealing is to minimize the cost function using an appropriately defined annealing schedule that “cools” the system slow enough to find the optimal solution.

1.5.3.1 Simulated Annealing Algorithm

The pseudocode in Figure 6 summarizes the general simulated annealing procedure.



```
1. Get an initial solution S
2. Get an initial temperature T > 0
3. While T > Lower bound, do the following
        3.1 Perform the following loop L times
        3.1.1 Pick a random neighbor S' of S
                3.1.2 Let $\Delta$ = cost (S') - cost (S)
                3.1.3 If $\Delta \leq 0$ (downhill move),
                        Set S = S'
                3.1.4 If $\Delta > 0$ (uphill move),
                        Set S = S' with probability $e^{-\Delta/T}$
        3.2 Set T = rT (reduce temperature by factor r)
4. Return S.
```

Figure 6 Simulated Annealing Pseudocode [18]

Note that the algorithm proceeds until the temperature reaches a lower bound set by the programmer. This value is typically set orders of magnitude lower than the starting temperature allowing a significant number of cycles to run through the algorithm before returning the optimal solution.

Simulated annealing acts similarly to local optimization in that it always accepts changes which reduce the cost. Where it improves upon local optimization is how it accepts increased cost function changes with a certain probability described by a Boltzmann distribution. That is, a system changes its configuration from E_1 to E_2 with probability p defined by:

$$p = \exp\left[-\frac{E_2 - E_1}{T}\right] \tag{3}$$

This acceptance rule is also known as the Metropolis criterion [19]. The formula can be verified because when $E_2 < E_1$, the value of p is greater than 1 and changes are always accepted. If $E_2 >$

E_1, a randomly generated number between 0 and 1 is compared to p. If the random number is less than or equal to p, the change is accepted even though the cost function has increased. One last thing to note about this distribution is its dependence on the temperature T. Because the temperature is decremented as the algorithm progresses, the likelihood of accepting an “uphill” change also decreases [16].

1.5.3.2 Annealing Schedule

While the majority of parameters which comprise the simulated annealing algorithm are fixed values that cannot be adjusted, there does exist an important set of parameters that govern the convergence of the simulated annealing algorithm which can be grouped together and referred to as the annealing schedule (or cooling schedule). Because these parameters are not fixed in the implementation of simulated annealing, they are considered to be the most important feature in the design of the algorithm [20]. In a paper published in 1991, Romeo and Sangiovanni-Vincentelli explained that an effective cooling schedule is crucial to reducing the execution time before finding an optimal solution [21].

The parameters of the cooling schedule include the starting temperature, cooling factor, stopping temperature, and the number of transitions (or moves) at each temperature [15]. Of these variables, the cooling factor is arguably the biggest determinant of the algorithm’s ability to reach an optimal solution. The cooling factor is described as the method for which simulated annealing reduces the temperature to its next value. Numerous strategies have been investigated in prior studies which examine various implementations of the cooling factor. While an exponential (or geometric) reduction is arguably the most common cooling technique, many studies have used variations of logarithmic, linear and adaptive cooling methods with positive results [20].

A study by Strenski and Kirkpatrick in 1991 on finite length cooling schedules found that geometric and linear cooling rates yielded a better result than logarithmic designs. Further, they determined that there was not a significant difference in performance between the linear and exponential implementations. With regards to the starting temperature, they concluded that excessively high initial temperatures do not greatly improve the optimization of the algorithm when a geometric cooling factor is used [22].

Another paper on simulated annealing published in 1995 by Brooks and Morgan made some important observations about the cooling schedule. They determined that doubling the number of transitions at each temperature step also doubles the execution time of the algorithm. Also, in a geometric rate of cooling, a larger scaling factor increases the reliability of the optimization process towards finding the global minimum solution set [19].

The design of a cooling schedule can be categorized one of two ways. Either all parameters are permanently set prior to the beginning of the algorithm (a static approach) or information is extracted while the algorithm is running which can alter the rate of convergence (adaptive) [23]. The aforementioned cooling strategies belong to the static category of schedules. A study published in 2007 by Ortner et al. described an adaptive simulated annealing cooling schedule for the application of object detection in imaging. The idea behind this strategy is to identify critical temperatures where the objective function is most vulnerable to optimization and decelerate the annealing schedule at these points. For all periods outside of these critical temperatures, the rate of annealing can be accelerated with little effect on the optimization process. The results of this study found that the adaptive schedule performed better than the compared geometric approach. A noticeable advantage in the adaptive cooling schedule was that increasing the value of the initial temperature did not drastically increase computational time (as it would in a geometric schedule). A disadvantage in using an adaptive schedule is that the algorithm becomes problem-dependent and cannot be used effectively in other combinatorial optimization problems [24].

1.5.3.3 More Simulated Annealing Applications

Since the development of the simulated annealing algorithm to solve combinatorial optimization problems several decades ago, numerous papers have been published which investigate the effectiveness of the algorithm to determine a minimum cost solution to various NP-complete problems. Kirkpatrick first introduced the algorithm not only as a solution to the common Travelling Salesman Problem but also as a technique for computer design. This process can be simplified into three stages: partitioning, component placement and wiring. Simulated annealing can be used to determine the partitioning of a design into groups small enough to fit an available package such as a single chip. Partitioning must be done in such a way that the number of circuits in a particular package and the number of signals that cross partition boundaries is minimized. The next step would be to assign the circuits a particular location on the chip (called placement). Optimization algorithms here attempt to minimize the length of connections in order to reduce the propagation time of a signal and increase speed. The last step is to use photolithography to assign specific routes to connect circuits together. The objective in wiring is to minimize wire lengths and any possible sources of noise such as the placement of two adjacent wires. All of these components of computer design can be assisted using simulated annealing optimization [17].

1.5.3.4 Simulated Annealing for Group Delay Data

Now that the foundation of the simulated annealing algorithm has been presented, specifics related to the optimization of group delay data can now be discussed. The input to this algorithm is the group delay estimated set of fiber diameters. The first step of the simulated annealing optimization here is to calculate the error function (previously referred to as the cost function). This is the objective function for which the algorithm attempts to minimize. It is derived in terms of the two-norm difference between the template maximum compound evoked potential and the group delay estimated compound evoked potential of all contributing fibers [14].

$$\triangle = \left\| \Psi^{(1)}(t) - \tilde{\Psi}^{(1)}(t) \right\|_2 \tag{4}$$

The next step of the procedure determines whether $\Delta < \Delta_{MIN}$ or $T < T_{MIN}$. These flags will cue the program to exit if either inequality is true. If both are false, a randomly chosen fiber from the group delay estimated set is selected and randomly assigned a new diameter. A new single fiber evoked potential is then calculated for the new diameter as well as a new compound evoked potential as a result of the change. The error function from above is then re-calculated and assigned to the variable Δ_{NEW}. If $\Delta_{NEW} < \Delta$ or a uniformly distributed random variable between 0 and 1 is less than or equal to Boltzmann probability (defined below), the fiber diameter change is accepted. If neither inequality is true, the changed fiber is discarded and replaced with its original value.

$$Y \le \exp\left[-\frac{|\Delta - \Delta_{NEW}|}{T}\right], \tag{5}$$

where Y is the uniformly distributed random number between 0 and 1

This process repeats for the number of trials per temperature step assigned by the programmer. After this limit is reached, the temperature is reduced and the procedure starts again. The algorithm exits when either $\Delta < \Delta_{MIN}$ or $T < T_{MIN}$. The choice of these boundary conditions is also up to the programmer. The output is the optimized set of fiber diameters which can be compared to the original input set before simulated annealing [14].

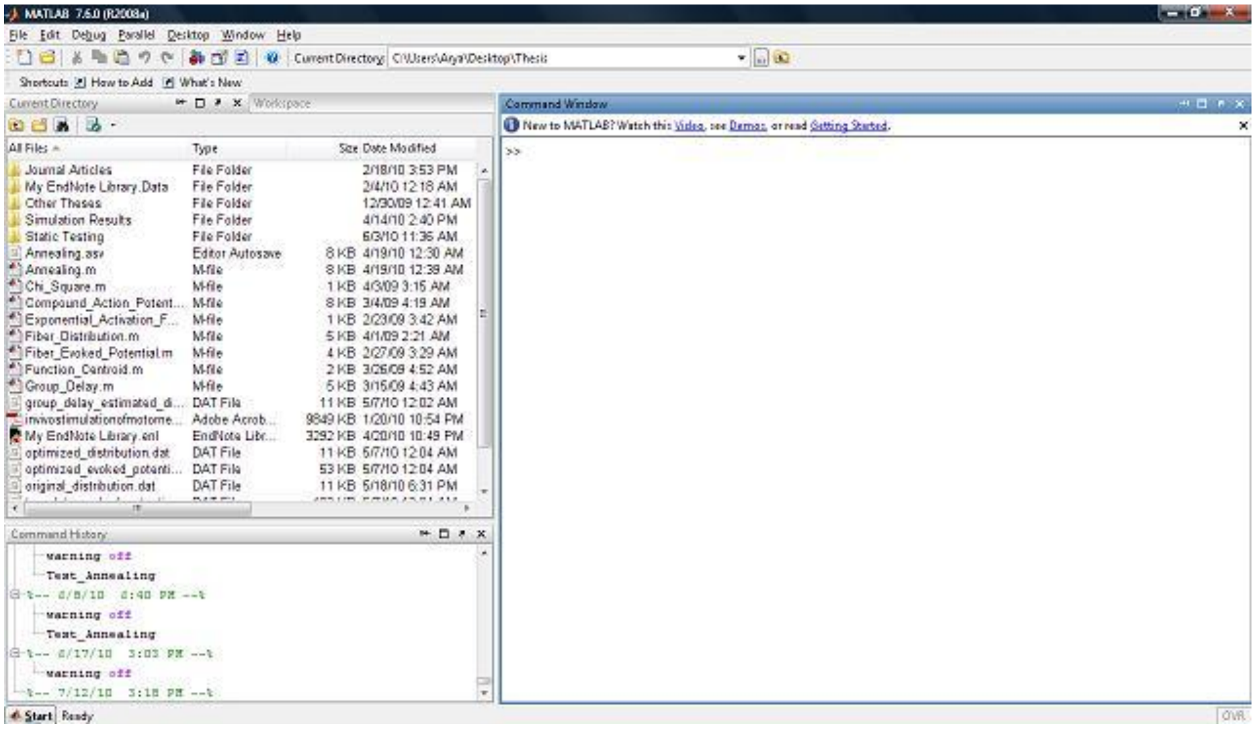
1.6 Project Scope

The work of R.B. Szlavik et al. have paved the way for further investigation regarding the efficiency of the simulated annealing algorithm for optimization of the group delay data. This thesis project will focus directly on the annealing (or cooling) schedule portion of the algorithm and attempt to determine an optimal parameter set of annealing variables which not only improves performance but also reduces the computational effort currently required to run the algorithm. Optimality will be quantified using two formulas. One compares the maximum compound evoked potentials between the estimated dataset and a template dataset while another computes a chi-squared valued to compare the fiber diameter distributions directly between the estimated and template sets.

2 Methods

2.1 MATLAB

The simulations performed in this study were executed using MathWorks' MATLAB version R2008a computing environment. MATLAB is a programming language commonly used in a variety of disciplines in industry. Some of the capabilities that can be used include implementing algorithms, plotting functions and data, calculating complex expressions and performing domain transforms. Figure 7 shows a typical console screen used to run the simulations.


MATLAB 7.6.0 (R2008a)
File Edit Debug Parallel Desktop Window Help
Current Directory: C:\Users\Arya\Desktop\Thesis
Shortcuts How to Add What's New
Current Directory
Workspace
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Command History
Start Ready
OVR

All Files	Type	Size	Date Modified
Journal Articles	File Folder		2/18/10 3:53 PM
My EndNote Library.Data	File Folder		2/4/10 12:18 AM
Other Theses	File Folder		12/30/09 12:41 AM
Simulation Results	File Folder		4/14/10 2:40 PM
Static Testing	File Folder		6/3/10 11:36 AM
Annealing.asv	Editor Autosave	8 KB	4/19/10 12:30 AM
Annealing.m	M-file	8 KB	4/19/10 12:39 AM
Chi_Square.m	M-file	1 KB	4/3/09 3:15 AM
Compound_Action_Potent...	M-file	8 KB	3/4/09 4:19 AM
Exponential_Activation_F...	M-file	1 KB	2/23/09 3:42 AM
Fiber_Distribution.m	M-file	5 KB	4/1/09 2:21 AM
Fiber_Evoked_Potential.m	M-file	4 KB	2/27/09 3:29 AM
Function_Centroid.m	M-file	2 KB	3/26/09 4:52 AM
Group_Delay.m	M-file	5 KB	3/15/09 4:43 AM
group_delay_estimated_di...	DAT File	11 KB	5/7/10 12:02 AM
invivostimulationofmotorne...	Adobe Acrob...	9849 KB	1/20/10 10:54 PM
My EndNote Library.enl	EndNote Libr...	3292 KB	4/20/10 10:49 PM
optimized_distribution.dat	DAT File	11 KB	5/7/10 12:04 AM
optimized_evoked_potenti...	DAT File	53 KB	5/7/10 12:04 AM
original_distribution.dat	DAT File	11 KB	5/18/10 6:31 PM

```
   warning off
   Test_Annealing
%-- 6/8/10  6:40 PM --%
   warning off
   Test_Annealing
%-- 6/17/10  3:03 PM --%
   warning off
%-- 7/12/10  3:18 PM --%
```

```
>>
```

Figure 7 MATLAB Console Screenshot

The original source code used to simulate the group delay technique and subsequent simulated annealing optimization was written by R.B. Szlavik and consists of the following files:

- Test_Annealing.m
- Annealing.m
- Chi_Square.m
- Compound_Action_Potential.m
- Exponential_Activation_Function.m
- Fiber_Distribution.m
- Fiber_Evoked_Potential.m
- Function_Centroid.m
- Group_Delay.m

The Test_Annealing source file is the main function of the simulation as it calls upon the other eight files to perform the specific computations necessary to compute the group delay estimation and simulated annealing optimization. Thus the simulation begins by making a call to this function first from the MATLAB console. For the scope of this project, only the Test_Annealing and Annealing source code was modified to observe changes in the simulated annealing performance. The remaining code was left unmodified in order to keep a consistent simulation environment and this code is provided in the Appendices.

2.2 Group Delay Simulation

As mentioned before, the first step of the simulation is to generate a population of nerve fiber diameters which will be referred to as the template distribution. The formula is implemented in MATLAB by calling the Fiber_Distribution function which returns the probability distribution function, the cumulative distribution function, and the vector of fiber diameters that are populated. Because fiber diameters less than 5 μm are removed, the number of generated fibers fluctuates around approximately 80 fibers. A histogram is also plotted by the function for a visual representation of the fiber diameter distribution.

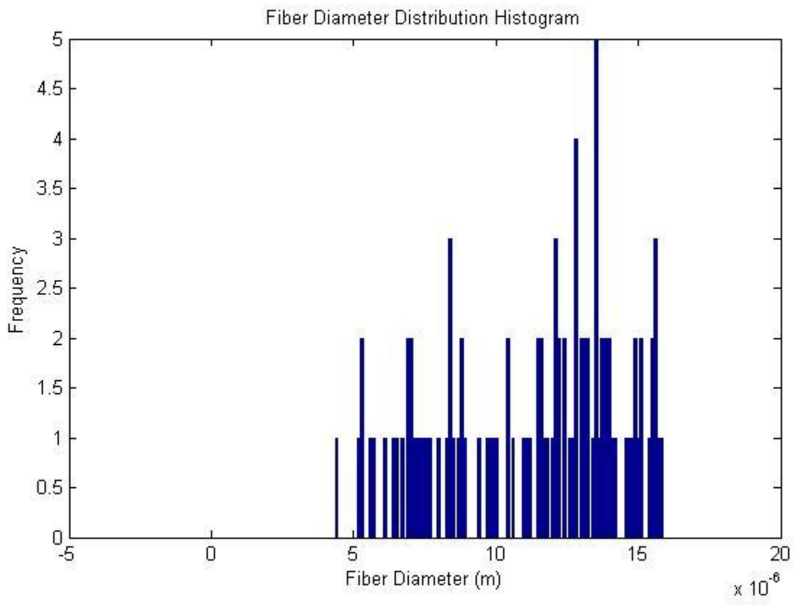


Figure 8 Example Population of Fiber Diameters

Once the template distribution is generated, the population of diameters is subjected to a series of increasing virtual current stimulus pulses to determine if a potential is evoked on the individual fibers. The stimulus current starts at 0 amperes (A) and increases in steps of 500 nA until it reaches a final current of 1.0 mA. This results in 2000 steps of an increasing stimulus current. The Exponential_Activation_Function file is called from the Compound_Evoked_Potential function to determine if the magnitude of the stimulus current is sufficient to excite any of the fibers in the vector based on its diameter. The expression in MATLAB is as follows:

```
act_function = psi_m*exp(-psi_d*d);
```

where `d` is the diameter of the fiber and `psi_m` and `psi_d` are 10 mA and 3.5 x 10^5 m^{-1}, respectively. If the stimulus current is greater than or equal to the value of the activation function, it is assumed that the action potential threshold was reached and the signal which results contributes to the compound evoked potential. This is illustrated with the following MATLAB code fragment:

```
for j = 1:length(fiber_pop)

    if stim_val >= Exponential_Activation_Function(psi_m, psi_d,
                        fiber_pop(j))
        Step = 1.0;
        act(j,i) = 1;
    elseif stim_val < Exponential_Activation_Function(psi_m, psi_d,
                        fiber_pop(j))
        Step = 0.0;
    end

    fiber_potential = Step*Fiber_Evoked_Potential(mode, c, step,
                      span, delta_fiber(j), r, radius(j), s_scale, I,
                      sigma_e, alpha);
    cap(:,i) = cap(:,i) + fiber_potential(:,2);

end
```

At each recording site, the compound evoked potential is computed by summing all single fiber evoked potentials after all steps of the stimulus current have been exhausted. A comparison of the compound evoked potentials at each stimulus step (2000 traces) for both recording sites is shown below. Note the shift in time of the waveforms between the first and second recording sites.

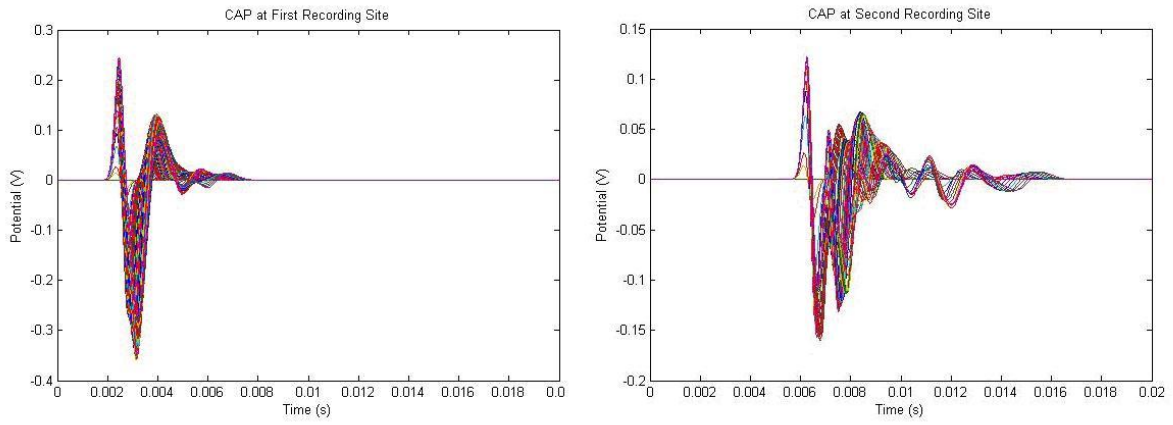


Figure 9 Compound Action Potentials at first (left) and second (right) recording sites.

Once the compound evoked potentials have been computed at each stimulus step, they are ready to be decomposed into a series of waveforms that make up the single fiber potentials at each stimulus step. This is accomplished by subtracting the compound evoked potential at each stimulus current step from the compound evoked potential at the step directly before it. Assuming the steps are small enough, the resulting waveforms after each subtraction will represent either the single fiber potentials that contribute to the new compound evoked potential or no waveform if the increase in stimulus does not recruit any addition fibers. A third scenario also exists where a current stimulus step may have recruited more than a single fiber (most common at lower current levels). The subtraction will then incorrectly combine multiple fiber potentials as a single fiber evoked potential. This behavior could also be expected in a real experiment and is included as an inherent part of the simulation study. Ideally, the subtraction process will produce the same number of non-zero waveforms (a vector of single fiber potential waveforms) as there were fibers generated in the template distribution. A comparison of the single fiber potential waveforms at both recording sites is shown below.

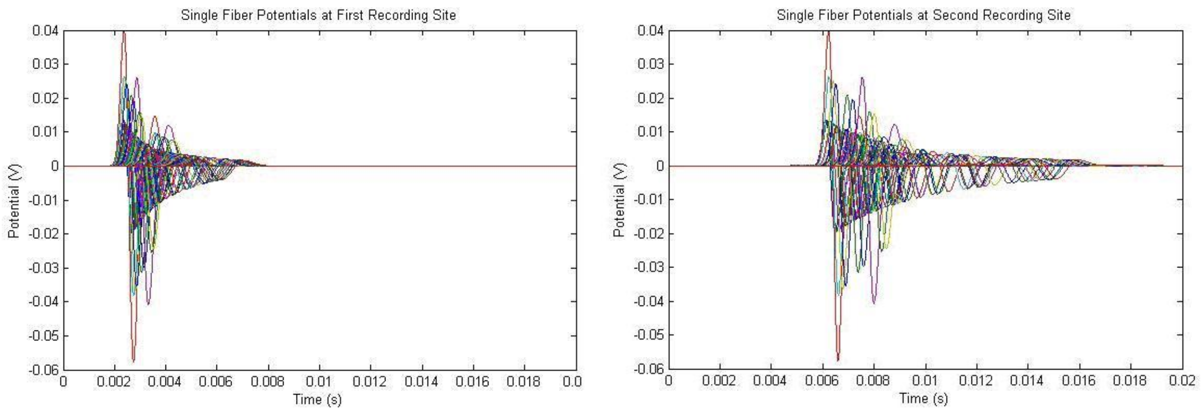


Figure 10 Single fiber evoked potentials at first (left) and second (right) recording sites.

The group delay estimation begins when the decomposition of the single fiber potentials at both recording sites is complete. The general idea behind this technique is based on control theory where a system consists of an input function (call it x(t)), an output function (y(t)), and an impulse function describing the system (h(t)). Figure 11 illustrates what the system looks like.

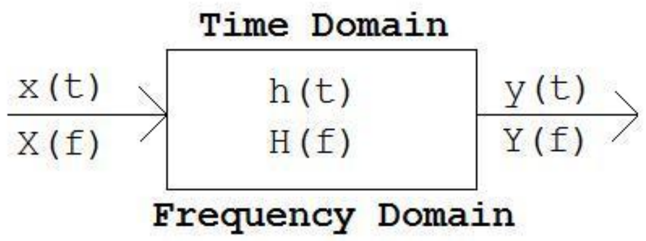


Figure 11 Control Theory Relating Input and Output

In the time domain, the impulse response h(t) is the relationship between the system output to its input. The equation for the output y(t) is:

$$y(t) = h(t) * x(t) \tag{6}$$

In other words, the output of the system is the convolution of the system's input with the system's impulse response. In the frequency domain, the frequency response H(f) is also the relationship between the system output to its input. The difference is that the output response Y(f) is now given as the product of the input's frequency response with the transfer function H(f).

$$Y(f) = H(f) \cdot X(f) \tag{7}$$

This equation can also be re-written to describe the transfer function simply as

$$H(f) = \frac{Y(f)}{X(f)} \tag{8}$$

The conversion from time domain to frequency domain is accomplished using a Fourier transformation [25].

If the group delay technique was now applied to the same theory discussed above, the single fiber evoked potentials at the first and second recording sites could be described as the inputs and outputs to the system, respectively.

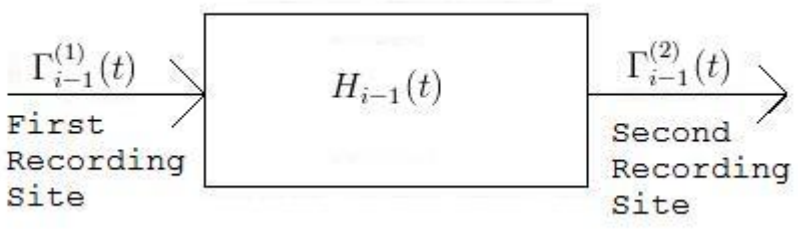


Figure 12 Relation between Group Delay Recording Sites

Using the same technique from before, we can re-write the above system after applying a Fourier Transform to describe the transfer function (or frequency response) $H_{i-1}(f)$ as:

$$H_{i-1}(f) = \frac{\mathfrak{F}[\Gamma_{i-1}^{(2)}(t)]}{\mathfrak{F}[\Gamma_{i-1}^{(1)}(t)]} \tag{9}$$

The frequency response of the system is described as the Fourier Transform of the second recording site's single fiber evoked potential divided by the Fourier Transform of the first recording site's single fiber evoked potential. Because this response consists of both a magnitude and phase, the group delay expressed as τ_{i-1} for each fiber's evoked potential is calculated as:

$$\tau_{i-1} = -\frac{1}{2\pi}\frac{d\theta_{i-1}(f)}{df} \tag{10}$$

Many previous studies have shown that there exists a linear relationship between the axon diameter and velocity of conduction in a nerve fiber [26]. Thus once the time (delay) of a single fiber evoked potential traversing between the two recording electrodes of a known distance is found, the diameter of the fiber can also be computed using the following formula:

$$d_{i-1} = \frac{l}{c\tau_{i-1}} \tag{11}$$

In this equation, *l* represents the length (in meters) between the recording electrodes and *c* is equal to 3.0 x 10^{6} s^{-1} [12]. The final product of the group delay algorithm becomes the vector of estimated fiber diameters which can be compared to the original template distribution using a chi-squared test.

2.3 Simulated Annealing Optimization

The arguments passed to the simulated annealing function (called annealing.m) include the vector of estimated fiber diameters and their respective time delays, the compound and single-fiber evoked potentials at the second recording site as well as other important variables which dictate how quickly the simulated annealing algorithm will converge to a solution. Some of these variables describe what is known as the simulated annealing cooling schedule and consist of the starting temperature, cooling factor, stopping temperature and the number of trials per temperature step.

The first step of the algorithm is to randomly select a fiber diameter from the group delay estimated vector passed to the function. This is accomplished in MATLAB using a random number generated between one and the length of the vector which represents the index of the vector of the chosen fiber.

```
selection_vector = randperm(length(fiber_pop));
```

Once the fiber is selected, a new fiber diameter is computed again using the random number generator.

```
new_fiber_diam = min_fib_diam + rand(1)*(max_fib_diam - min_fib_diam);
```

where `min_fib_diam` and `max_fib_diam` represent the smallest and largest group delay estimated fiber diameter, respectively. The formula computes a new diameter that lies somewhere between the minimum and maximum values in the vector of fiber diameters. Once a new diameter is computed, the associated time delay is also determined as before. Using the new parameter values, the single fiber evoked potential is re-calculated as well as the compound evoked potential which will also change as a result of the new fiber diameter.

In order to quantify the effectiveness of simulated annealing for optimizing the estimated fiber diameter distribution, a numeric value must be used to determine whether the change to one fiber results in a positive 'downhill' change or a negative 'uphill' change. The expression which determines this value is known as the cost function and, as previously mentioned, is calculated as the two-norm difference between the maximum compound evoked potential from the template distribution and the maximum compound evoked potential from the estimated fiber set. In other words, at every time point that is plotted, the template waveform is subtracted from the estimated waveform and those values are squared. Next these values are summed together and the square root is taken to obtain the error value. In MATLAB, the relevant code is shown below.

```
%Calculating Two-Norm Error Value
error_value = 0;

for i = 1:count
    error_value = error_value + (template(i) - compound(i))^2;
end

error_value = sqrt(error_value);
```

The loop terminates after the 2000th iteration (variable `count`) as that is the number of time points used to plot the functions in the simulation. This error value (cost function) is calculated after every diameter change to the estimated set and is compared to the previously calculated error value. Diameter changes which decrease the cost function from its previous value are always kept while changes which increase the cost function are accepted with a certain probability given by the Boltzmann distribution. In MATLAB, we simulated this as:

```
change = res_error - error_value;
oracle_value = rand(1);

if((oracle_value <= exp(-abs(change)/temp)) || (change >= 0))

...

     %Change accepted, set new error value
     res_error = error_value;

else

     %Change not accepted, so restore original values
     fiber_pop(chosen_fiber) = original_fiber_diam;
     delay_vect(chosen_fiber) = original_fiber_delay;
```

The final step of the simulated annealing loop is to decrement the temperature variable. Referring back to the analogy of annealing a metal, it is crucial for the rate of cooling to be sufficiently slow for the solid to reach a ground state at each temperature interval. Likewise in this simulation, the temperature variable must not decrease too quickly or else a sub-optimal solution may result. After the number of iterations at each temperature step has been exhausted, the following expression decrements the temperature:

```
temp = temp * temp_factor;
```

Where `temp_factor` is the ratio of “cooling” set by the programmer and varies between—but not including—0 and 1. This is an example of exponential temperature decay and is the default variation of the cooling schedule used in this optimization. The figure below illustrates exactly how many temperature steps are required at specific decrement factors between 0 and 1 given the default starting and stopping temperatures (10 and 1E-5, respectively).

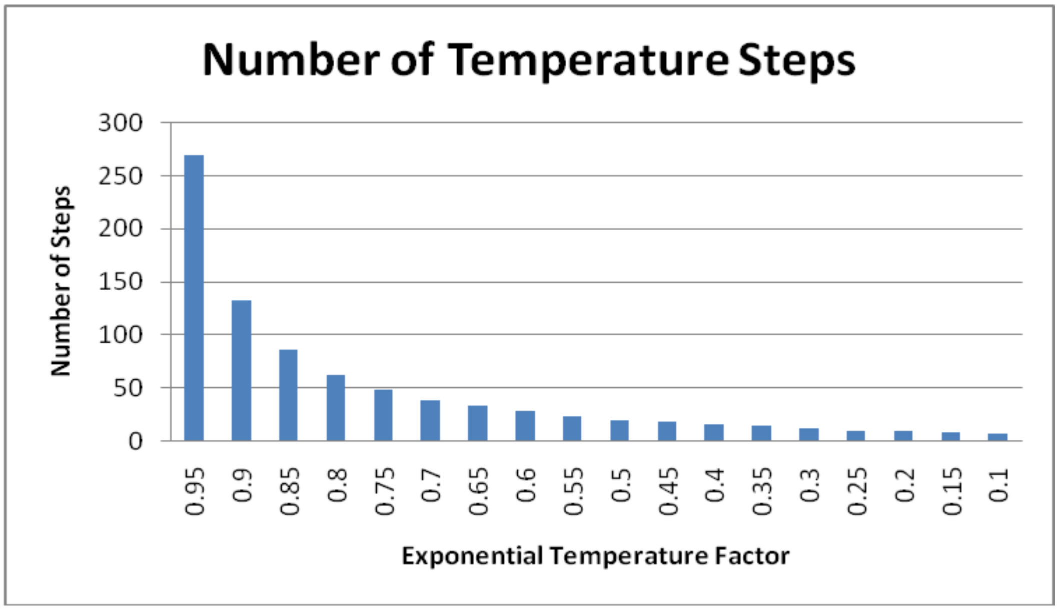
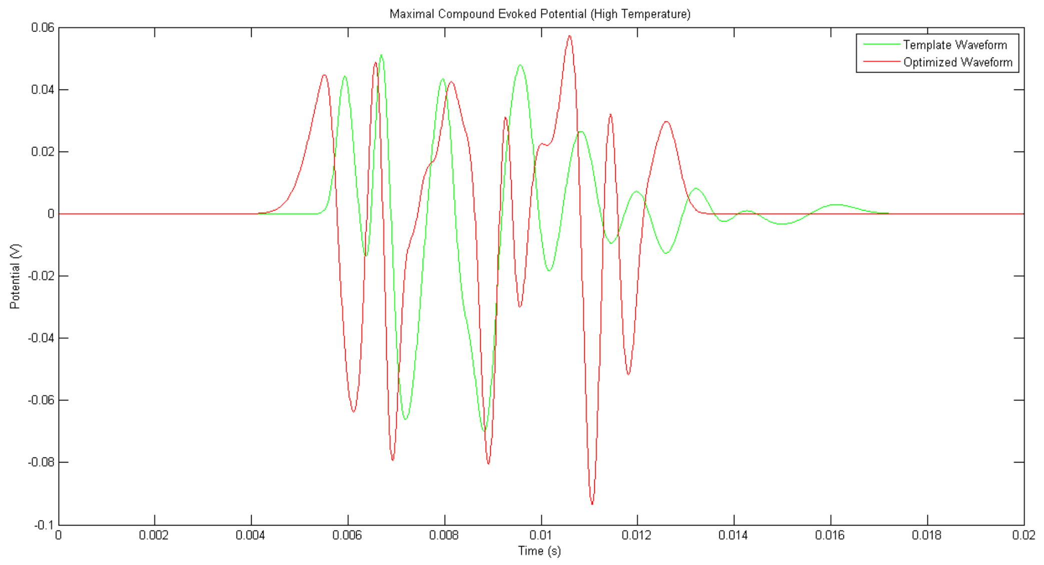


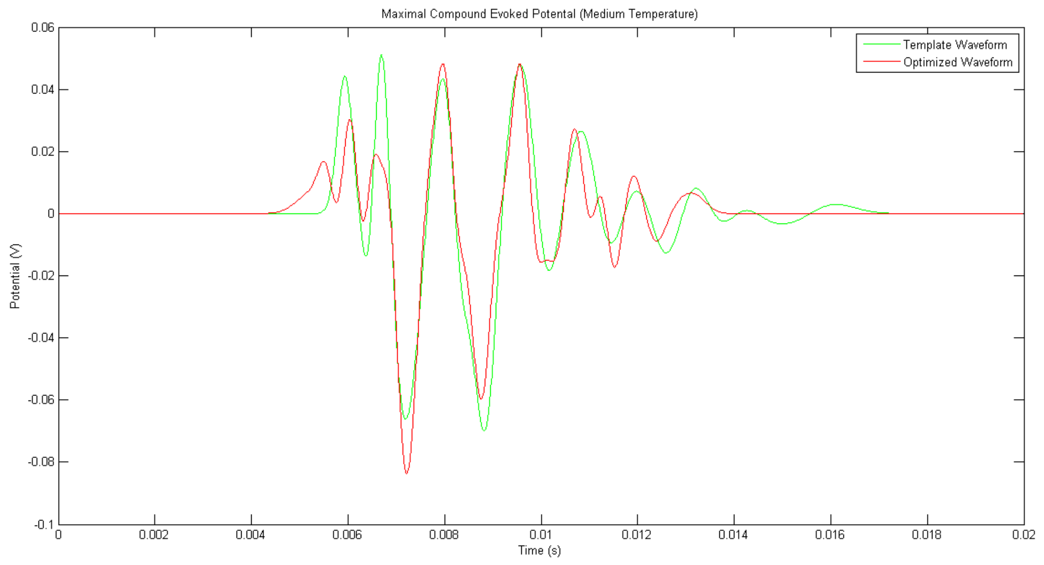
Figure 13 Number of Temperature Steps at Various Step Factors

Other variations of cooling rates have also been proposed in previous studies. Logarithmic, linear and adaptive cooling models will be tested in this study against the default exponential decay to determine if there is a significant difference in optimization performance.

2.4 Effect of Temperature on Simulated Annealing Cost Function

As mentioned before, the cost function is evaluated after every change to the estimated fiber diameter distribution. It is the ultimate goal of the simulated annealing optimization to minimize the difference between the template and estimated compound evoked potential when stimulated by the maximum current pulse. This difference fluctuates significantly during the early phases of the annealing algorithm when the temperature is high due to nearly all diameter changes being accepted. As the algorithm progresses and temperature decreases, less fiber changes are allowed which force the cost function towards an optimal solution. The following figures depict this behavior.





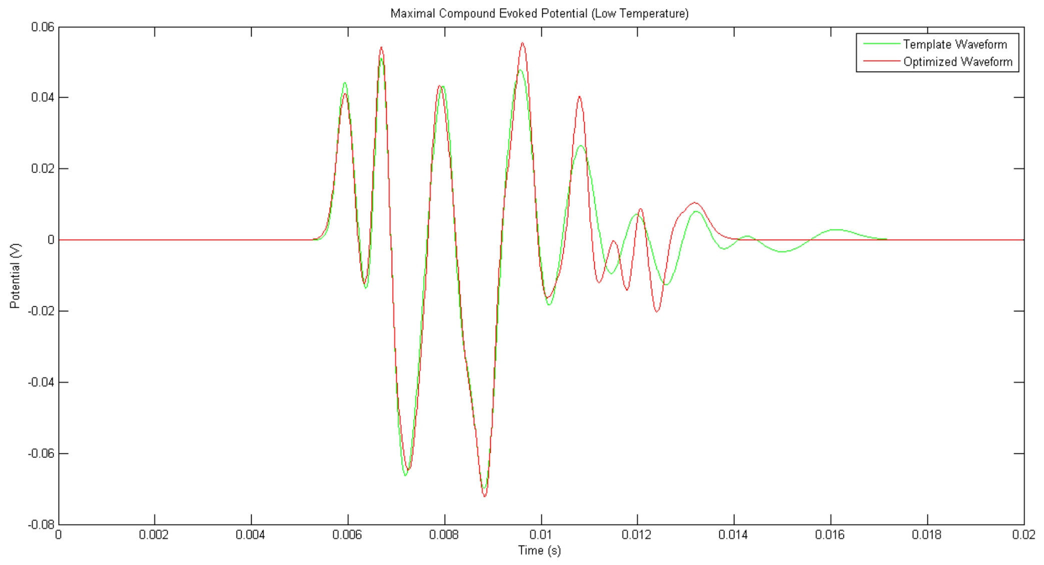


Figure 14 a) Maximum CEPs at High Temperature b) Maximum CEPs at Medium Temperature c) Maximum CEPs at Low Temperature

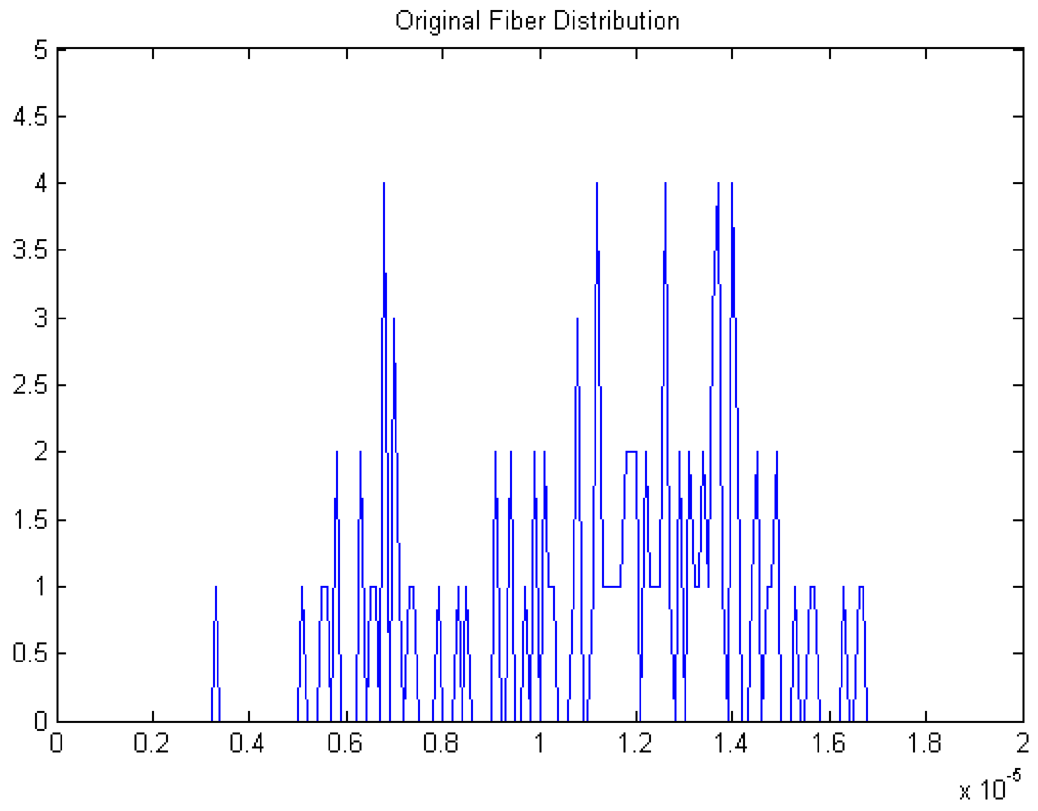
At the early stages of optimization when the temperature is high, the estimated compound evoked potential is a poor fit to the template waveform. It becomes apparent that as the temperature decreases, the maximum compound evoked potential from the optimized fiber diameter set improves the fit to the template maximum compound evoked potential.

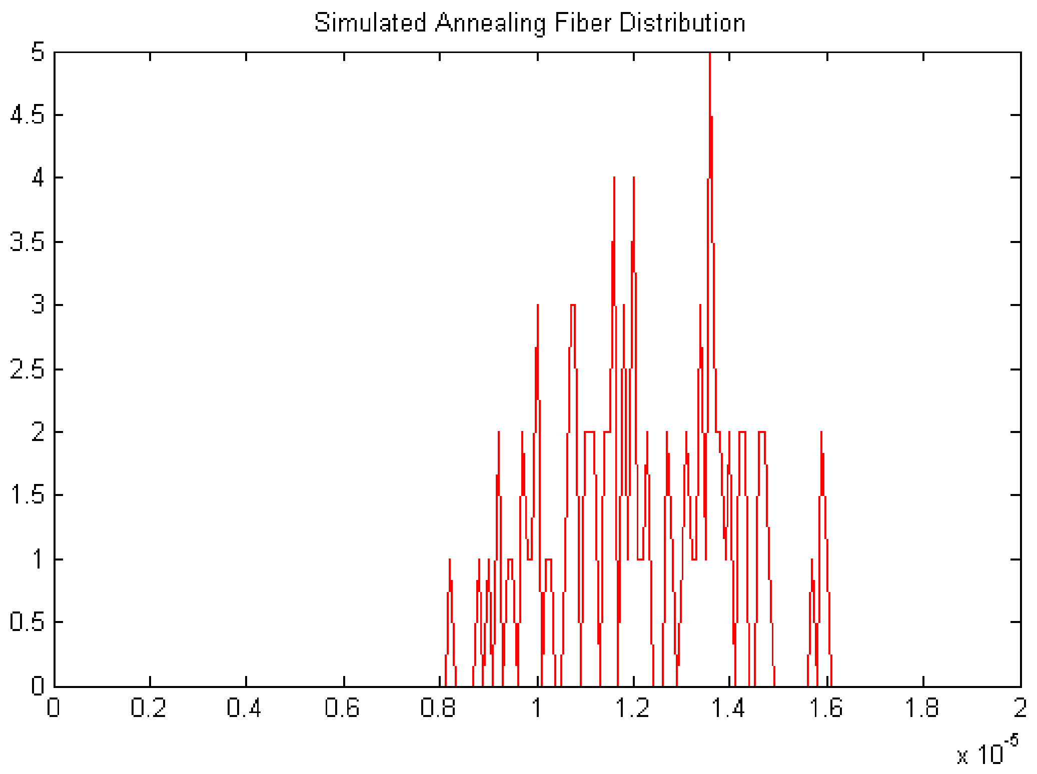
2.5 Using Chi-Squared to Test the Fit of Distributions

There is another way of quantifying the performance of simulated annealing at improving the fiber diameter distribution estimation after group delay. The chi-squared test (χ^2) is used in statistics to determine if there is an association between two sets of a particular category of data. It compares observed results with those that are expected and assigns a value between 0 and 1 to denote how well the datasets “fit” with one another. In the case of this study, our observed results are the fiber diameter distribution estimated by group delay and optimized by simulated annealing while our expected results are the fiber diameter distribution generated by the probability density function which we call the template distribution. A chi-square value of 0 means there is no association between the sets while a value of 1 means the sets are a match. In this study, our aim is to increase the chi-squared value using the simulated annealing approach. One of the final steps of the simulation is to compare the group delay estimated fiber size distribution to the template distribution and calculate the chi-squared value. Similarly, after the simulated annealing algorithm finishes, the optimized fiber size distribution is also compared to the template in order to determine if the algorithm improved the initial group delay estimation.

This calculation is explained in greater detail in the next section.

To illustrate the comparison of two different fiber diameter distributions, the following figures show an example template distribution, an estimated distribution optimized by simulated annealing and a comparison of the two.




Simulated Annealing Fiber Distribution
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
x 10^{-5}

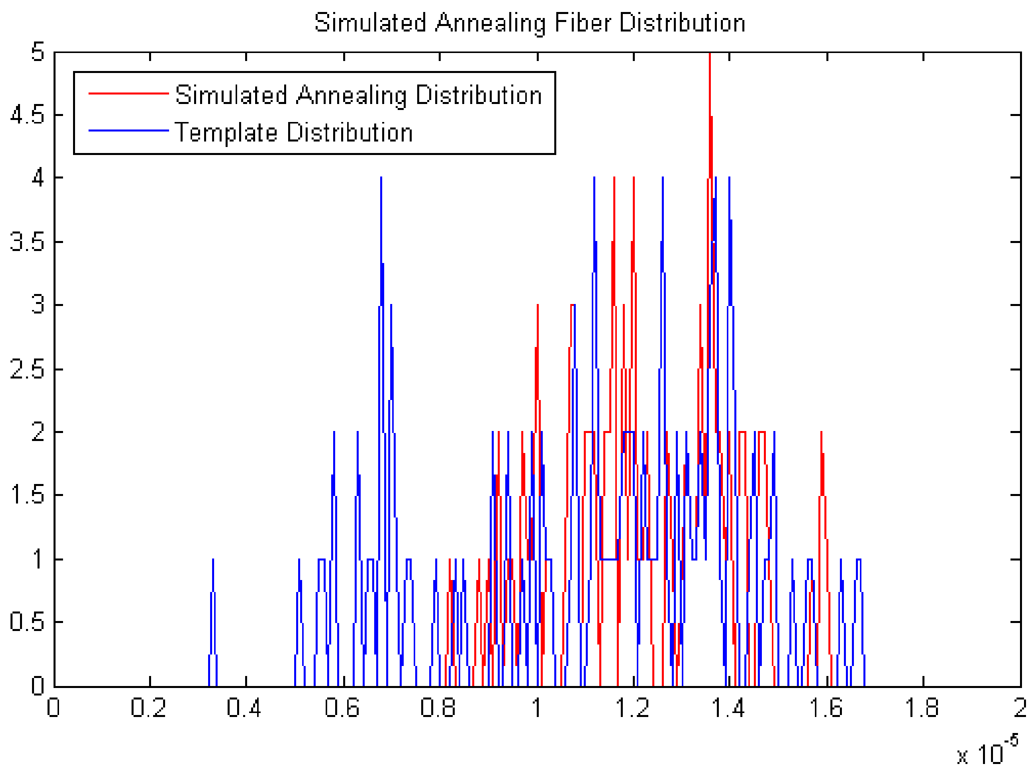


Figure 15 Example Fiber Distributions Compared using the Chi-Squared Test

2.5.1 Measuring the Chi-Squared Difference After Simulated Annealing

For this project, a value hereafter referred to as the chi-squared difference will be used to quantify the performance of simulated annealing towards optimizing the size distribution of fiber diameters to the template distribution. Let GD_{chi} be the chi-squared value of the group delay estimated fiber distribution compared to the original template distribution calculated in (2). Let SA_{chi} be the chi-squared value of the simulated annealing optimized distribution of fiber diameters averaged over the number of trials in the simulation compared to the original template distribution calculated in (2). The chi-squared difference is then calculated from the following equation.

$$Chi\ Squared\ Difference = SA_{chi} - GD_{chi} \tag{12}$$

From equation 12, we can see that a positive chi-squared difference indicates that the simulated annealing optimization improved upon the original group delay estimated fiber diameter distribution when compared to the template distribution ($SA_{chi} > GD_{chi}$). Furthermore, a negative chi-squared value indicates that the optimized fiber diameter distribution became a worse fit than the group delay estimated fit when compared to the template distribution ($SA_{chi} < GD_{chi}$). In other words, the simulated annealing optimization made the original estimation worse.

2.6 Cooling Schedule Simulations

The task of determining how manipulating the parameters of the cooling schedule affects the performance of optimizing the group delay estimate was accomplished by initially isolating several key parameters from the cooling schedule and varying their values around a pre-determined range. By only changing one parameter's value in a particular simulation, its impact can be directly assessed based on the output of the annealing algorithm. For some test cases, after the range of values for one parameter was exhausted, the procedure was repeated after changing a second parameter. The details of these types of test cases are explained in subsequent sections.

More sophisticated variables such as the cooling ratio require more in-depth test coverage. Not only was the default exponential decay explored, but other cooling options that have been described in previous studies were also applied to our group delay scenario. Logarithmic, linear, adaptive and a unique linear-exponential hybrid cooling methods were individually simulated and then compared to one another to determine which cooling strategy performs the best optimization.

Quantification of the simulated annealing algorithm was achieved by running each type of simulation a pre-determined number of times in order to calculate an acceptable average final error and chi-squared value. Because simulated annealing belongs to the class of Monte Carlo optimization algorithms (that is, improvements are made by making random changes to the dataset), the result from one trial may differ significantly from the next even in the exact same test environment. Therefore, a significant number of trials need to be simulated in order to obtain a proper average. The fewer number of trials that are simulated will mean a higher probability of observing random data and inconclusive results.

Another important characteristic that must be considered when assessing the performance of a particular annealing schedule is the computational effort required to achieve that result. A balance must be established between the optimization of the final error and chi-squared values versus the amount of time or computing power it takes to get to that point. For example, a simulation that runs for a week and finds an extremely low value of the cost function is less valuable to our study than a simulation that runs for half an hour and reaches a fairly acceptable approximation of the minimum value of the cost function. The best solution will be one that produces near-optimal results but more importantly can be implemented in a practical, real-world setting. The metric used for this analysis will be the number of temperature steps explored before reaching the lower temperature bound (stopping temperature). Time is not used to measure computational effort because the simulations in this study were occasionally run on different computing environments with unequal processor speeds which impact the speed of execution.

In order to maintain a consistent simulation environment, the generation of the template fiber distribution was only performed at the beginning of each type of simulation. Likewise, the group delay technique was executed only one time in order to produce the initial diameter estimation. From there, the simulated annealing optimization was performed on the same group delay estimation so that each simulated annealing trial was receiving the same input to the algorithm. Due to the already random nature of the algorithm, this technique provided some degree of uniformity in order to obtain an appropriate average from the results.

The cooling schedule variable declarations are located in the beginning of the driver function Test_Annealing.m. The original default values which produced the results published in [14] are shown below.

```
temp_start = 10;          %Starting Temperature
temp_factor = 0.9;        %Cooling Ratio (Exponential Decay)
temp_bound = 1.0E-5;      %Stopping Temperature
max_step = 1000;          %Number of Trials per Temperature Step
```

For the other cooling strategies such as the logarithmic or linear decay method, the `temp_factor` variable is obsolete and ignored. The implementation of these techniques will be described in the appropriate sections.

2.6.1 Starting Temperature (`temp_start`)

It is generally agreed upon from previous simulated annealing studies that the starting temperature value should be sufficiently high to allow virtually all transitions to be accepted at the beginning of the algorithm regardless of whether or not they increase the error (cost) function

[27]. In other words, the Boltzmann distribution acceptance criterion should be approximately equal to 1 at the starting temperature [15].

$$1 \approx \exp\left[-\frac{|\Delta - \Delta_{NEW}|}{T}\right] \tag{13}$$

This is an important period during the simulated annealing process because it allows the algorithm to explore the entire solution space before settling into a region where nominally the global minimum of the function is located.

In order to solve for the starting temperature T in the above equation, an expected value for the change in the error function (Δ - Δ_{NEW}) must be determined. This value was calculated empirically by running the simulated annealing optimization enough times to obtain a confident average value for the change in error after each individual fiber diameter change. After 1000 changes to the fiber diameter distribution vector, this value was approximated at -0.0087.

Substituting the empirically determined average change in error function into the equation, the acceptance criterion can be plotted as a function of the starting temperature to determine approximately what temperature produces an acceptance rate of nearly 100 percent. This graph is shown below.

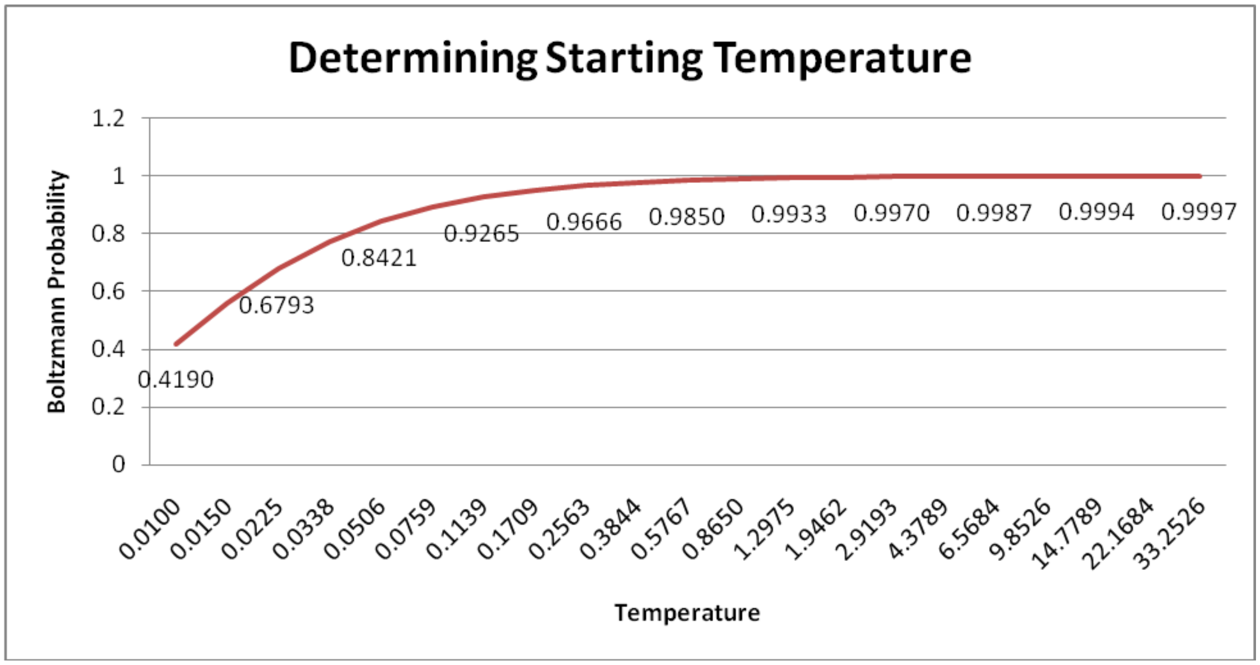


Figure 16 Plot of Acceptance Criterion as a function of Starting Temperature

From Figure 16, we can expect to find the ideal starting temperature to be approximately in the range of 14-35 although even around a temperature of about 1, the acceptance rate will still be over 99 percent. Although this is simply an approximation and actual results may vary, the difference between starting at a value of 35 versus 1 is significant especially in an exponential decay because of the additional computation required to reduce the temperature from 35 to 1.

To test the hypothesized ideal starting temperature, the simulated annealing optimization was executed 25 times at the following starting temperatures: 0.1, 1.0, 10 and 50. For the other cooling schedule parameters, the following values were used: 1E-5 for stopping temperature and 500 transitions at each temperature step. The default exponential decay strategy was used to cool the system. The final error and chi-squared values were averaged for 25 trial runs and compared with results from the other starting temperatures. Lastly, the procedure was repeated after changing the exponential cooling rate from 0.9 to 0.5 and finally 0.1.

The lowest possible starting temperature that does not drastically reduce the performance of the optimization algorithm is said to be the best value so as to minimize redundant computations at the beginning of the optimization which provide no added benefit to the annealing process.

2.6.2 Stopping Temperature (`temp_bound`)

In contrast to the starting temperature, the stopping temperature must be sufficiently low for an extended period of time in order for the algorithm to make its way towards the local minimum and ideally the global minimum of the cost function. Recall that at decreasing temperatures, the Boltzmann probability distribution will gradually make it more difficult to accept changes which increase the cost function until the temperature is low enough where virtually no changes are accepted. For this reason, the algorithm needs enough time to only transition “downhill” to finish at what is expected to be the lowest possible value of the cost function.

Using the same technique as with the starting temperature, we can use our previously calculated average value for the change in error function after a new fiber diameter is generated to plot the acceptance criterion as a function of temperature to determine at what point it approaches approximately zero percent.

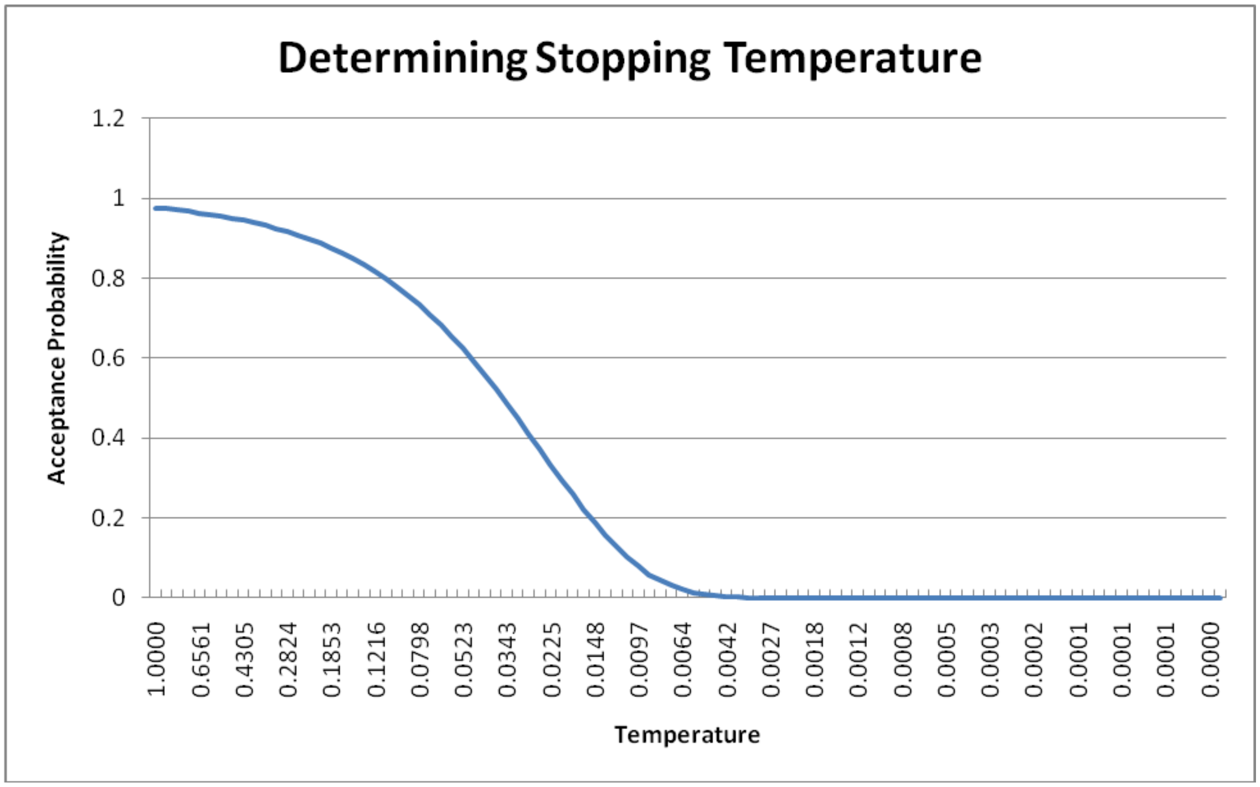


Figure 17 Determining When Boltzmann Acceptance Criterion Reaches Zero

From the graph, we expect the acceptance criterion to be close to zero at a temperature of about 0.004. To verify, plugging this temperature into the Boltzmann equation yields a value of 0.000207 (or .0207% acceptance probability). Again it is important re-iterate that the algorithm needs sufficient time at this "no-acceptance" period to only make downhill transitions in order to nominally find the global minimum of the cost function. Therefore, we hypothesize that an appropriate stopping temperature must exceed 4E-3. To test this assumption, the stopping temperature was varied between 1E-1 to 1E-9 (in steps of 1E2) and the simulation was run 25 times for each value. The procedure was then repeated for exponential decay ratios of 0.5 and 0.1 and compared to the default value of 0.9. The same statistical analysis was performed on the resulting output and a plot of the average error value and chi-squared value was computed for comparison of the varying final temperature values.

2.6.3 Number of Trials per Temperature Step (`max_step`)

Possibly the second most important factor that determines the computational effort required to run a particular cooling schedule for simulated annealing is the number of transitions at one temperature step. This value sets the limit for the number of fiber diameter changes that will be made before decrementing the temperature. Other implementations of this variable in the literature only count the number of “accepted” transitions (those which reduce the error function). In our study, this variable includes both uphill and downhill changes. According to the theory behind the algorithm, you must allow enough transitions to take place in order to reach a state of equilibrium before decreasing the temperature. By default, this value is set to 1000. The list of values that were tested for this parameter includes 10, 100, 500, 1000, 2500 and 5000. For each value, 25 simulations were run and the average final error and chi-squared values were recorded and compared to one another and plotted. These trials were performed at exponential decay factors of 0.5 and 0.1.

To broaden the scope of this test scenario, the above procedure was repeated for a second group delay estimated dataset. This study allows us to see what effect (if any) a good initial group delay estimate has on the annealing process versus a poorer estimated dataset. We expect to find that a worse estimate of the fiber diameter distribution will yield a larger chi-squared improvement after the simulated annealing optimization is executed.

2.6.4 Cooling Ratio (`temp_factor`)

2.6.4.1 Exponential Decay

Arguably the most important variable in the entire simulated annealing algorithm is the method for which the temperature parameter is reduced. It determines how soon we reach our final answer and how much computational effort is required to get there. There are many variations for implementing this parameter but the default technique implemented by Szlavik et al. is an exponentially decaying cooling schedule [14]. The `temp_factor` variable in MATLAB represents the ratio (between 0 and 1) for which the temperature variable will be reduced in its next step. For example, if the current temperature in the algorithm is 10 and the `temp_factor` variable is set to 0.9, the next step of the annealing process will reduce the temperature to 9, then 8.1, and so forth. The higher this variable is set, the greater the number of temperatures that will be explored and as a result, the longer the simulation will run.

Previous implementations of the simulated annealing algorithm for solving optimization problems have also determined that the best results are found when the cooling ratio is between 0.8 and 0.99 [15]. In this study, the cooling ratio was varied from the default value of 0.9 down to 0.1 in intervals of 0.2. The other cooling schedule parameters were set as followed: starting temperature set to 10, stopping temperature set to 1E-5 and number of trials per temperature step set to 500. The simulations were run on the same group delay estimated dataset 25 times and an average final error and chi-squared value was found for each value of the cooling ratio. The five different ratios were plotted together to compare the results.

2.6.4.2 Logarithmic Decay

Another method for implementing the cooling strategy is to use a logarithmic decay. The formula used to model this type of schedule was described by Geman et al. and is as follows:

$$T(t) = \frac{c}{log(t+d)} \tag{14}$$

where *d* is usually set to one and *c* is greater than or equal to the largest energy barrier in the problem [28]. Studies have proven this method effective in finding the global minimum of a cost function but in the limit of infinite time [20].

We encounter an interesting situation in the design of this cooling strategy. Because of the asymptotically slow temperature decrease, a logarithmic approach in our current simulation environment is impractical because the stopping temperature bound (default 1E-5) will only be reached in an extraordinarily large amount of time. Running this simulation in MATLAB will continue indefinitely until the user forces a stoppage. To counteract this issue, a new variable is created in the simulation that acts as a time index which can limit the number of temperature steps in the simulation so that we only simulate the active portion of the decay and not the period where the algorithm is essentially “frozen.” This variable imitates the value of the time vector t in the above equation. We restrict our simulation to compute 150 steps of this logarithmic schedule at different values of the variable *c* to investigate its effectiveness as a cooling scheme. The MATLAB code below shows this implementation.

```
%Logarithmic Decay
index = index + 1;
if (index < 151)
    temp = c/(log(index) + 1);
else
    temp = 0;
end
% End Log Decay
```

Figure 18 below illustrates the range of *c* values tested and their associated logarithmic decay.

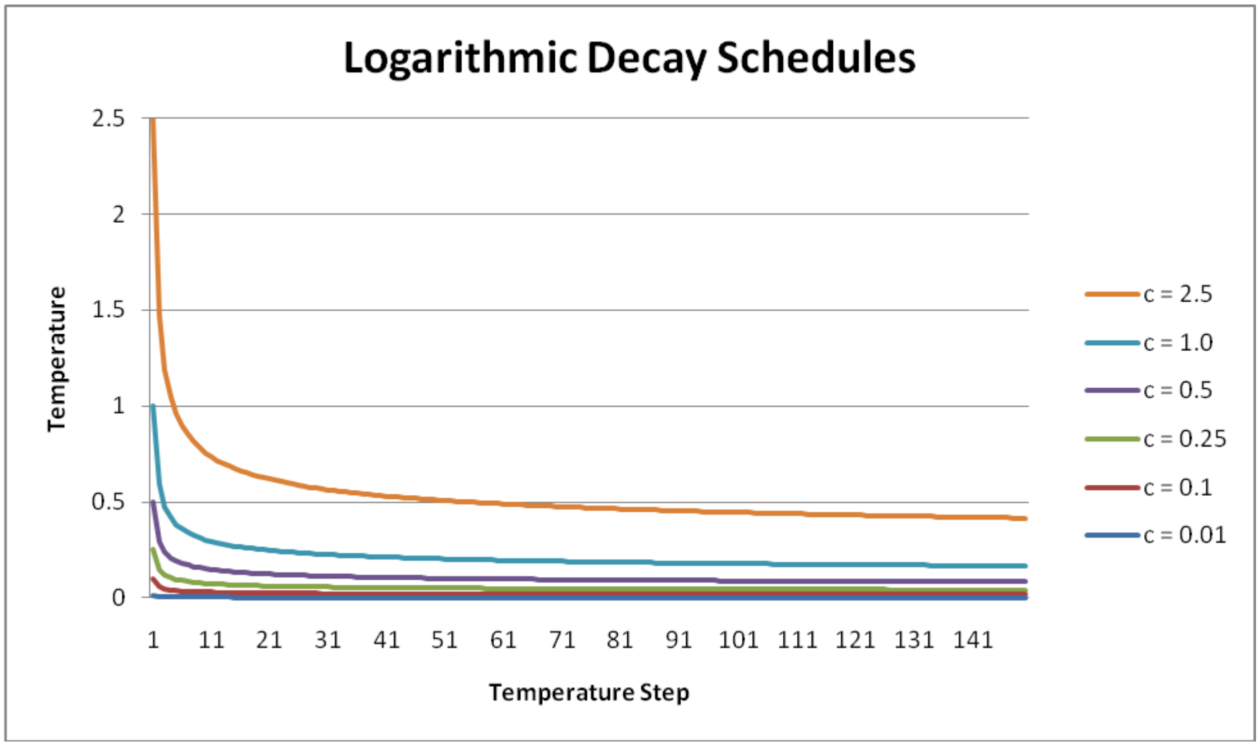


Figure 18 Various Log Decay Schedules Simulated

Note that the value of *c* chosen also sets the starting temperature for the simulation because at step t = 1, the function T(1) evaluates to *c*. An associated table is provided which describes the initial and final values of each log function plotted above.

Table 3 Summary of Tested Log Functions

c =	0.001	0.01	0.1	0.25	0.5	1	2.5
t = 1	0.001	0.01	0.1	0.25	0.5	1	2.5
t = 2	0.000591	0.0059	0.0591	0.1477	0.2953	0.5906	1.4765
…	…	…	…	…	…	…	…
t = 149	0.000167	0.0017	0.0167	0.0416	0.0833	0.1666	0.4164
t = 150	0.000166	0.0017	0.0166	0.0416	0.0832	0.1664	0.4159

Note the small change in temperature between steps t = 149 and t = 150 due to the asymptotic behavior of the log function.

For each of the six logarithmic cooling schedules tested, the simulation was executed 25 times using the same group delay estimated set as in the exponential decay above. All other parameters including the number of moves per temperature step (500) were also kept the same.

2.6.4.3 Linear Decay

A third commonly implemented strategy for reducing the temperature in simulated annealing is the linear approach. In contrast to exponential and logarithmic decay, a linear cooling schedule reduces the temperature by the same amount throughout the annealing process. In MATLAB, the previous exponential decay code used to decrement the temperature was modified to the following:

```
temp = temp - linear_factor;
```

where `linear_factor` is a constant which describes the amount the temperature is reduced after each iteration of the annealing. For this study, the test set was divided into two scenarios. The first scenario compares three linear cooling methods with the same initial temperature but different decrement values. The second scenario compares four schedules which start at different starting temperatures and have different decrement values but all run for 200 temperature steps before reaching the lower limit temperature boundary (temperature actually reaches zero). These two scenarios are summarized in the table below and are illustrated in the following figures.

Table 4 Summary of Linear Cooling Schedules

	Scenario 1			**Scenario 2**			
Schedule Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Starting Temperature	10	10	10	10	1	0.1	0.01
Linear Decrement Factor	0.25	0.1	0.05	0.05	0.005	0.0005	0.00005
Number of Temperature Steps	40	100	200	200	200	200	200
Final Temperature	0	0	0	0	0	0	0

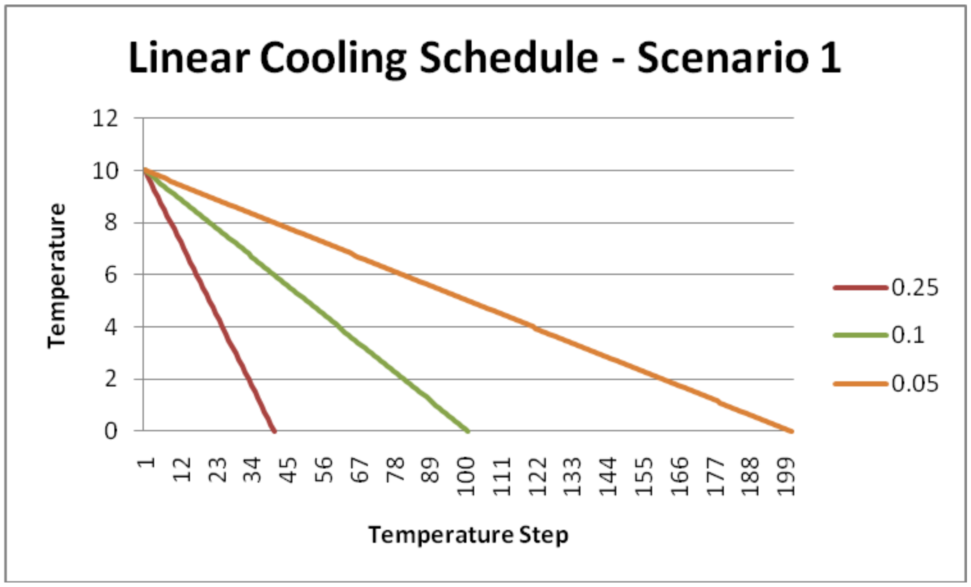


Figure 19 Scenario 1 Linear Cooling Schedules with Different Decrement Factors

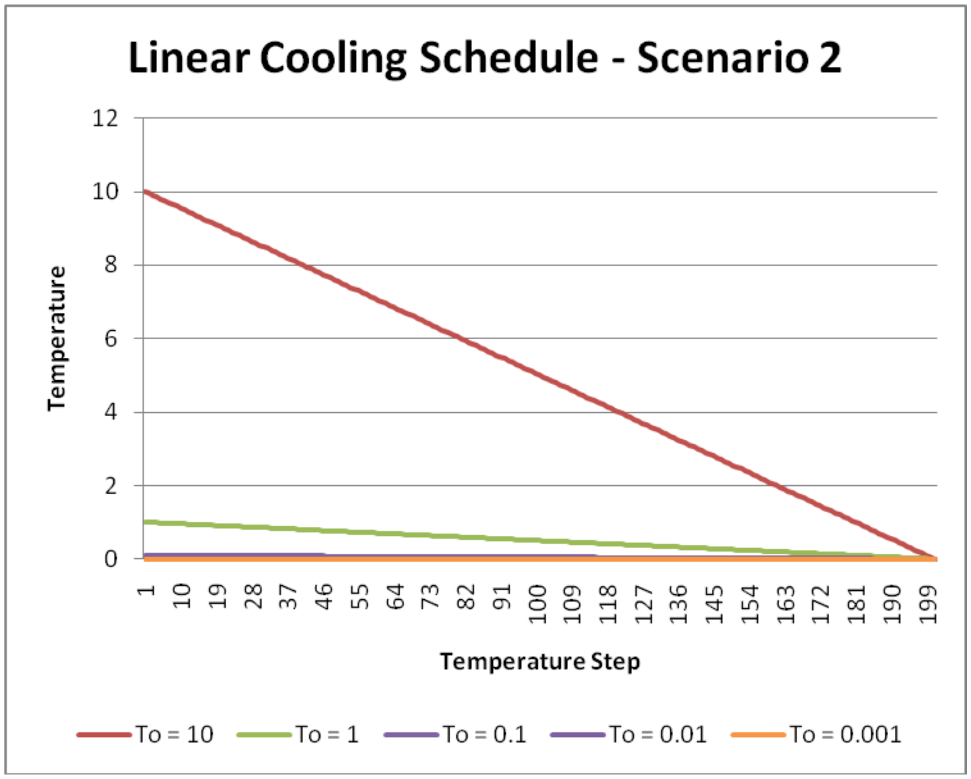


Figure 20 Scenario 2 Linear Cooling Schedules with Different Starting Temperatures

Each linear schedule above was simulated 25 times using the same group delay estimated dataset as in the exponential and logarithmic techniques. The number of fiber diameter changes at each temperature step was set to 500 and as before, the average final error and chi-squared value was computed for each schedule and compared to one another.

2.6.4.4 Linear-Exponential (LinEx) Hybrid Schedule

In an effort to minimize the computational effort while maintaining a thorough annealing approach, a linear-exponential technique is proposed which combines the sharp temperature decrease of a linear schedule with the broad temperature range of an exponential decay. The schedule starts at temperature 10 and finishes at 1E-5 but cools more than twice as quickly as the default exponential schedule (given a 0.9 cooling ratio). The equations below and Figure 21 illustrate this hybrid schedule.

$$\begin{array}{cc} temp = temp - 1, & 1 < temp \\ temp = temp - 0.1, & 0.1 < temp \leq 1 \\ temp = temp - 0.01, & 0.01 < temp \leq 0.1 \\ temp = temp - 0.001, & 0.001 < temp \leq 0.01 \\ temp = temp - 0.0001, & 0.0001 < temp \leq 0.001 \\ temp = temp - 0.00001, & temp <= 0.0001 \end{array} \tag{15}$$

This hybrid approach scales the linear decrement factor at specific ranges throughout the annealing process. When the temperature is between 10 and 1, the decrement factor (as in the linear model) is 1. Between temperatures of 0.9 and 0.1, the decrement factor is scaled by a factor of 10 to 0.1. This process continues until the default stopping temperature of 1E-5 is reached after 55 temperature steps.

Using this linear-exponential schedule, the simulated annealing optimization is executed 25 times while making 500 fiber diameter transitions at each temperature step. The average final error and chi-square values will be compared to all other cooling strategies to determine if a reduction in computation effort does not drastically affect the performance of the optimization.

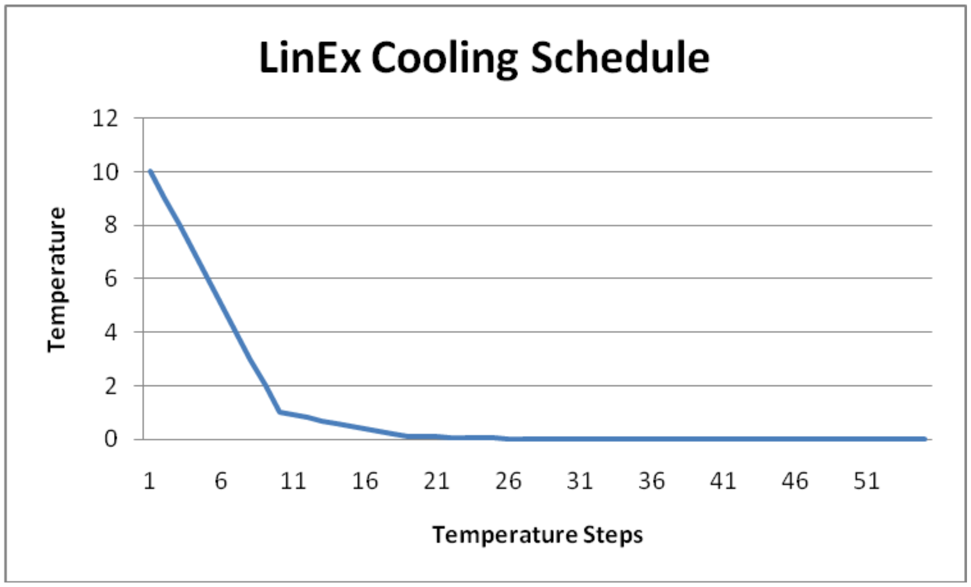


Figure 21 Linear-Exponential Decay Schedule

2.6.4.5 Adaptive Cooling Schedule

Expanding on the relatively simple linear-exponential implementation, an effort was made to identify critical temperatures similar to that in [24] where simulated annealing seems to make the largest improvements in the final error computation. The algorithm uses an exponential decay model but with a slight twist. The cooling ratio value increases throughout as the annealing progresses. To accomplish this adaptive feature, the range of temperatures in the schedule are broken down into three specific categories and assigned an associated cooling ratio. These ranges are to be determined by inspection after running a standard exponential decay schedule as described in section 2.6.4.1.

Table 5 Summary of Adaptive Cooling Schedule

	Exponential Cooling Ratio
High Temperatures	0.1
Medium Temperatures	0.5
Low Temperatures	0.9

This adaptive schedule is simulated 25 times with the same group delay estimated dataset. Results were then compared to all previously discussed schedules.

3 Results

3.1 Starting Temperature

Recall from earlier that we hypothesized that an ideal starting temperature would be at a point where the system behaves as if in a “molten” state and also not set too high in order to reduce any unnecessary computations which can increase execution time. The “molten” state is described as the period where all or most fiber diameter changes are accepted and the algorithm is free to explore a broad solution space. The following figures compare results of the final error averages and chi-squared differences for each starting temperature tested. The number of transitions at each temperature step was set to 500 and the stopping temperature was equal to 1E-5 for all simulations in this test case. The final error graph is a measure of the difference between the template maximum compound evoked potential and the optimized maximum compound evoked potential as calculated in section 2.3. Chi-squared values are computed as the difference between the simulated annealing chi-squared value and the group delay estimated chi-squared value (both of which are tested against the original template distribution). The initial group delay estimated dataset yielded a chi-squared value of 0.2854.

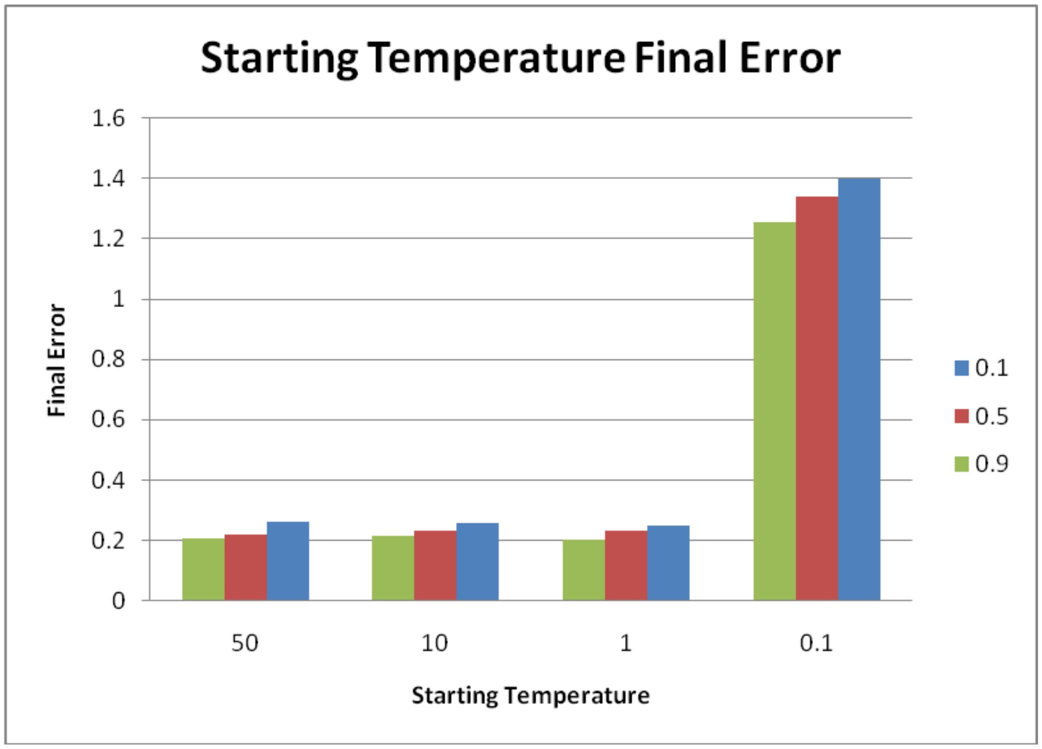


Figure 22 Final Error Computation for Different Starting Temperatures. Simulation Tested at Exponential Cooling Factors of 0.1, 0.5 and 0.9.

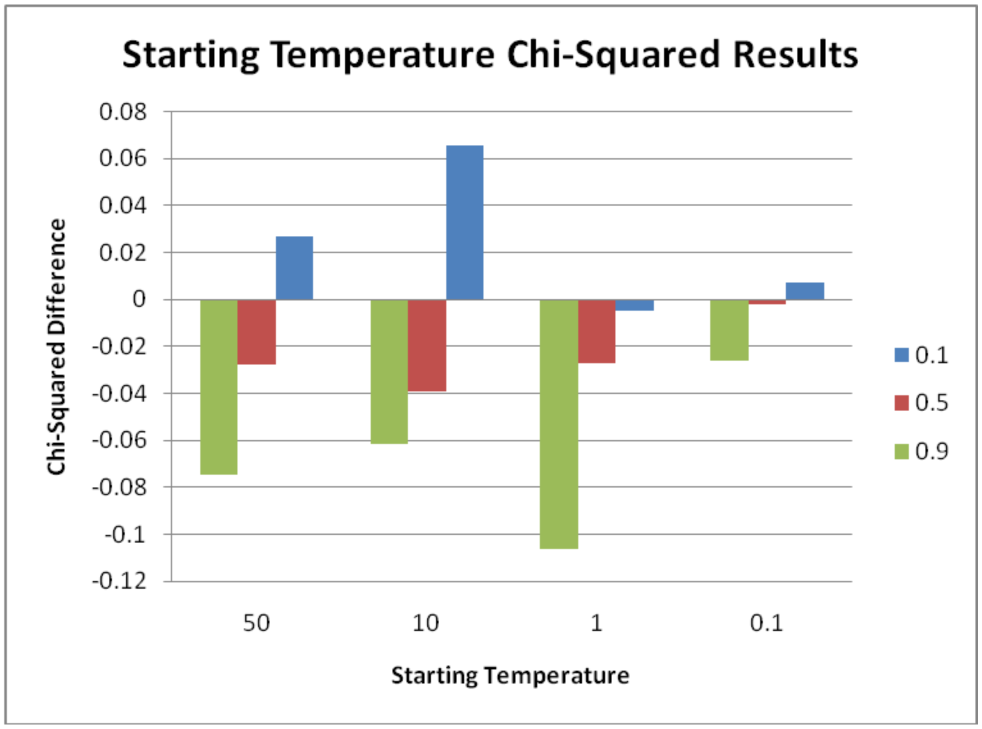


Figure 23 Chi-Squared Difference for Different Starting Temperatures. Simulation Tested at Exponential Cooling Factors of 0.1, 0.5 and 0.9. Initial Chi-Squared value from group delay equal to 0.2854.Recall a positive chi-squared difference indicates an improved fit after simulated annealing.

3.2 Stopping Temperature

The following figures show the averaged results of the 25 simulations executed for each of the different stopping temperatures. Recall from our earlier analysis that we expect an ideal stopping temperature to be lower than 4E-3 because that is the estimated point where the acceptance criterion rejects all fiber diameter changes which would increase the cost function. The final error graph is a measure of the difference between the template maximum compound evoked potential and the optimized maximum compound evoked potential using the two-norm equation as shown in section 2.3 and equation (4). The chi-squared difference graph is a measure of the difference between the group delay estimated fiber diameter distribution (when compared to the template distribution) and the simulated annealing optimized fiber diameter distribution (also when compared to the template distribution). Recall that a negative value indicates that the annealing process reduced the chi-squared value from the original group delay estimation. For each scenario in this test case, the starting temperature was set to 10 and the number of transitions per temperature step was equal to 500. The input to the simulated annealing algorithm for this test case was a group delay estimated fiber diameter dataset which yielded an initial chi-squared result of 0.4708 therefore Figure 25 compares the difference from this value after annealing.

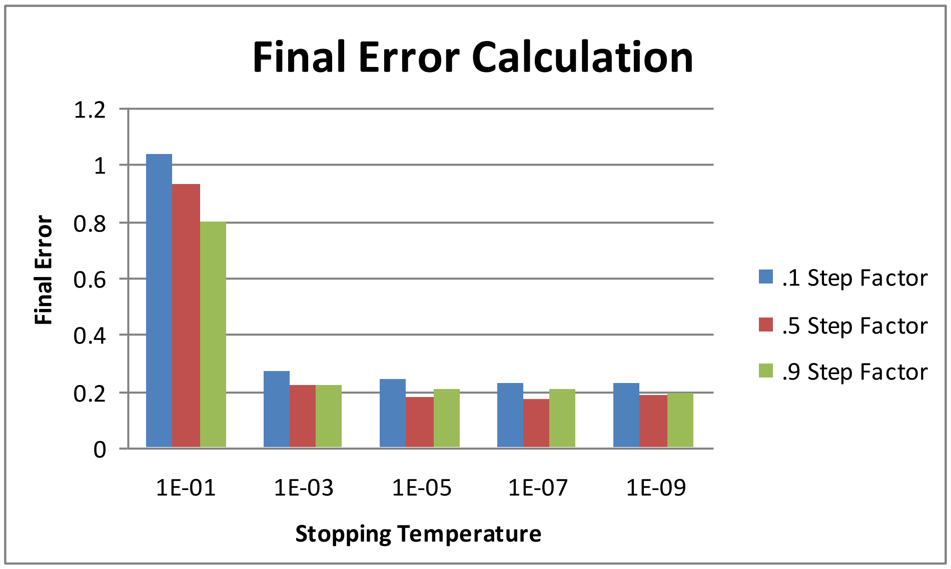


Figure 24 Final Error Calculations at Different Stopping Temperatures. Simulations repeated for Exponential Step Factors of 0.5 and 0.1.

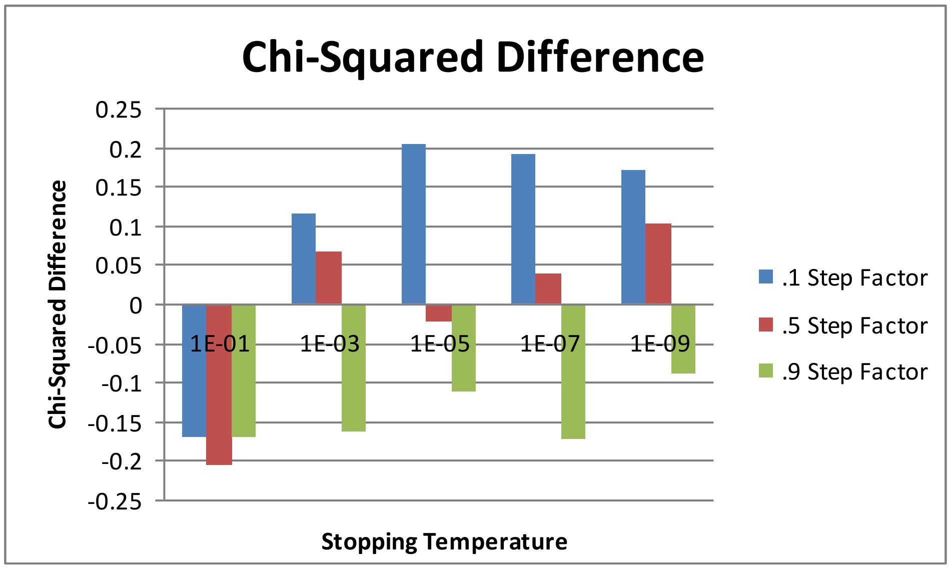


Figure 25 Chi-Squared Calculation for Different Stopping Temperatures. Simulation was Repeated for Step Factors of 0.5 and 0.1.

3.3 Number of Transitions per Temperature Step

The following figures compare the average final error values and chi-squared differences for the 25 trials executed at each number of transitions per temperature step. Simulations were run using an exponentially decaying cooling schedule with cooling ratios of 0.1 and 0.5. Parameters such as the starting and stopping temperatures were kept at their default values (10 and 1E-5, respectively). Further, two different group delay estimated datasets were used in the simulated annealing algorithm. The first dataset yielded an initial chi-squared value of 0.3057 while the second dataset yielded a value of 0.1073. Both datasets are plotted together to determine if there is an effect on the optimization process given two different group delay estimated results.

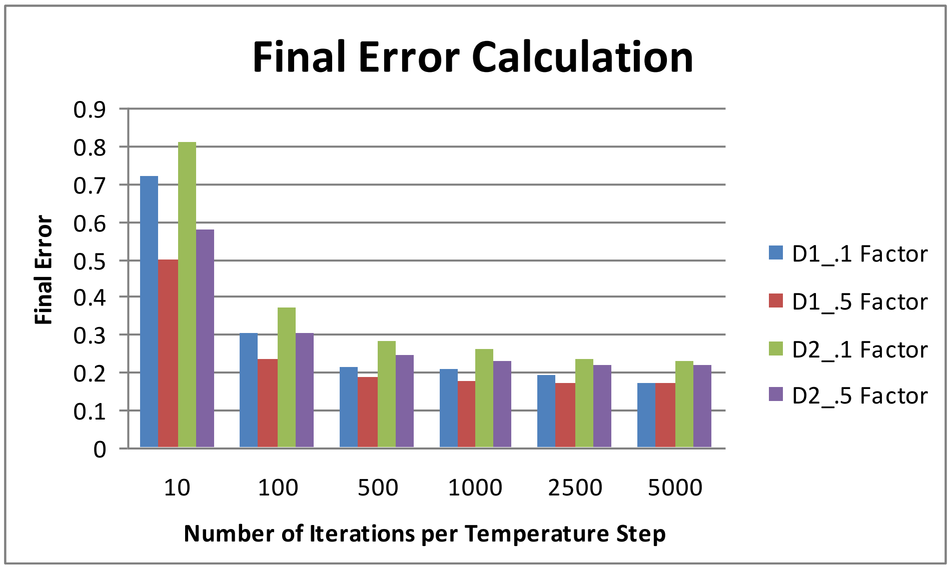


Figure 26 Final Error Calculations for Two Different Datasets (D1 and D2) at Various Numbers of Transitions per Temperature Step. Simulation was run at Exponential Cooling Ratios of 0.1 and 0.5.

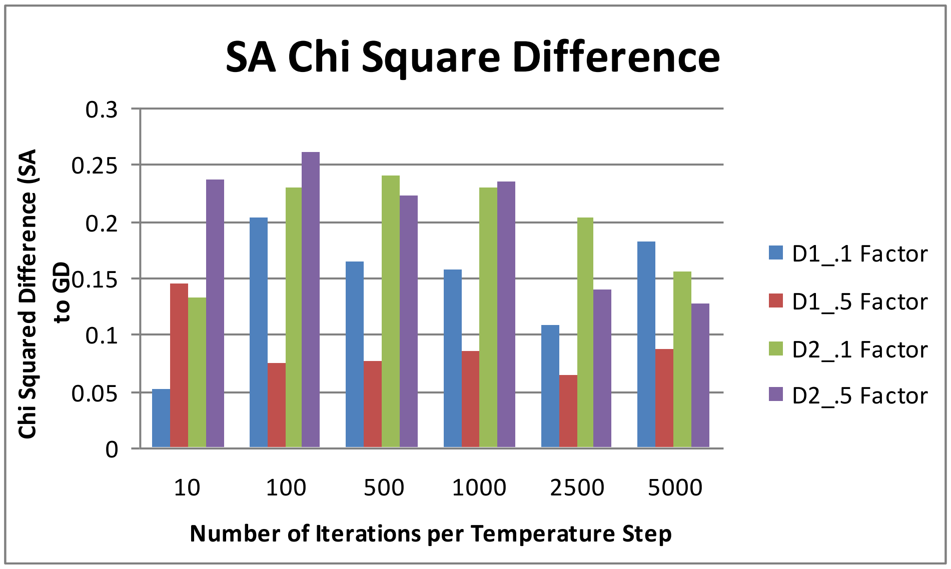


Figure 27 Chi-Squared Difference Versus Number of Transitions per Temperature Step. Simulation Executed using Exponential Cooling Ratios of 0.1 and 0.5.

3.4 Cooling Strategy

The following sections compare the various cooling strategies that can be implemented in simulated annealing and their respective results for the optimization of both the maximum compound evoked potential (final error) and the fiber diameter distribution (chi-squared difference).

3.4.1 Exponential Cooling Schedule

The final error and chi-squared results averaged for 25 trials are depicted in Figures 27 and 28, respectively. For the chi-squared difference calculation, the original group delay estimated dataset yielded a result of 0.2854. The following values were used for the controlled parameters of the schedule: starting temperature equal to 10, final temperature equal to 1E-5 and number of transitions per temperature step equal to 500.

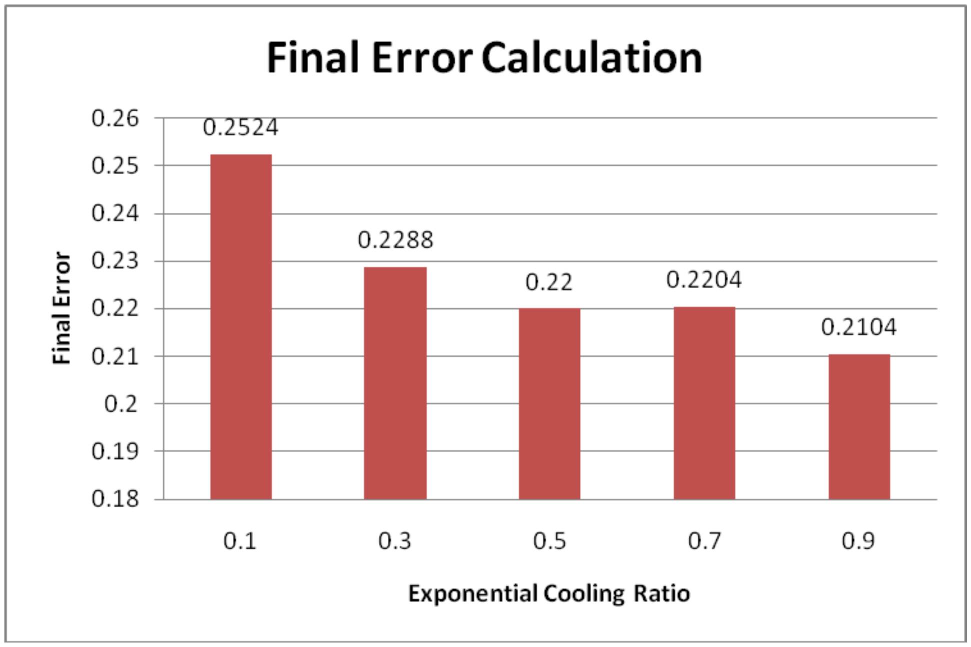


Figure 28 Final Error Calculations for Exponential Cooling Schedule

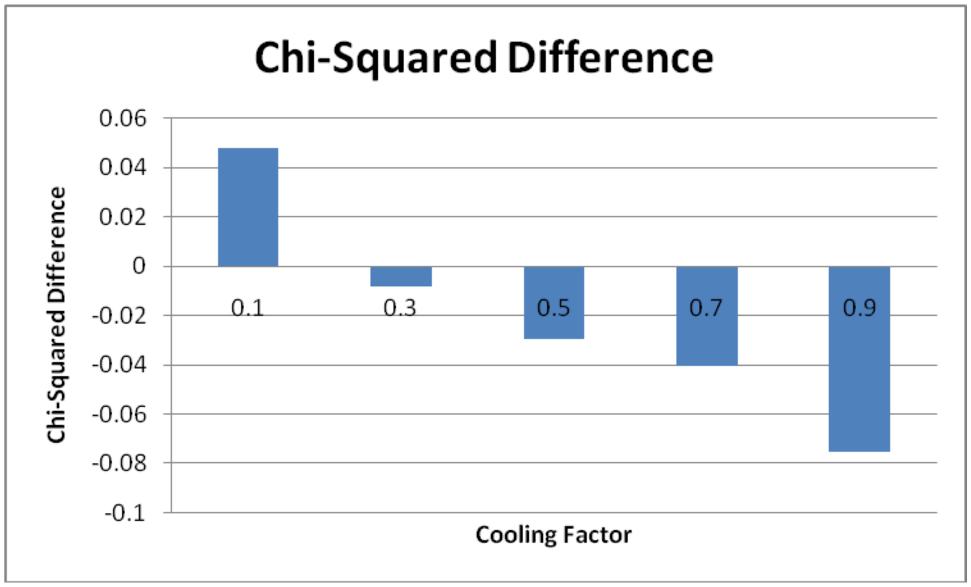


Figure 29 Chi-Squared Calculation for Exponential Cooling Schedule. Original Group Delay Chi-Squared Value was 0.2854.

3.4.2 Logarithmic Cooling Schedule

The following figures show the average results of the 25 simulation trials for each of the tested logarithmic schedules. The final error calculations in Figure 30 compare the estimated compound evoked potential with the template compound evoked potential using the two-norm calculation previously mentioned. The chi-squared results in Figure 31 represent the difference from the initial group delay estimated chi-squared result. For this particular simulation, the initial chi-squared value after the group delay estimation was 0.2854.

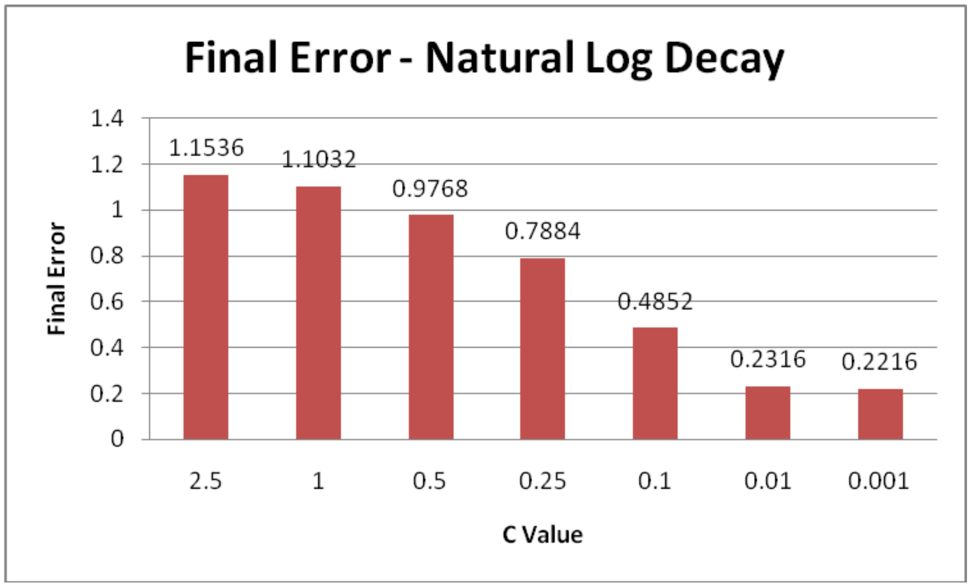


Figure 30 Final Error Calculation for Natural Logarithmic Cooling Schedule at Various C Values

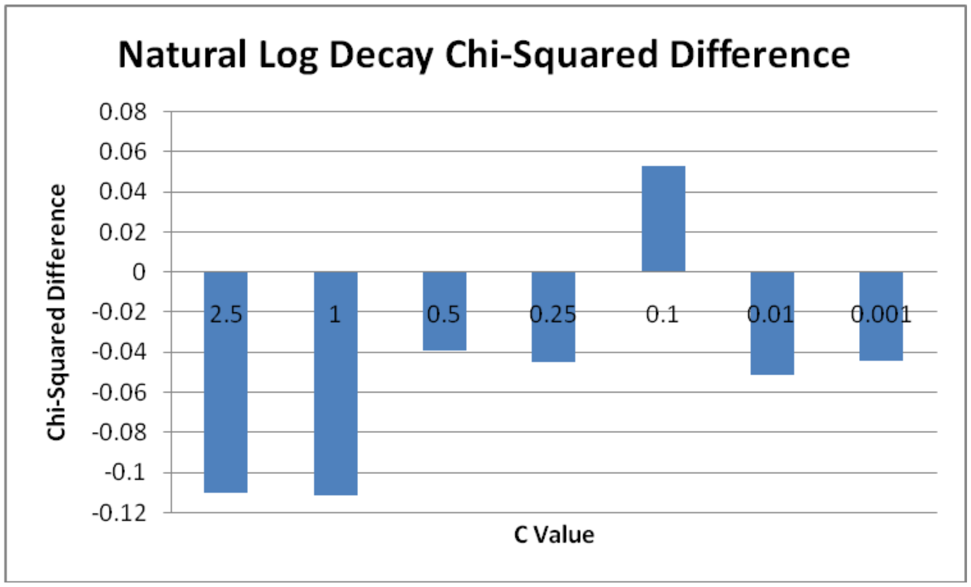


Figure 31 Chi-Squared Difference for Various C Values of the Natural Logarithmic Cooling Schedule

3.4.3 Linear Cooling Schedule

The two linear schedule scenarios discussed in section 2.6.4.3 and their respective final error and chi-squared results are presented in the next couple sections. Results are averaged from 25 trials for each schedule. Unless otherwise mentioned, starting temperatures, stopping temperature and the number of transitions per temperature step were fixed at 10, 1E-5 and 500, respectively.

3.4.3.1 Scenario 1

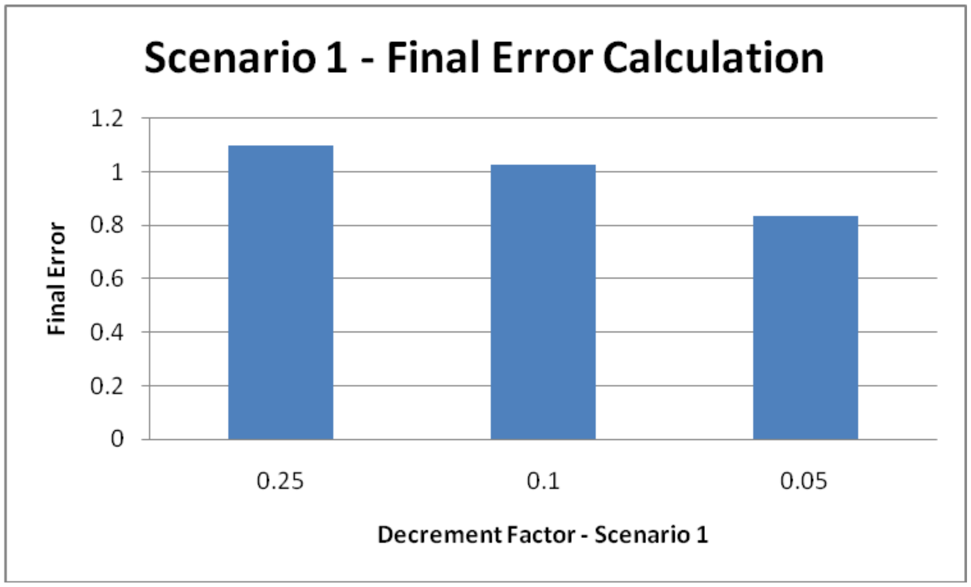


Figure 32 Final Error Calculation for Scenario 1 of Linear Decay Cooling Schedule. Starting Temperature for all Groups was 10.

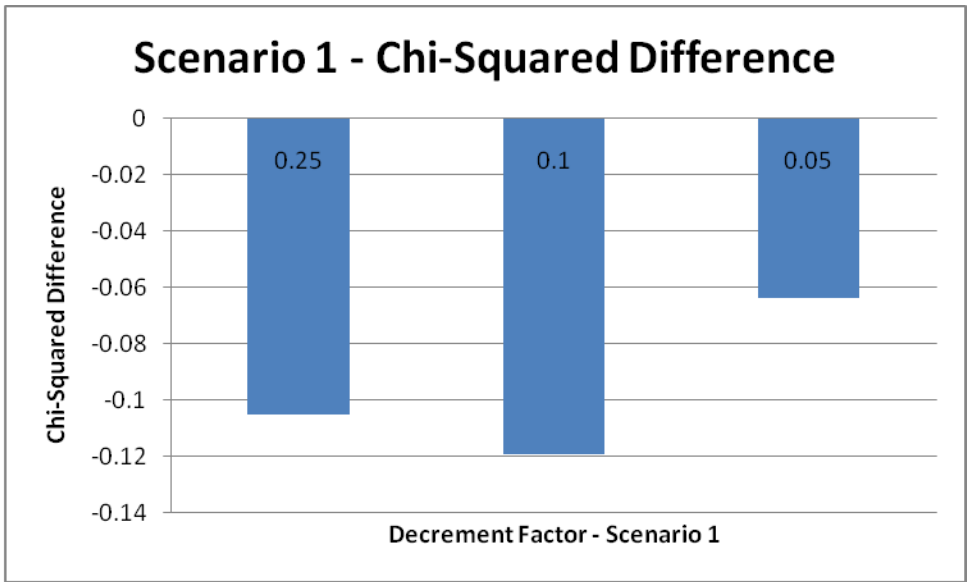


Figure 33 Chi-Squared Difference for Scenario 1 of Linear Decay Cooling Schedule. Starting Temperature for all Groups was 10. Original Group Delay Chi-Squared Value was 0.2854.

3.4.3.2 Scenario 2

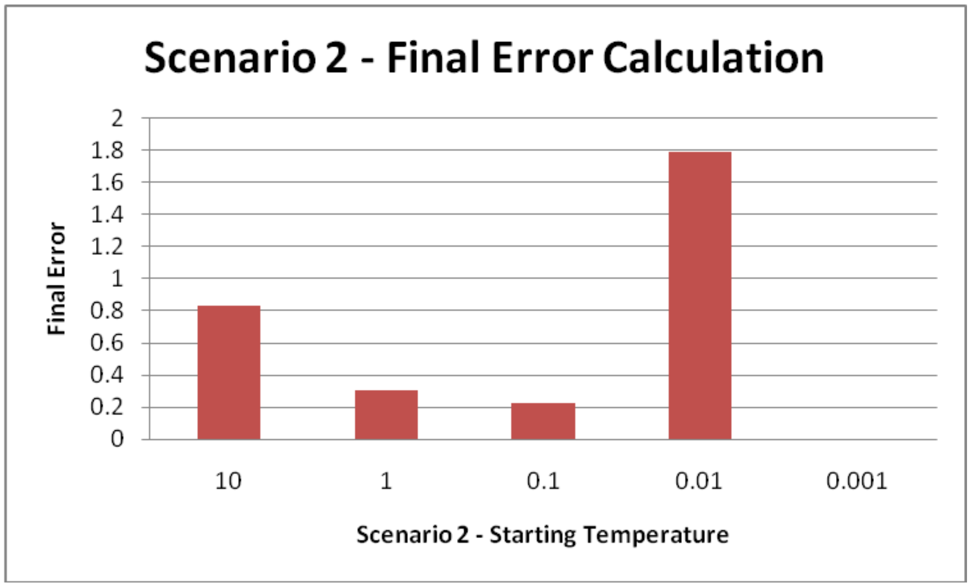


Figure 34 Final Error Calculation for Scenario 2 of Linear Decay Cooling Schedule. The final error at 0.001 could not be computed.

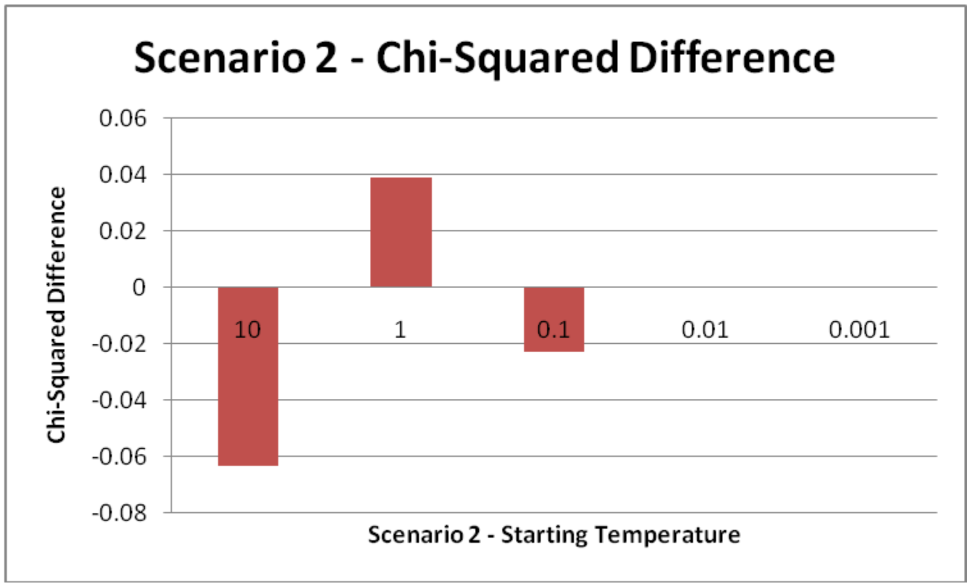


Figure 35 Chi-Squared Difference Calculation for Scenario 2 of Linear Decay Cooling Schedule. Original Group Delay Chi-Squared Value equal to 0.2854. The values at 0.01 and 0.001 could not be computed.

3.4.4 Linear-Exponential and Adaptive Cooling Schedule

The linear-exponential hybrid schedule and the simple adaptive cooling schedule were simulated 25 times each and the average final error and chi-squared results were plotted together in Figures 36 and 37. The linear-exponential schedule follows the model described in 2.6.4.4.

In order to implement the adaptive cooling schedule, the limits described in section 2.6.4.5 needed to be set. These limits were found by visual inspection of the simulation output log during several test iterations.

Table 6 Summary of Adaptive Temperature Ranges

Temperature Category	**Range**
High	Start Temp → 0.5
Medium	0.5 → 0.05
Low	0.05 → Stop Temp

The MATLAB code for implementing this schedule is shown below:

```
%Adaptive Temperature Decay
if(temp < 0.05)
    temp = temp * 0.9;
elseif(temp < 0.5)
    temp = temp * 0.5;
else
    temp = temp * 0.1;
end
```

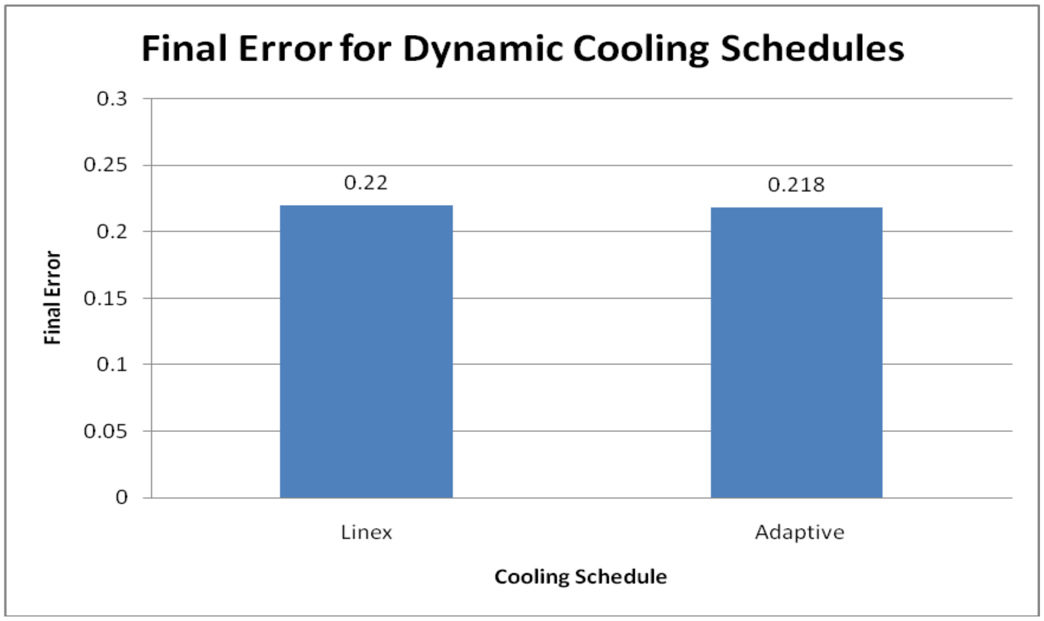


Figure 36 Final Error Calculations for Dynamic Cooling Schedules

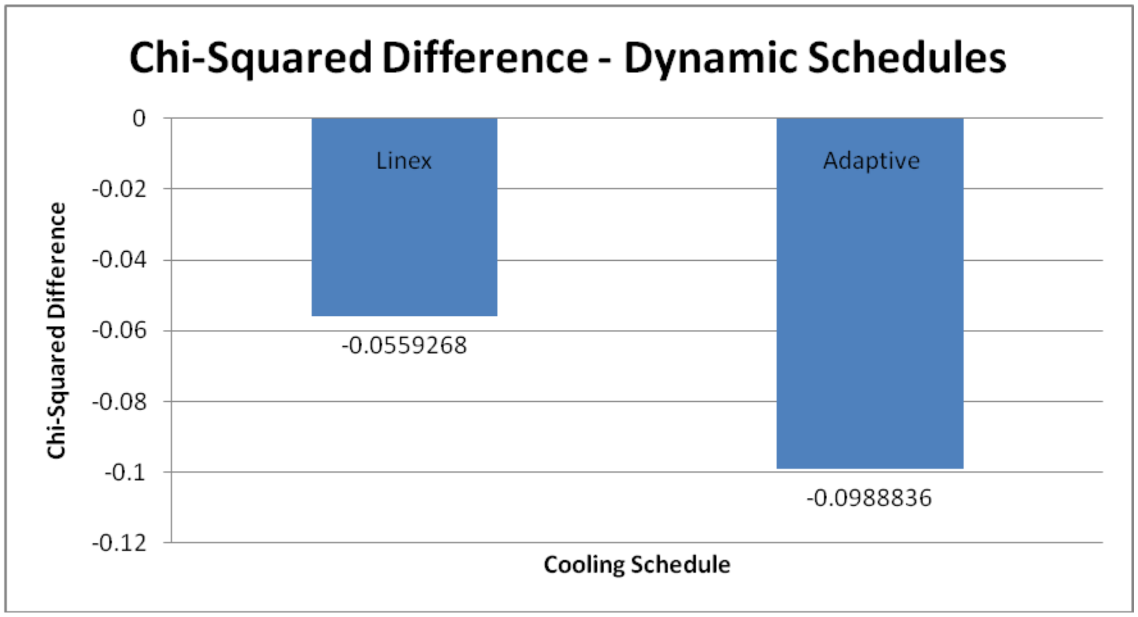


Figure 37 Chi-Squared Difference Calculation for Dynamic Cooling Schedule. Original Group Delay Estimated Chi-Squared Value equal to 0.2854.

4 Analysis

4.1 Starting Temperature

The results from the starting temperature simulations reinforce the original hypothesis that an ideal temperature to begin annealing would be at the lowest point where all or nearly all fiber diameter transitions are accepted or in other words, the acceptance criterion described in section 2.6.1 is approximately equal to one. We calculated, through an empirically-derived formula, that this temperature would be around 1. Our final error graph in Figure 21 confirms that for each set of exponential cooling ratios tested, the optimization performs nearly the same up until the point where the starting temperature becomes too low. Between the starting temperatures of 1 and 0.1, we see a dramatic increase in the final error value for all three cooling ratios. In contrast, moving the starting temperature to higher values such as 10 or 50 did little if anything to improve the performance of the optimization. This result is also expected behavior because this range of temperatures allows frequent movement through the solution space while not necessarily improving the cost function. We can conclude then that lowering the starting temperature from the original default value of 10 down to 1 will not affect the final error but will reduce the overall computational effort in the algorithm. Table 7 summarizes the reduced number of temperature steps explored given a 0.9 exponential cooling ratio after changing the starting temperature from 10 to 1.

Table 7 Reduction in Temperature Steps between Starting Temperature of 10 and 1

Starting Temperature	**10**	**1**
Cooling Ratio	0.9	0.9
Stopping Temperature	1.00E-05	1.00E-05
Number Of Temperature Steps	133	111

While Figure 21 is able to provide some useful information regarding the starting temperature, the results of the Chi-Squared calculations in Figure 22 do not yield much information that allow any reasonable conclusions to be drawn. We do observe a noticeable decrease in the distribution fit (a negative chi-squared difference value) for most cooling schedule environments simulated minus a few cases using the 0.1 exponential cooling ratio. This is not only a contrast to the original hypothesis but also to the results of the final error calculations. We expected to see a similar trend as the final error results from Figure 21.

We may be able to predict the results of the forthcoming exponential cooling schedule simulations from this simulation alone. Comparing these ratios for a particular starting temperature, we see from Figure 21 that the 0.9 cooling ratio yielded the lowest final error, followed by 0.5 and 0.1. This was true for all four groups of starting temperatures tested. On the other hand, we do not see a convincing trend from the chi-squared data of Figure 22. The 0.9 cooling ratio group seemed to perform the worst in this particular test while 0.1 was able to produce some improvement to the distributions in certain instances. This issue will be discussed in a later section.

4.2 Stopping Temperature

Figures 23 and 24 summarize the final error and chi-squared results from the stopping temperature simulation, respectively. From Figure 23, we observe a substantial drop in the error value after reducing the stopping temperature from 1E-1 to 1E-3 and a smaller drop in error from 1E-3 to 1E-5. These results are expected from the hypothesis that an appropriate stopping temperature would be after a period of “zero acceptance” of fiber diameter changes. There does not appear to be a substantial difference between reducing the stopping temperature past 1E-5. We can conclude that the algorithm cannot improve the final solution much more from this lower bound therefore any additional computations are redundant and unnecessary.

In Figure 24, we observe a general trend of improving chi-squared values when the stopping temperature is at least 1E-3. This however was not true using the 0.9 exponential cooling ratio where a significant reduction in the chi-squared value was recorded. Overall, there does not appear to be a correlation between the stopping temperature and the improvement of the chi-squared values from this particular simulation.

4.3 Number of Transitions per Temperature Step

Figure 25 shows the results of the final error computations when the number of transitions per temperature step variable was simulated at 10, 100, 500, 1000, 2500 and 5000.

The simulation was expanded for this parameter to include a second group delay estimated dataset to determine if there is a noticeable difference in the optimization process when a poorly estimated group delay set is compared to a substantially better estimation. From the final error data, we observed a significant improvement in minimizing the final error between 10 and 100 transitions per temperature step. Further improvement is seen between 100 and 500 transitions and slight improvements are recorded after that point. Table 8 below summarizes the final error data.

Table 8 Final Error Summary for the Number of Transitions per Temperature Step Simulation Note: 0.1 and 0.5 refer to the exponential cooling ratio used for each simulation .

		Dataset 1		**Dataset 2**	
		0.1	**0.5**	**0.1**	**0.5**
# of Transitions per Temp Step	**10**	0.7196	0.496	0.8104	0.5792
	100	0.302	0.2316	0.3716	0.3012
	500	0.2124	0.1836	0.2828	0.2416
	1000	0.2072	0.1736	0.2592	0.228
	2500	0.192	0.1688	0.2352	0.2164
	5000	0.1688	0.1708	0.2284	0.2152

From the chi-squared data in Figure 26, we observe an improvement from the group delay estimation after simulated annealing is performed in all test cases. Unfortunately, no definitive trend can be drawn between chi-squared improvement and increasing the number of transitions at each temperature step.

Regarding the two datasets, a noticeable trend can be observed from Figures 25 and 26. Recall that the original group delay estimated chi-squared values for datasets 1 and 2 were 0.3057 and 0.1073, respectively (dataset 2 is a poorer original estimation). In the final error calculations from Figure 25, dataset 1 was able to achieve a lower error value than dataset 2 in all groups for both the 0.1 and 0.5 exponential cooling ratios. From the chi-squared data in

Figure 26, in all cases except at 5000 transitions, dataset 2 yielded a larger improvement than dataset 1. We can conclude that a poorer initial estimate from group delay will allow for a larger gain in chi-squared improvement after the simulated annealing optimization is applied. On the contrary, a poorer group delay estimate will restrict the minimization of the final error value as can be seen from Figure 25. Lower final error values are observed when the initial group delay estimate is more accurate (dataset 1 in this case).

4.4 Cooling Strategy

The aim of the cooling strategy simulations were to explore several commonly used techniques from previous studies in order to compare which cooling schedule maximizes the performance of the simulated annealing optimization for the group delay application. After analyzing each method individually, an overview of the best schedules will be presented.

The exponential cooling schedules simulated in this project created interesting results as can be seen from Figures 27 and 28. For the final error calculations in Figure 27, we see a nice downward trend indicating the improvement in minimizing the error value as we increase the cooling ratio from 0.1 to 0.9. This behavior is somewhat expected and has been commonly observed in other studies as well. It’s worth mentioning again that increasing the cooling ratio can drastically increase the execution time of the simulation due to the increase in the number of temperature steps in the cooling schedule. For example, a 0.7 ratio correlates to 40 steps while a

0.9 ratio will make 133 steps (given default starting and stopping temperatures of 10 and 1E-5, Figure 10). Table 9 below summarizes these two exponential step factors.

Table 9 Summary of 0.7 and 0.9 Exponential Cooling Ratios

NTTS = Number of Transitions per Temperature Step

	Cooling Ratio	
	0.7	**0.9**
Starting Temp	10	10
NTTS	500	500
# Temp Steps	40	133
Stopping Temp	1.00E-05	1.00E-05
Final Error	0.2204	0.2104

A design choice must be made to determine if the 4.8 percent reduction in final error is worth the significant increase in computational effort due to the additional temperature steps.

While the final error is gradually reduced at larger cooling ratios, we see the opposite effect in the chi-squared difference data from Figure 28. The largest improvement in the distribution after simulated annealing is recorded at the lowest cooling ratio of 0.1. For each increasing ratio after that, a steady decline in the chi-squared value is observed. It appears that the more modifications applied to the original dataset during simulated annealing, the worse the outputted fiber diameter distribution after annealing is completed. To further investigate this issue, a new metric is introduced which will quantify the total number of fiber diameter changes (whether they are accepted or not, abbreviated TNFDC) for each exponential cooling schedule. The Schedule_Calc.m code provided in the appendices explains how the computation is performed. Table 10 below compares the TNFDC for each exponential cooling ratio tested.

Table 10 TNFDC Calculation for Each Simulated Cooling Ratio

	0.1	**0.3**	**0.5**	**0.7**	**0.9**
TNFDC	3500	6000	10000	19500	66000

For now, we will accept that an increase in the TNFDC results in a decrease in the chi-squared value after simulated annealing even though a reduced final error may be observed.

In the logarithmic cooling schedule simulations, we see from Figure 29 that a lower value of the "C" parameter in our formula (and equivalently a lower starting temperature) results in a lower final error. In fact, only when using a "C" value of 0.01 or 0.001 are the results comparable to the exponential schedule. The limitation of the logarithmic cooling strategy being too slow is exposed from these results. Since we restrict the schedule to 150 temperature steps, the high and low temperature values (t = 1 and t = 150) do not encompass a very broad range as in the exponential implementation. We see the best results at the lower "C" values because the algorithm operates at critical temperatures where the annealing does the most "downhill" work. For the higher "C" values, the temperature never gets low enough to make consistent improvements to the cost function. Instead, the acceptance probability is still high enough to allow many fiber diameter changes which increase the final error cost function.

In Figure 30, we observe the chi-squared value is only improved at c = 0.1 although not by much. For all other schedules, the fiber distribution after annealing became a worse fit compared to the template distribution.

In the analysis of linear cooling schedules, the results of the final error calculations for scenario 1 in Figure 31 were not promising. While the 0.05 decrement factor yielded the lowest final error (0.8348), it was still much higher than the best logarithmic and exponential schedules. In addition, it takes 200 temperature steps to reach that value. The reason for such a high final error is because the lowest temperature reached before the algorithm terminates is 0.05, much higher than the default lower bound of 1E-5. The cooling schedules in scenario 1 are impractical because they never reach the required “frozen” state where the real optimizing occurs. For the chi-squared results of scenario 1, the 0.05 decrement factor also produced the best result but in all three schedules the distribution underwent negative changes in the chi-squared value.

In scenario 2, we carry over the best performing schedule from scenario 1 (0.05 decrement factor) and compare it to other schedules which also run for 200 temperature steps. Figure 33 shows that starting the annealing at lower temperatures will also lower the final error after optimization up until the point where the starting temperature is too low. We record a final error of 0.2236 when starting at a temperature of 0.1. While this is a good result from a numerical standpoint, starting at a temperature this low transforms our algorithm from simulated annealing to a local search strategy described earlier because very few “uphill” transitions take place. Starting even lower at 0.01 yielded a final error of 1.792 because the algorithm is unable to explore the solution space and find the global minimum cost since the temperature variable blocks most (if not all) “uphill” changes. For the same reason, starting at 0.001 also did not produce useful results.

For the chi-squared data in Figure 34, no trend can be confirmed using the schedules in scenario 2. For starting temperatures of 0.01 and 0.001, no change was made to the fiber diameter distribution in any of the 25 simulation trials which is why the chi-squared difference was 0.

4.5 Dynamic Cooling Schedules

The final error and chi-squared results for the two dynamic schedules explored in this project are presented in Figures 35 and 36, respectively. We see that the linear-exponential and adaptive cooling schedules both performed well and are comparable to the best results of the exponential and logarithmic schedules. The LinEx schedule produced a final error of 0.2200 and the adaptive schedule yielded a result of 0.2180. From a computational metric, the LinEx schedule explores 56 different temperatures (far lower than any other successful schedule) while the adaptive schedule makes 80 temperature steps. In both scenarios, 500 transitions were performed at each step.

From the chi-squared data, we see that both schedules did not experience an improvement in the distribution fit. The linear-exponential schedule had a better chi-squared result than the adaptive schedule (likely due to a lower TNFDC).

4.6 Summary of Select Annealing Schedules

A summary of the best performing (those which yielded the lowest final error average) cooling schedules are listed in Table 11 with a full description of their respective simulation conditions.

Table 11 Summary of the Best Performing Schedules in Each Category

	Exponential	Logarithmic	Linear	LinEx	Adaptive
Starting Temp	10	0.001	0.1	10	10
Number of Transitions per Temperature Step	500	500	500	500	500
Number of Temperature Steps	133	150	200	56	80
Cooling Ratio	0.9	N/A	N/A	Variable	Variable
Decrement Factor	N/A	N/A	0.0005	Variable	N/A
C Value (if applicable)	N/A	0.001	N/A	N/A	N/A
Stopping Temperature	1.00E-05	1.66E-04	0	1.00E-05	1.00E-05
Final Error	0.2104	0.2216	0.2236	0.22	0.218

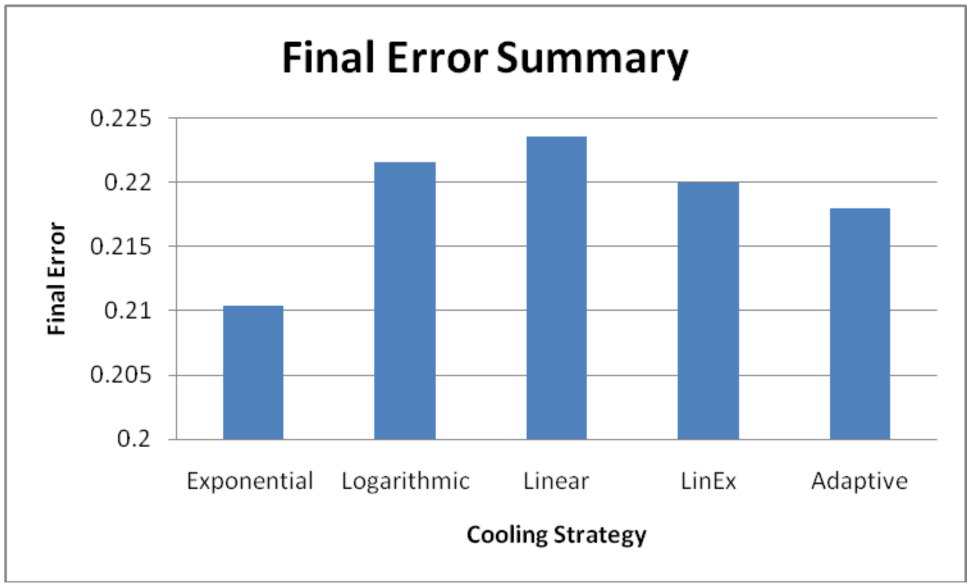


Figure 38 Summary of the Best Performing Schedules in Each Category

While we see that the exponential cooling strategy simulated with a cooling ratio of 0.9 resulted in the lowest final error value, it does not necessarily prove to be the best schedule in this study. Comparing the number of temperature steps we see that the adaptive schedule takes approximately 66% fewer steps than the exponential schedule and only increases the final error by about 3.5%. We can further improve the exponential schedule however by utilizing our results from the starting temperature analysis. If we start the simulation at a temperature of 1, we reduce the number of steps to 111 which is a significant improvement though still greater than the adaptive schedule.

5 Discussion

5.1 Conclusions

This study aimed to investigate an improvement of the optimization process for the estimation of the fiber diameter distribution and associated maximum compound evoked potential acquired by a novel technique known as group delay developed by R.B. Szlavik et al. [12]. A combinatorial optimization algorithm known as simulated annealing was used in [13] to achieve the first set of promising results.

Simulated annealing—whose fundamental methodology can be traced back nearly 60 years but was officially introduced by Kirkpatrick in 1983—has been used extensively in many fields of science and industry for a wide range of applications [17]. Consequently, numerous studies have investigated a particular segment of the algorithm known as the cooling or annealing schedule in an attempt to maximize the performance of the optimization process. While some rules for implementing this set of parameters are widely adopted, most would agree that the design of the cooling schedule is largely problem-independent [20, 27].

The cooling schedule in this particular application was broken down into the individual parameters which guided the convergence of simulated annealing towards a final solution. The starting temperature, stopping temperature, number of transitions at each temperature step as well as the type of cooling strategy were examined and simulated under various environments to determine their effect on optimizing the estimated compound evoked potential when compared to an empirically-generated template compound evoked potential. When appropriate, simulation environments were compared to one another to determine if a specific cooling schedule works best to optimize the group delay estimation. Several schedules composed of various cooling strategies were found to produce comparable results but in fairly different ways. A second metric which quantifies the computational effort required to reach the final solution was introduced to compare schedules which yielded nearly the same final solution but at different convergence rates. It was determined that an adaptive schedule which can vary its cooling rate during different points of the algorithm can perform similarly to the default exponential decay implementation but at a significantly faster rate.

5.2 Chi-Squared Difference Problem

While the final error cost function was able to provide consistent and accurate results for quantifying the performance of the simulations executed in this project, the chi-squared test did not prove to be very useful. After careful study of the simulated annealing algorithm and extensive review of the cooling schedule results, it is determined that it is misleading to assume that the optimization of one function would translate into the optimization of a related function.

In the case of this study, our cost function was the two-norm difference between the template maximum compound evoked potential and the simulated annealing optimized maximum compound evoked potential. It was incorrectly predetermined that minimizing the final error cost function by applying the simulated annealing algorithm would also increase the fit of the fiber diameter distributions (optimized versus template) compared by the chi-squared test. This is not the case because fiber diameter changes which decrease the cost function are always accepted but they may also decrease the chi-squared value. This is also true for changes near the “frozen” state of the system. Even though the final error is improving (being lowered), there is no guarantee that the same is true for the chi-squared value. The algorithm has no knowledge of this secondary test used in our study therefore it does not make any transition decisions based on this value.

Simulated annealing only has one cost function therefore a design choice must be made based on which function is more important: the maximum compound evoked potential or the fiber diameter distribution. In this project, we continued our analysis using the consistency of the final error calculation as the primary metric.

5.3 Future Work

The first and arguably most important study that could be investigated in future works could involve re-writing the annealing.m code to change the cost function to the calculation of the chi-squared value. Again since simulated annealing only has one cost function, the trade-off here is that no guarantee can be made that the maximum compound evoked potential estimation will also be optimized.

To expand on this subject, it may be possible to incorporate two cost functions into the algorithm though no documentation on this topic was found during the research of this project. This would add significant complexity to the acceptance criterion equation and would make fiber diameter changes much more restricted during the simulation.

While this project covered a large number of permutations of cooling schedules, the design of a simulated annealing algorithm has endless possibilities. For example, the number of transitions per temperature step does not have to be a constant. Other cooling schedule implementations will proceed to reduce the temperature after a certain number of accepted or rejected transitions occur. In our implementation, we could allow for a certain number of fiber diameter changes which are accepted due to a lower resulting cost function (not counting those which are accepted by the Boltzmann criterion). Another suggestion could be to not use the Boltzmann criterion at all but implement another technique for accepting uphill transitions.

The original purpose of the group delay and simulated annealing experiment is to develop a non-invasive way to determine the fiber diameter distribution in a nerve segment for those suffering from a neuropathy. While standard nerve conduction velocity tests are useful, they cannot diagnose the specific type of nerve fibers which may be damaged. As a result, many types of neuropathies may be left undiagnosed or untreated. The research performed for this project can be useful in the transition from a simulated environment in MATLAB to an actual real-world experimental setup involving animal models. The electrophysiology lab in Cal Poly’s Biomedical Engineering department can move forward by implementing a LabVIEW equivalent of the group delay and simulated annealing algorithms for use in the current leech model used for experimentation.

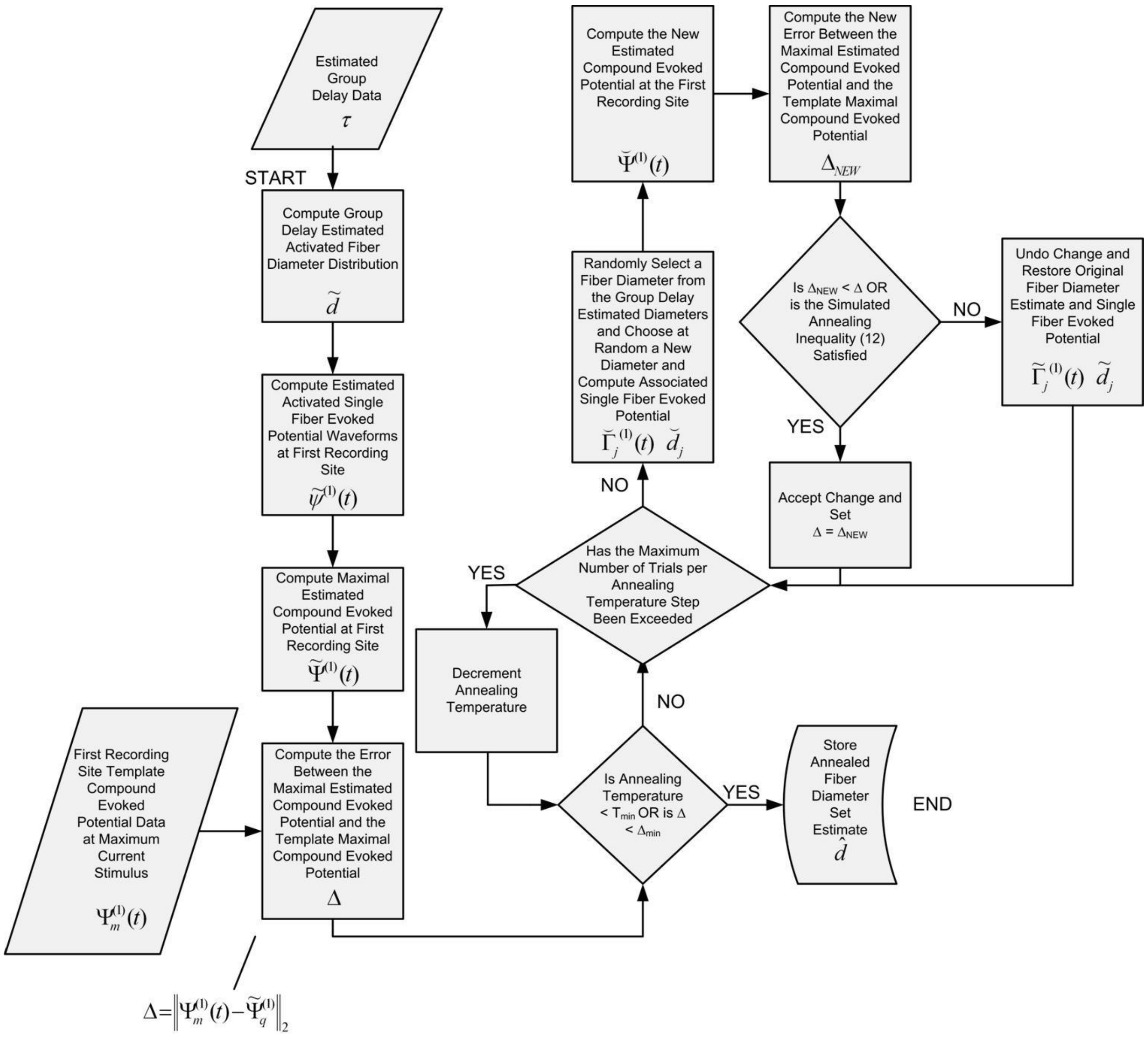
In regards to the implementation of a cooling schedule for simulated annealing, the design decisions are mostly a balance of performance versus computational effort. The choices made are entirely dependent upon the requirements and expectations for the application it is intended for.

References

1. Marieb, E.N. and K. Hoehn, *Human Anatomy & Physiology*. Seventh ed. 2007, San Francisco: Pearson Benjamin Cummings.
2. Matthews, G.G., *Cellular Physiology of Nerve and Muscle*. Fourth ed. 2003: Blackwell Publishing.
3. Szlavik, R., *In Vivo Electrical Stimulation of Motor Nerves,* in *Electrical and Computer Engineering*. 1999, McMaster University: Hamilton, Ontario. p. 160.
4. Gordon Smith, A. and J. Robinson Singleton, *Idiopathic neuropathy, prediabetes and the metabolic syndrome.* J Neurol Sci, 2006. **242**(1-2): p. 9-14.
5. Kernich, C.A., *Patient and family fact sheet. Peripheral neuropathy.* Neurologist, 2001. **7**(5): p. 315-6.
6. Hughes, R., *Peripheral nerve diseases: the bare essentials.* Pract Neurol, 2008. **8**(6): p. 396-405.
7. Poncelet, A.N., *An algorithm for the evaluation of peripheral neuropathy.* Am Fam Physician, 1998. **57**(4): p. 755-64.
8. Minde, J., et al., *A novel NGFB point mutation: a phenotype study of heterozygous patients.* J Neurol Neurosurg Psychiatry, 2009. **80**(2): p. 188-95.
9. Latov, N., *Diagnosis of CIDP.* Neurology, 2002. **59**(12 Suppl 6): p. S2-6.
10. Dorfman, L.J., et al., *Studies of diabetic polyneuropathy using conduction velocity distribution (DCV) analysis.* Neurology, 1983. **33**(6): p. 773-9.
11. Harati, Y., *Diabetic peripheral neuropathies.* Ann Intern Med, 1987. **107**(4): p. 546-59.
12. Szlavik, R.B., *A novel method for characterization of peripheral nerve fiber size distributions by group delay.* IEEE Trans Biomed Eng, 2008. **55**(12): p. 2836-40.
13. Erlanger, J. and H.S. Gasser, *Electrical Signs of Nervous Activity*, ed. U.o.P. Press. 1973, Philadelphia, PA.
14. Szlavik, R.B. and G.E. Turner, *A novel method for characterization of peripheral nerve fiber size distributions by group delay measurements and simulated annealing optimization.* Conf Proc IEEE Eng Med Biol Soc, 2008. **2008**: p. 5008-14.
15. Aarts, E. and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. 1989, Great Britain: John Wiley & Sons.
16. Press, W.H., et al., *Numerical Recipes in Pascal: The Art of Scientific Computing*. 1989, Cambridge, MA: Press Syndicate of the University of Cambridge.
17. Kirkpatrick, S., C.D. Gelatt, Jr., and M.P. Vecchi, *Optimization by Simulated Annealing.* Science, 1983. **220**(4598): p. 671-680.
18. Johnson, D.S., *Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning.* Operations Research, 1989. **37**(6): p. 865-892.
19. Brooks, S.P., *Optimization using simulated annealing.* The Statistician, 1995. **44**: p. 241-257.
20. Nourani, Y.A.B., *A comparison of simulated annealing cooling strategies.* Journal of Physics, 1998. **31**(41): p. 8373-8386.
21. Romeo, F. and A. Sangiovannai-Vincentelli, *A theoretical Framework for Simulated Annealing.* Agorithmica, 1991. **6**(1-6): p. 302-345.

22. Strenski, P.N. and S. Kirkpatrick, *Analysis of finite length annealing schedules.* Algorithmica, 1991. **6**(1-6): p. 346-366.
23. Henderson, D., S. Jacobson, and A. Johnson, *The Theory and Practice of Simulated Annealing.* International Series in Operations Research & Management Science, 2003. **57**: p. 287-319.
24. Ortner, M., X. Descombes, and J. Zerubia, *An adaptive simulated annealing cooling schedule for object detection in images.* Rapport de recherche, 2007. **RR n 6336**.
25. Porat, B., *A Course In Digital Signal Processing*. 1997, New York: John Wiley.
26. Hodes, R., *Linear relationship between fiber diameter and velocity of conduction in giant axon of squid.* J Neurophysiol, 1953. **16**(2): p. 145-54.
27. Atiqullah, M.M., *An Efficient Simple Cooling Schedule for Simulated Annealing.* Lecture Notes in Computer Science, 2004. **3045**: p. 396-404.
28. Geman, S. and D. Geman, *Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images.* Journal of Applied Statistics, 1984. **20**(5): p. 721-741.

Appendix A – Supplemental Figures



Estimated Group Delay Data τ
START
Compute Group Delay Estimated Activated Fiber Diameter Distribution $\tilde{d}$
Compute Estimated Activated Single Fiber Evoked Potential Waveforms at First Recording Site $\tilde{\psi}^{(1)}(t)$
Compute Maximal Estimated Compound Evoked Potential at First Recording Site $\tilde{\Psi}^{(1)}(t)$
First Recording Site Template Compound Evoked Potential Data at Maximum Current Stimulus $\Psi_m^{(1)}(t)$
Compute the Error Between the Maximal Estimated Compound Evoked Potential and the Template Maximal Compound Evoked Potential Δ
$\Delta = \left\| \Psi_m^{(1)}(t) - \tilde{\Psi}_q^{(1)} \right\|_2$
Decrement Annealing Temperature
Is Annealing Temperature $< T_{min}$ OR is $\Delta < \Delta_{min}$
YES
NO
Store Annealed Fiber Diameter Set Estimate $\hat{d}$
END
Has the Maximum Number of Trials per Annealing Temperature Step Been Exceeded
YES
NO
Randomly Select a Fiber Diameter from the Group Delay Estimated Diameters and Choose at Random a New Diameter and Compute Associated Single Fiber Evoked Potential $\breve{\Gamma}_j^{(1)}(t)$ $\breve{d}_j$
Compute the New Estimated Compound Evoked Potential at the First Recording Site $\breve{\Psi}^{(1)}(t)$
Compute the New Error Between the Maximal Estimated Compound Evoked Potential and the Template Maximal Compound Evoked Potential Δ_{NEW}
Is $\Delta_{NEW} < \Delta$ OR is the Simulated Annealing Inequality (12) Satisfied
NO
YES
Undo Change and Restore Original Fiber Diameter Estimate and Single Fiber Evoked Potential $\tilde{\Gamma}_j^{(1)}(t)$ $\tilde{d}_j$
Accept Change and Set $\Delta = \Delta_{NEW}$

Appendix B – MATLAB Source Code

See following pages for MATLAB Source Code.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File:               Test_Annealing.m
%   Revision Date:      03/25/09
%   Author:             Robert B. Szlavik
%
%   Modified By:        Arya Vigeh
%   Last Modified:      03/01/11
%
%   Description:
%   Driver file for group delay and simulated annealing operations. Calls
%   functions for determining the template distributions, computing compound
%   action potentials and performing the group delay estimation followed by
%   simulated annealing optimization.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MAX_TRIALS = 25;            %Total Number of Annealing Trials

% Group Delay Variables
dist_1 = 20.0E-3;
dist_2 = 50.0E-3;
I_o = 0.0;
I_f = 1.0E-3;
Delta_I = 0.5E-6;
mode = 1;
c = 5.0E5;
step = 10.0E-6;
span = 20.0E-3;
r = 1.0E-3;
s_scale = 1;
I = 1;
sigma_e = 1;
alpha = 0.998;
psi_m = 10E-3;
psi_d = 3.5E5;

% Simulated Annealing Variables
sa_mode = 1;
temp_start =10;             %Starting Temperature
temp_factor = 0.9;          %Exponential Cooling Ratio
temp_bound = 1.0E-5;        %Stopping Tmeperature
error_bound = 1.0E-4;       %Error Bound
max_step = 1000;            %Number of Transitions per Temperature Step
fig_num = 8;
t_step = 10;

num_modes = 2;
p_mat = [0.35 7.5E-6 1.699E-6; 0.65 13.0E-6 1.699E-6];
distrib_low_bound = 0;
distrib_high_bound = 20.0E-6;
distrib_step = 1.0E-7;
pop_size = 100;
bin_spacing = 1.0E-7;
```

```
line_limit_factor = 1;

% Generate the Template Fiber Diameter Distribution
[distrib_vect, distrib_function, cum_function, pop_diam_vect] =
Fiber_Distribution(num_modes, p_mat, distrib_low_bound, distrib_high_bound,
distrib_step, pop_size, bin_spacing);

figure(1)
plot(distrib_vect(:,1), distrib_vect(:,2))
title('Original Fiber Distribution');

distrib_vect_original = distrib_vect;

save original_distribution.dat distrib_vect -ASCII -DOUBLE

time_count = span/step;

t = zeros(time_count, 1);

for i = 1:time_count;
    t(i) = (i-1)*step;
end

% Compute the Compound Evoked Potential at the First Recording Site
[cap_1, e_pot_1, act_1, fibers_1] = Compound_Action_Potential(dist_1, I_o,
I_f, Delta_I, c, step, span, r, s_scale, I, sigma_e, alpha, psi_m, psi_d,
pop_diam_vect);

%figure(2)
%plot(t, cap_1)
%figure(3)
%plot(t, e_pot_1)

% Compute the Compound Evoked Potential at the Second Recording Site
[cap_2, e_pot_2, act_2, fibers_2] = Compound_Action_Potential(dist_2, I_o,
I_f, Delta_I, c, step, span, r, s_scale, I, sigma_e, alpha, psi_m, psi_d,
pop_diam_vect);

figure(4)
plot(t, cap_2)
figure(5)
plot(t, e_pot_2)
figure(6)
plot(t, e_pot_1)

diff_count = length(e_pot_2(1,:));

tau = zeros(diff_count,1);
H_Phase = zeros(time_count/2,diff_count);
H_P = zeros(time_count/2,1);
L_M = zeros(time_count/2,1);
Lines_Matrix = zeros(time_count/2,diff_count);
```

```
f = zeros(time_count/2,1);
delay_vect = zeros(diff_count,1);
fiber_pop = zeros(diff_count,1);
template = zeros(time_count,1);

for i = 1:time_count/(2*line_limit_factor)
    f(i) = ((i-1)*1/step)/time_count;
end

% Perform Group Delay Estimation
for i = 1:diff_count
    [tau(i), H_P, L_M, est_diam, est_v]=Group_Delay(e_pot_1(:,i),
e_pot_2(:,i), step, span, line_limit_factor, dist_2-dist_1, c);
    H_Phase(:,i) = H_P;
    Lines_Matrix(:,i) = L_M;
    delay_vect(i)=dist_2/est_v;
    fiber_pop(i) = est_diam;
end

figure(7)
plot(f, H_Phase)
hold on
plot(f, Lines_Matrix,'r')

%Setup the histogram bin vector

count = (distrib_high_bound - distrib_low_bound)/bin_spacing;

distrib_vect_gd = zeros(count,2);

for i = 1:count
    distrib_vect_gd(i) = distrib_low_bound + (i-1)*bin_spacing;
end

distrib_vect_gd(:,2)=hist(fiber_pop, distrib_vect_gd(:,1));

distrib_vect_group_delay = distrib_vect_gd;

figure(2)
plot(distrib_vect_gd(:,1), distrib_vect_gd(:,2))
title('Group Delay Fiber Distribution');

save group_delay_estimated_distribution.dat distrib_vect_gd -ASCII -DOUBLE

% Perform Simulated Annealing Optimization for N = MAX_TRIALS iterations
for r = 1:MAX_TRIALS
    fprintf('%i :  ', r)

    [est_size, est_comp, res_error, fin_temp] = Annealing(sa_mode, fiber_pop,
delay_vect, dist_2, cap_2(:,2000), e_pot_2, step, span, c, temp_start,
temp_factor, temp_bound, error_bound, max_step, r, s_scale, I, sigma_e,
alpha, fig_num, t_step);
```

```
    %Setup the histogram bin vector
    count = (distrib_high_bound - distrib_low_bound)/bin_spacing;

    distrib_vect_sa = zeros(count,2);

    for i = 1:count
        distrib_vect_sa(i) = distrib_low_bound + (i-1)*bin_spacing;
    end

    distrib_vect_sa(:,2)=hist(est_size, distrib_vect_sa(:,1));

    distrib_vect_annealed = distrib_vect_sa;

    template_evoked_potential(:,1) = t(:,1);
    template_evoked_potential(:,2) = cap_2(:,2000);

    save template_evoked_potential.dat template_evoked_potential -ASCII -
DOUBLE
    save optimized_distribution.dat distrib_vect_sa -ASCII -DOUBLE
    save optimized_evoked_potential.dat est_comp -ASCII -DOUBLE

    [prob,df]=Chi_Square(distrib_vect_original, distrib_vect_original);
    %prob
    %df
    [probgd,df]=Chi_Square(distrib_vect_original, distrib_vect_group_delay);
    %probgd
    %df
    [probsa,df]=Chi_Square(distrib_vect_original, distrib_vect_annealed);
    %probsa
    %df
    fprintf('\t GD ChiSquared: %.4g', probgd)
    fprintf('\t SA ChiSquared: %.4g\n', probsa)
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Function:       Annealing
%   Revision Date:  03/25/09
%   Author:         Robert B. Szlavik
%
%   Modified By:      Arya Vigeh
%   Last Modified:  03/04/11
%
%   Implements the simulated annealing algorithm to determine the size
%   distribution of a population of fibers for which the summated compound
%   action potential most close, in an optimized sense, resembles a
%   maximal evoked potential template.
%
%   USES FUNCTIONS:     Function_Centroid
%                       Fiber_Evoked_Potential
%
%   Arguments:
%           sa_mode         =   Annealing mode (1 uses Centroid Function)
%           fiber_pop       =   Vector of fiber diameters from Group Delay
%           delay_vect      =   Associated time delay of each fiber in
%                               fiber_pop
%           dist            =   distance between recording electrodes
%           template        =   Template fiber diameters
%           e_pot           =   Single fiber evoked potentials from
%                               template
%           step            =   10.0E-6
%           span            =   20.0E-3
%           c               =   5.0E5
%
%           temp_start      =   Simulated Annealing Starting Temperature
%           temp_factor     =   Exponential Decay Factor
%           temp_bound      =   Stopping Temperature
%           error_bound     =   Error bound limit for SA
%           max_step        =   Number of iterations at each temperature step
%
%           r           =   1.0E-3
%           s_scale     =   1
%           I           =   1
%           sigma_e     =   1
%           alpha       =   0.998
%           fig_num     =   figure number for MATLAB plots
%           t_step      =   10
%
%
%   Returns:
%           est_size  =   Fiber diameter distribution after annealing
%           est_comp  =   Compound Evoked potential vector from
%                             optimized distribution
%           res_error =   Current value of the Final Error (cost function)
%           fin_temp  =   Last temperature value reached before algorithm
%                             exits
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [est_size, est_comp, res_error, fin_temp] = Annealing(sa_mode,
fiber_pop, delay_vect, dist, template, e_pot, step, span, c, temp_start,
temp_factor, temp_bound, error_bound, max_step, r, s_scale, I, sigma_e,
alpha, fig_num, t_step)

mode = 1;
index = 0;                %for log decay and linex
linear_step = 0.0005;       %for linear decrease
temperature = temp_start;

%LinEx array of values
linear_index = [10 9 8 7 6 5 4 3 2 1
                0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1
                0.09 0.08 0.07 0.06 0.05 0.04 0.03 0.02 0.01
                0.009 0.008 0.007 0.006 0.005 0.004 0.003 0.002 0.001
                0.0009 0.0008 0.0007 0.0006 0.0005 0.0004 0.0003 0.0002
                  0.0001
                9E-05 8E-05 7E-05 6E-05 5E-05 4E-05 3E-05 2E-05 1E-05 0];

count = span/step;

t = zeros(count,1);

for i = 1:count
    t(i) = (i-1)*step;
end

if (sa_mode == 1)

    for i = 1:length(fiber_pop)

        v_centroid = zeros(count,2);

        v_centroid(:,1) = t;
        v_centroid(:,2) = e_pot(:,i);

        [cent_val, centroid] = Function_Centroid(v_centroid, t_step);

        fiber_delay = delay_vect(i);

        if (fiber_delay > cent_val)

            delay_count = floor((fiber_delay - cent_val)/step);

            e_current = e_pot(:,i);
            e_pot(:,i) = zeros(count,1);

            for j = 1:(count-delay_count)
                e_pot(j+delay_count,i) = e_current(j);
            end

        end
        if (fiber_delay <= cent_val)

            delay_count = floor((cent_val - fiber_delay)/step);

            e_current = e_pot(:,i);
            e_pot(:,i) = zeros(count,1);

            for j = 1:(count-delay_count)
                e_pot(j,i) = e_current(j+delay_count);
            end
        end

    end

end

selection_vector = zeros(length(fiber_pop));

compound = zeros(count,1);

min_fib_diam = min(fiber_pop);
max_fib_diam = max(fiber_pop);

error_value = error_bound + 1;
res_error = error_value;
temp = temp_start;

while ((error_value > error_bound) && (temp > temp_bound))

    k = 0;
    oracle_count = 0;
    change_count = 0;
    total_change_count = 0;
    rejected_count = 0;

    while ((k < max_step) && (error_value > error_bound))

        k = k + 1;
        selection_vector = randperm(length(fiber_pop));
        chosen_fiber = selection_vector(1);

        new_fiber_diam = min_fib_diam + rand(1)*(max_fib_diam -
min_fib_diam);
        new_fiber_delay = dist/(c*new_fiber_diam);

        original_fiber_diam = fiber_pop(chosen_fiber);
        original_fiber_delay = delay_vect(chosen_fiber);

        fiber_pop(chosen_fiber) = new_fiber_diam;
        delay_vect(chosen_fiber) = new_fiber_delay;
        if (sa_mode == 0)

            v_vect = Fiber_Evoked_Potential(mode, c, step, span, 
new_fiber_delay, r, new_fiber_diam/2, s_scale, I, sigma_e, alpha);
            e_pot(:,chosen_fiber) = v_vect(:,2);
        end

        if (sa_mode == 1)

            v_centroid = zeros(count,2);

            v_centroid(:,1) = t;
            v_centroid(:,2) = e_pot(:,chosen_fiber);

            [cent_val, centroid] = Function_Centroid(v_centroid, t_step);

            if (new_fiber_delay > cent_val)

                delay_count = floor((new_fiber_delay - cent_val)/step);

                e_current = e_pot(:,chosen_fiber);
                e_pot(:,chosen_fiber) = zeros(count,1);

                for i = 1:(count-delay_count)
                    e_pot(i+delay_count,chosen_fiber) = e_current(i);
                end

            end

            if (new_fiber_delay <= cent_val)

                delay_count = floor((cent_val - new_fiber_delay)/step);

                e_current = e_pot(:,chosen_fiber);
                e_pot(:,chosen_fiber) = zeros(count,1);

                for i = 1:(count-delay_count)
                    e_pot(i,chosen_fiber) = e_current(i+delay_count);
                end
            end

        end

        compound = zeros(count,1);

        for j = 1:length(fiber_pop)
            for i = 1:count
                compound(i) = compound(i) + e_pot(i,j);
            end
        end

        %Calculating Two-Norm Error Value
        error_value = 0;

        for i = 1:count
            error_value = error_value + (template(i) - compound(i))^2;
        end

        error_value = sqrt(error_value);
        change = res_error - error_value;
        oracle_value = rand(1);

        if((oracle_value <= exp(-abs(change)/temp)) || (change >= 0))

            if (oracle_value <= exp(-abs(change)/temp))
                oracle_count = oracle_count + 1;  %Change due to Boltzmann
                                                  %probability
            end

            if (change >= 0)
                change_count = change_count + 1;  %Change due to decreased
                                                  %cost function
            end

            total_change_count = total_change_count + 1;

            res_error = error_value;

        else

            % Change not accepted, so restore original values
            fiber_pop(chosen_fiber) = original_fiber_diam;
            delay_vect(chosen_fiber) = original_fiber_delay;

            if (sa_mode == 0)

                v_vect = Fiber_Evoked_Potential(mode, c, step, span,
original_fiber_delay, r, original_fiber_diam/2, s_scale, I, sigma_e, alpha);
                e_pot(:,chosen_fiber) = v_vect(:,2);

            end

            if (sa_mode == 1)

                e_pot(:,chosen_fiber) = e_current;

            end

            rejected_count = rejected_count + 1;

        end


    end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Cooling Schedules Implemented Below %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Decrement temperature by scale factor (Exponential Decay)
    temp = temp*temp_factor;

    % Logarithmic Decay
    %index = index + 1;
    %if (index < 151)
    %    temp = .001/(log(index) + 1);
    %else
    %    temp = 0;
    %end
    % End Log Decay

    % Start LinEx
    %index = index + 1;
    %temp = linear_index(index);
    % End LinEx

    % Adaptive Temperature Decay
    %if(temp < 0.05)
    %    temp = temp * .9;
    %elseif(temp < 0.5)
    %    temp = temp * .5;
    %else
    %    temp = temp * .1;
    %end


    % Linear Decrease
    %temp = temp - linear_step;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Temperature Decrements       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    compound = zeros(count,1);

    for j = 1:length(fiber_pop)
        for i = 1:count
            compound(i) = compound(i) + e_pot(i,j);
        end
    end

    %fprintf('Temp:  %3.2g, \n', temp)
    %fprintf('\t Error:  %3.2g,', error_value)
    %fprintf('\t Tot Ch:  %d, ', total_change_count)
    %fprintf('\t Con Ch %d \n ', change_count)
    %figure(fig_num)
    %plot(t, template, 'g')
    %hold on
    %plot(t, compound, 'r')
    %hold off


end

fprintf('\t Error:  %3.2g,', error_value)
fprintf('\t Tot Ch:  %d, ', total_change_count)
fprintf('\t Con Ch %d  ', change_count)

est_size = fiber_pop;
est_comp = compound;
fin_temp = temp;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Function:       Chi_Square
%   Revision Date:  04/03/09
%   Author:         Robert B. Szlavik
%
%   Modified By:      Arya Vigeh
%   Last Modified:  3/17/11
%
%   Computes the Chi-Squared Value between two distributions of fiber
diameters.
%
%   USES FUNCTIONS:                     None
%
%   Arguments:  first_distrib   =   the original template distribution of
fiber diameters
%
%           second_distrib  =   a modified distribution of fiber diameters
either after
%                           the group delay estimation or simulated annealing
%                           optimization.
%
%   Returns:    prob                =   chi-squared value
%
%           df          =   degrees of freedom
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Chi Square Calculation

function [prob, df] = Chi_Square(first_distrib, second_distrib)

chi_square = 0;
df=length(first_distrib(:,1))-1;

sum1 = 0;
sum2 = 0;

for i = 1:length(first_distrib(:,1))
    sum1 = sum1+first_distrib(i,2);
    sum2 = sum2+second_distrib(i,2);
end

for i = 1:length(first_distrib(:,1))
    if ((first_distrib(i,2) ~= 0) || (second_distrib(i,2)~=0))
        chi_square = chi_square + ((first_distrib(i,2)-
second_distrib(i,2))^2)/(first_distrib(i,2)+second_distrib(i,2));
    end
    if ((first_distrib(i,2) == 0) && (second_distrib(i,2)==0))
        df=df-1;
    end
end

prob=1-gammainc(0.5*chi_square,0.5*df)';
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Function:       Compound_Action_Potential
%   Revision Date:  03/04/09
%   Author:         Robert B. Szlavik
%
%   Computes an array of compound evoked potentials.
%
%   USES FUNCTIONS:                 Exponential_Activation_Function
%                                   Fiber_Evoked_Potential
%
%   Arguments:  dist                =   propagation distance in (m)
%               I_o                 =   initial stimulus current value in
%                                       (A)
%               I_f                 =   final stimulus current value in
%                                       (A)
%               Delta_I             =   stimulus current increment in (A)
%                                       Fiber_Evoked_Potential function
%               c                   =   velocity diameter constant (1/s)
%               step                =   time step in (s)
%               span                =   simulation time span in (s)
%               r                   =   distance from recording point to
%                                       fiber centerline (m)
%               s_scale =           =   scaling factor for s variable
%                                       (dimensionless) (s = s_scale*a)
%               I                   =   current through the second pole (A)
%               sigma_e             =   extracellular conductivity (S/m)
%               alpha               =   fraction of I distributed to first
%                                       pole
%               psi_m               =   current scaling factor in (A)
%               psi_d               =   exponential scaling factor in (1/m)
%               fiber_pop           =   column vector of fiber diameters in
%                                       (m) where the dimensions are
%                                       fiber_pop[# of fibers, 1]
%
%
%   Returns:    cap                 =   array of compound action potentials
%                                       at each stimulus curent level.
%                                       cap[length(time),# of stim steps]
%               e_pot               =   decomposed evoked potential array
%                                       of non-zero potential increments
%                                       adding into the compound evoked
%                                       potential
%                                       e_pot[length(time),# non-zero
%                                       potential increments]
%               act                 =   array of all activated fibers at
%                                       a stimulus current step.  Position of
%                                       1s in each column vector correspond
%                                       to the activated fiber in the fiber
%                                       population vector
%                                       act[# in fiber_pop, # of stim
%                                       steps]
%               fibers              =   array of newly activated fibers at
%                                       a given stimulus current step.
%                                       Position of 1s in each column
%                                       vector correspond to the activated
%                                       fiber in the fiber population
%                                       vector.
%                                       fibers[# in fiber_pop, # of stim
%                                       steps]
%
%
%   Internal:   mode                =   mode for Fiber_Evoked_Potential
%                                       function.  Set equal to 1 for
%                                       time function.
%               delta_fiber         =   fiber time delay vector in (s) for
%                                         input
%                                       into Fiber_Evoked_Potential
%                                       function.
%                                       delta_fiber[# in fiber_pop]
%               radius              =   fiber radius vector in (m)
%                                       radius[# in fiber_pop]
%               stim_loop_count     =   number of stimulus current steps
%                                       as per
%
%                                       (I_f - I_o)/Delta_I
%
%               time_loop_count     =   number of time steps as per
%
%                                       span/step
%
%               fiber_potential     =   accumulated compound action
%                                       potential vector
%                                       fiber_potential[:,1] = time points
%                                       in (s).
%                                       fiber_potential[:,2] = accumulated
%                                       potential values in (V).
%               stim_val            =   current value of the stimulus
%                                       current amplitude in (A).
%               step                =   unit step function value for
%                                       determining if the current fiber
%                                       evoked potential is added to the
%                                       accumulating compound evoked
%                                       potential.
%               diff_count          =   count variable for the number of
%                                       non zero and thus incremental
%                                       fiber evoked potentials.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [cap, e_pot, act, fibers] = Compound_Action_Potential(dist, I_o,
I_f, Delta_I, c, step, span, r, s_scale, I, sigma_e, alpha, psi_m, psi_d,
fiber_pop)

    mode = 1;
    delta_fiber = zeros(length(fiber_pop));
    radius = zeros(length(fiber_pop));

    stim_loop_count = (I_f - I_o)/Delta_I;
    time_loop_count = span/step;

    fiber_potential = zeros(time_loop_count, 2);
    cap = zeros(time_loop_count, stim_loop_count);
    act = zeros(length(fiber_pop), stim_loop_count);
    fibers = zeros(length(fiber_pop), stim_loop_count);

    for i = 1:length(fiber_pop)
        delta_fiber(i) = dist/(c*fiber_pop(i));
        radius(i) = fiber_pop(i)/2;
    end

    stim_val = I_o;

    for i = 1:stim_loop_count

        for j = 1:length(fiber_pop)

            if stim_val >= Exponential_Activation_Function(psi_m, psi_d,
fiber_pop(j))
                Step = 1.0;
                act(j,i) = 1;
            elseif stim_val < Exponential_Activation_Function(psi_m, psi_d,
fiber_pop(j))
                Step = 0.0;
            end

            fiber_potential = Step*Fiber_Evoked_Potential(mode, c, step,
span, delta_fiber(j), r, radius(j), s_scale, I, sigma_e, alpha);
            cap(:,i) = cap(:,i) + fiber_potential(:,2);

        end

        stim_val = stim_val + Delta_I;

    end

    diff_count = 0;

    fibers(:,1) = act(:,1);

    for i = 2:stim_loop_count

        fibers(:,i) = act(:,i)-act(:,i-1);

        if norm(act(:,i)-act(:,i-1)) ~= 0
            diff_count = diff_count + 1;
        end

    end

    e_pot = zeros(time_loop_count,diff_count);
```

```
diff_count = 0;

for i = 2:stim_loop_count

    if norm(act(:,i)-act(:,i-1)) ~= 0
        diff_count = diff_count + 1;
        e_pot(:,diff_count) = (cap(:,i)-cap(:,i-1));
    end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Function:       Exponential_Activation_Function
%   Revision Date:  02/23/09
%   Author:         Robert B. Szlavik
%
%   Arguments:  psi_m               =   current scaling factor in (A)
%               psi_d               =   exponential scaling factor in (1/m)
%               d                   =   fiber diameter in (m)
%
%   Returns:    act_function        =   fiber activation current value in
%                                       (A)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function act_function = Exponential_Activation_Function(psi_m, psi_d, d)

    act_function = psi_m*exp(-psi_d*d);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Function:       Fiber_Distribution
%   Revision Date:  02/27/09
%   Author:         Robert B. Szlavik
%
%   Arguments:  num_modes           =   Number of Gaussian modes in the
%                                       fiber diameter probability density
%                                       function specified by n
%               p_mat               =   (n,3) matrix of mode perameters
%                                       where the columns are the
%                                       parameters psi, mu and sigma for
%                                       the normalized Gaussian mode
%                                       defined below
%
%   Mode(x)     = (psi)/(sigma*sqrt(2*pi))*exp(-(x-mu)^2/(2*sigma^2))
%
%                                       where x is the diameter in (m)
%               distrib_low_bound   =   lower fiber diameter bound of the
%                                       distribution in (m)
%               distrib_high_bound  =   upper fiber diameter bound of the
%                                       distribution in (m)
%               step                =   probability distribution step size
%                                       in (m)
%               pop_size            =   number of fibers in the simulated
%                                       population.
%               bin_spacing         =   bin spacing size in (m)
%
%   Returns:    distrib_vect(:,1)   =   vector of fiber diameters in (m)
%               distrib_vect(:,2)   =   vector of fiber frequencies  for
%                                       plotting histogram
%               pop_diam_vect       =   vector of actual fiber diameters in
%                                       (m)
%
%
%   Internal:   count                   =   number of points in vector
%                                           based
%                                           on bound span and step or
%                                           bin_size
%               distrib_function(:,1)   =   vector of fiber diameters in
%                                           (m)
%               distrib_function(:,2)   =   probability density vector
%               cum_function(:,1)       =   vector of fiber diameters in
%                                           (m)
%               cum_function(:,2)       =   cumulative distribution vector
%               temp_function           =   vector used in computation of
%                                           the cumulative distribution
%                                           function;
%               x                       =   vector of fiber diameter in (m)
%               random_vect             =   uniformly distribution random
%                                           vector used to generate fiber
%                                           diameter distribution
%               pop_diam_vector         =   randomly generated fiber
%                                           diameter vector in (m)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [distrib_vect, distrib_function, cum_function, pop_diam_vect] =
Fiber_Distribution(num_modes, p_mat, distrib_low_bound, distrib_high_bound,
step, pop_size, bin_spacing)

    count = (distrib_high_bound-distrib_low_bound)/step;

    distrib_function = zeros(count,2);
    cum_function = zeros(count,2);
    temp_function = zeros(count,1);
    x = zeros(count,1);
    random_vect = zeros(pop_size,1);
    pop_diam_vect = zeros(pop_size,1);

%   Compute the probability distribution function.

    for i = 1:count
        distrib_function(i,1) = distrib_low_bound + (i-1)*step;
        x(i) = distrib_function(i,1);
        for j = 1:num_modes
            distrib_function(i,2) = distrib_function(i,2) +
((p_mat(j,1))/(p_mat(j,3)*sqrt(2*pi)))*exp(-(x(i)-
p_mat(j,2))^2/(2*p_mat(j,3)^2));
        end
    end

%  Compute the cumulative distribution function.

    for i = 1:count
        cum_function(i,1) = distrib_low_bound + (i-1)*step;
        for j = 1:i
            temp_function(j) = distrib_function(j,2);
        end
        cum_function(i,2) = step*trapz(temp_function);
    end

%  Do the inverse mapping to compute the population of fiber diameters

    random_vect = rand(pop_size,1);

    for i = 1:pop_size
        pop_diam_vect(i) = interp1(cum_function(:,2), cum_function(:,1),
random_vect(i), 'spline');
    end

%  Setup the histogram bin vector

    count = (distrib_high_bound - distrib_low_bound)/bin_spacing;

    distrib_vect = zeros(count,2);

    for i = 1:count
        distrib_vect(i) = distrib_low_bound + (i-1)*bin_spacing;
```

```
end

distrib_vect(:,2)=hist(pop_diam_vect, distrib_vect(:,1));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Function:       Fiber_Evoked_Potential
%   Revision Date:  02/27/09
%   Author:         Robert B. Szlavik
%
%   Uses Fleisher's model to generate the time or position dependent
%   extracellular waveform of a single fiber evoked potential.
%
%   Arguments:  mode    =   potential as a function of time (mode = 1)
%                           potential as a function of distance (mode = 0)
%               c       =   velocity diameter constant (1/s)
%               step    =   time or distance step in (s) or (m)
%                           respectively
%               span    =   total time in (s) or total length in (m)
%               delta   =   time shift in (s) or space shift in (m)
%               r       =   distance from recording point to fiber
%                           centerline (m)
%               a       =   fiber radius (m)
%               s_scale =   scaling factor for s variable (dimensionless)
%                           (s = s_scale*a)
%               I       =   current through the second pole (A)
%               sigma_e =   extracellular conductivity (S/m)
%               alpha   =   fraction of I distributed to first pole
%                           (dimensionless)
%
%
%   Returns:    v_vect      v_vect(1,:) = vector of time in (s) or distance
%                                         in (m)
%                           v_vect(2,:) = potential (V)
%
%   Internal:   count   =   number of points in vector (count = span/delta)
%               diameter=   fiber diameter (diameter = 2*a)
%               s       =   distance from center 0 to first two poles (m)
%                           (s = s_scale*a) in (m)
%               D       =   parameter defined in Fleisher's paper
%                           (D = (a+s)/(r+s))
%               u       =   distance from the center 0 to the third pole
%                           (u = (s)*(1+alpha)/(1-alpha) in (m)
%               z       =   current distance value in (m)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function v_vect = Fiber_Evoked_Potential(mode, c, step, span, delta, r, a,
s_scale, I, sigma_e, alpha)

    count = span/step;

    v_vect = zeros(count,2);

    diameter = 2*a;
    s = s_scale*a;
    D = (a+s)/(r+s);
    u = (s)*((1+alpha)/(1-alpha));
    if (mode == 1)
        for i = 1:count
            v_vect(i,1) = (i-1)*step;
            z = (v_vect(i,1)-delta)*c*diameter;
            v_vect(i,2) = (I*D^2)/(4*pi*a*sigma_e)*(alpha*exp(-
((D/4)^2)*(((z+s)/a)^2))-exp(-((D/4)^2)*(((z-s)/a)^2))+(1-alpha)*exp(-
((D/4)^2)*(((z-u)/a)^2)));
        end
    end

    if (mode == 0)
        for i = 1:count
            v_vect(i,1) = (i-1)*step;
            z = v_vect(i,1)-delta;
            v_vect(i,2) = (I*D^2)/(4*pi*a*sigma_e)*(alpha*exp(-
((D/4)^2)*(((z+s)/a)^2))-exp(-((D/4)^2)*(((z-s)/a)^2))+(1-alpha)*exp(-
((D/4)^2)*(((z-u)/a)^2)));
        end
    end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Function:       Function_Centroid
%   Revision Date:  03/20/09
%   Author:         Robert B. Szlavik
%
%   Arguments:  v       =   potential and time vector
%                           (:,1) = vector of time points (s)
%                           (:,2) = potential values at time points (V)
%               t_step  =   point step value (speeds up integration)
%
%   Returns:    cent_val                =   centroid value in (s)
%               centroid                =   computed function centroid
%                                           (:,1)=time vector (s)
%                                           (:,2)=centroid_top_function
%                                           (:,3)=centroid_bottom_function
%
%   Internal:   centroid_top_function   =   array holder for t*f(t)
%                                           function for centroid
%               centroid_bottom_function=   array holder for f(t) function
%                                           for centroid
%               step                    =   time step in (s)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [cent_val, centroid] = Function_Centroid(v, t_step)

    centroid = zeros(length(v(:,1))/t_step,3);

    centroid_top_function = zeros(length(v(:,1))/t_step);
    centroid_bottom_function = zeros(length(v(:,1))/t_step);

    step = v(2,1)-v(1,1);

    for i = 1:t_step:length(v(:,1))
        centroid_top_function(i) = abs(v(i,1)*v(i,2));
        centroid_bottom_function(i) = abs(v(i,2));
        centroid(i,1) = v(i,1);
        centroid(i,2) = centroid_top_function(i);
        centroid(i,3) = centroid_bottom_function(i);
    end

    cent_val =
(step*trapz(centroid_top_function))/(step*trapz(centroid_bottom_function));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Function:       Group_Delay
%   Revision Date:  03/15/09
%   Author:         Robert B. Szlavik
%
%   Computes a group delay estimate of the propagation time between
%   two recording sites as well as a group delay estimate of the fiber
%   diameter and the fiber conduction velocity.
%
%   Arguments:  vwfe_1              =   single fiber potential waveform at
%                                       the first recording electrode in
%                                       (V) as a function of the time
%                                       vector with points sampled at step
%                                       (s) for a total of span (s)
%               vwse_2              =   single fiber potential waveform at
%                                       the second recording electrode in
%                                       (V) as a function of the time
%                                       vector with points sampled at step
%                                       (s) for a total of span (s)
%               step                =   time step in (s)
%               span                =   time span in (s)
%               line_limit_factor   =   constant factor that limits the
%                                       length of the phase vector used
%                                       in computing the least squares
%                                       estimate of the slope by
%                                       2*line_limit_factor
%               dist                =   distance between recording
%                                       electrodes in (m)
%               c                   =   velocity diameter constant (1/s)
%
%   Returns:    tau                 =   group delay estimated propagation
%                                       time between the two recording
%                                       sites (s)
%               H_Phase             =   phase spectrum vector (radians)
%               Lines_Vector        =   least squares estimate of linear
%                                       phase response vector (radians)
%               est_diam            =   group delay estimated fiber
%                                       diameter (m)
%               est_v               =   group delay estimated fiber
%                                       propagation velocity (m/s)
%
%
%   Internal:   f                   =   frequency vector (Hz)
%               H_of_f              =   fiber frequency response H(f)
%                                       vector
%               H_of_f_Phase        =   angle of the H(f) vector (radians)
%               sum_numerator       =   variable used in computing the
%                                       least squares estimate of the
%                                       slope of the phase response
%               sum_denominator     =   variable used in computing the
%                                       least squares estimate of the
%                                       slope fo the phase response
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [tau, H_Phase, Lines_Vector, est_diam, est_v] = Group_Delay(vwfe_1,
vwse_2, step, span, line_limit_factor, dist, c)

count = span/step;

f = zeros(count,1);
Lines_Vector = zeros(count/2,1);
H_of_f = zeros(count,1);
H_Phase = zeros(count/2,1);

for i = 1:count
    f(i) = ((i-1)*1/step)/count;
end

S_fe = fft(vwfe_1);
S_se = fft(vwse_2);

for i = 1:count
    H_of_f(i) = S_se(i)/S_fe(i);
end

H_of_f_Phase = unwrap(angle(H_of_f));

for i = 1:count/2
    H_Phase(i,1) = H_of_f_Phase(i);
end;

sum_numerator = 0.0;
sum_denominator = 0.0;
for i = 1:count/(2*line_limit_factor)
    sum_numerator = sum_numerator + f(i)*H_of_f_Phase(i);
    sum_denominator = sum_denominator + f(i)^2;
end
Lines = sum_numerator/sum_denominator;

for i = 1:count/(2*line_limit_factor)
    Lines_Vector(i) = Lines*f(i);
end

tau = (-1/(2*pi))*Lines;
est_diam = dist/(c*tau);
est_v = dist/tau;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filename: Schedule_Calc.m
% Author: Arya Vigeh
% Last Modified: 1/18/11
% Description: Calculates the Total Number of Fiber Diameter Changes
%               (TNFDC) for Exponential Cooling Schedules
%
%   Inputs:
%       start_temp : The starting temperature
%       stop_temp : The stopping temperature
%       max_trials : The number of transitions per temperature step
%       factor : The exponential cooling ratio (factor)
%
%   Outputs:
%       cs_value : Total Number of Fiber Diameter Changes (TNFDC)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [cs_value] = Schedule_Calc(start_temp, stop_temp, max_trials,
factor)
    starttemp = start_temp;
    stoptemp = stop_temp;
    trials = max_trials;
    scale = factor;

    count = 0;
    i = starttemp;
    while i > stoptemp
        count = count + trials;
        i = i*scale;
    end


    cs_value = count



end
```