

# Implementation of a Line Attractor-Based Model of the Gaze Holding Integrator Using Nonlinear Spiking Neuron Models

by

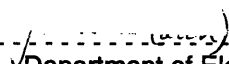
Ben Y. Reis

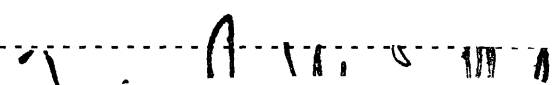
Submitted to the Department of Electrical Engineering and Computer Science  
in Partial Fulfillment of the Requirements for the Degree of  
Master of Engineering in Electrical Engineering and Computer Science  
at the Massachusetts Institute of Technology

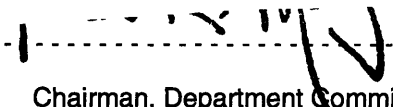
February 2, 1996

Copyright 1996 Ben Y. Reis. All Rights Reserved.

The author hereby grants to M.I.T. permission to reproduce  
distribute publicly paper and electronic copies of this thesis  
and to grant others the right to do so.

Author  .....  
Department of Electrical Engineering and Computer Science  
February 2, 1996

Certified by  .....  
Mirganka Sur  
Supervisor

Accepted by  .....  
F.R. Morgenthaler  
Chairman, Department Committee on Graduate Theses

MASSACHUSETTS INSTITUTE  
OF TECHNOLOGY

JUN 11 1996

Eng.

Implementation of a Line Attractor-Based Model of the Gaze Holding Integrator  
Using Nonlinear Spiking Neuron Models

by

Ben Y. Reis

Submitted to the  
Department of Electrical Engineering and Computer Science

February 2, 1996

In Partial Fulfillment of the Requirements for the Degree of  
Master of Engineering in Electrical Engineering and Computer Science

**ABSTRACT**

The gaze holding integrator of the oculomotor system performs a temporal integration on the rate-encoded eye velocity signals fed to it from the visual and vestibular systems. During intervals between eye movements, the brain holds the eyes motionless using a memory of eye position that is believed to be maintained by the gaze holding integrator of the oculomotor system. When the eyes are motionless, the dynamics of the integrator network can be described qualitatively in terms of a line attractor in its state space. In this study, we construct a neural network model with approximate line attractor dynamics, using biologically plausible spiking nonlinear neuron models. Several classes of networks and feedback are discussed, and a local learning rule is developed. These networks are capable of maintaining multiple stable patterns of persistent activity, corresponding respectively to multiple eye positions. The number of such states is limited by the number of neurons in the network, leading to the prediction that the integrator can maintain only a finite number of eye positions, an effect that should be easier to observe in smaller networks. Implications on future experiments on the relatively small integrator network in goldfish are discussed.

AT&T Bell Laboratories Supervisors: David Tank, Sebastian Seung, Dan Lee  
MIT Thesis Advisor: Professor Mirganka Sur

## **I. Introduction**

Recent studies in computational neuroscience have focused on merging the growing understanding of cellular phenomena with a better understanding of large scale network computation capabilities. To pursue this goal, a system is needed that is appropriate for experimental studies on both the cellular and network levels. Such a system would facilitate exploration of how global computational capabilities emerge from local cellular phenomena.

Current research on neural networks also involves attempts to understand the roles played by feedback. Feedback issues arise in questions involving system dynamics, stability, and learning. This paper discusses these issues and how they pertain to our analysis of the integrator of the oculomotor system.

### **The gaze holding integrator of the oculomotor system**

The brain directs fast eye movements during saccades and slow movements when tracking moving objects or while counteracting head motion. The optokinetic system and the vestibuloocular reflex (VOR) are employed to counteract the effects of motion on vision, thereby preventing image drift across the retina.

A region of the oculomotor system, located in the prepositus hypoglossi and medial vestibular nucleus (MVN) in the hindbrain, receives its inputs from the semicircular canals (involved in the vestibuloocular reflex), and from the premotor brainstem neurons (involved in normal eye saccades) in the form of rate encoded horizontal eye velocity signals. The output from this region to the abducens nuclei includes a rate encoded horizontal eye position signal. This region performs a temporal integration on its inputs, from velocity to position, and is therefore referred to as the integrator [1].

In addition to its integration capabilities, the integrator's function also suggests that it maintains an independent memory of eye position which it uses to keep the eyes at a fixed position [1]; thus its full name -- the Gaze Holding Integrator.

The memory of eye position is believed to be encoded in the firing rates of the neurons constituting the integrator network. Experimental evidence has shown that this network can maintain many patterns of persistent firing activity. Each one of these patterns corresponds to a specific eye position [2, 3].

### **Why study the integrator?**

Several attributes of the small animal gaze holding integrator render it a system highly suited for studies on both the cellular and network levels. Experimental access to behavioral inputs is simple; animals normally perform saccades every few seconds and the VOR can easily be invoked by simply turning the animal. The output of the system is also reasonably easy to monitor by using a scleral search coil to precisely measure changes in eye position. Studies have suggested that the integrator is located in a small, easily reachable region of the brain, enabling detailed recordings from individual neurons using a variety of techniques [4]. In goldfish (*Carassius Auratus*) this area is reported to contain on the order of 25-40 neurons [4], making it one of the smaller, and presumably simpler, examples of integrator systems available for study.

Investigations of the integrator may also contribute to a better understanding of phenomena involving system dynamics, stability, and learning. Processes underlying real-time temporal integration and system responses to changing inputs are presumed to rely heavily on system dynamics. In addition, it is plausible that memory of eye position is an expression of different stable states of the system. Finally, learning has previously been studied in the integrator [5], a prime example of a system which adjusts its responses dynamically in order to adapt to changing conditions. This paper will examine the issues of dynamics, stability, and learning with an emphasis on the roles played by different types of feedback in the integrator network.

### **Modeling: A line attractor approximation**

Development of an entire integrator with a linear response characteristic is beyond the scope of this project. Our efforts have focused on modeling an important component of integrator function: namely, the process of maintaining gaze stability at multiple points. To this end, we have modeled a network that can maintain many discrete patterns of persistent activity.

The N-dimensional state space of a network depicts the various combinations of firing rates of the N neurons in the network [6, 7, 8]. The integrator system responds to changing inputs in a manner that counteracts deviations from its equilibrium states; the dynamics cause the system to evolve by descending along gradients of its energy function. The stability of each point is determined by the gradients of the energy function surrounding that point. Fully stable points are located at energy minima depressions.

Seung [9] has observed that the integrator network displays an approximate line attractor dynamics. Experiments have shown that individual neurons in this system display a linear rate vs. eye position behavior [2, 3]. Likewise, the combined rate vs. position behavior of the entire integrator network is also linear, forming a N-dimensional line in the network's state space.

We infer that the range of observed eye positions corresponds to a range of points along a line in the state space of the integrator network. Indeed, all experimental measurements of persistent activity resulted in values corresponding to points along this line, indicating that all other points throughout the system are unstable. This line in state space is said to be attractive because the trajectories of transitions from all points in state space evolve to some point on the line.

This behavior is best visualized by means of a state space landscape depiction of the system's energy function. A line attractor in a network's state space lies along the bottom of an elongated trough in the landscape depicting the energy function of the system. The trough is bounded by steep slopes representing the steep gradients down which all states relax to the line. Therefore, whenever the system is shifted to any point off the line, it will respond with fast relaxation towards the line.

Each of the points along the line that corresponds to a stable eye position is termed a fixed point, since once the system evolves to that point, it remains there unless perturbed. It is important to note that these points are not fully stable, as they do allow for transitions to other states along the line.

A perfect line attractor, with no motion along the line, can be constructed easily with linear neuron models [10, 11]. It is, however, quite difficult to imagine a perfect line attractor constructed from biologically realistic nonlinear neuron models; this would require perfect cancellation of the neuron nonlinearities as a precondition of achieving system linearity.

We have therefore chosen to investigate networks characterized by dynamics that only approximate those of a line attractor. The large-scale trough structure remains intact, but the dynamics along the line at the base of the trough now consist of shallow energy gradients with local depressions corresponding to the fixed points of the system. An approximate attractor model could consist of either many stable fixed points along a line, or alternatively of one or more stable fixed points along a line distinguished by very slow relaxation rates along the direction of the line.

Experiments [12] have shown that during normal saccades, saccadic command neurons stimulate the integrator with input pulses of rate encoded velocity information. The integrator network integrates these pulses into a position signal. When the network is repeatedly stimulated by a succession of input pulses, each pulse perturbs the system off the line, after which the line attractor dynamics of the system cause fast relaxation to a nearby point on the line. Input pulses represent velocity steps, and each point along the line represents a discrete value in a range of position codes. Repeated unidirectional input velocity pulses would thus step the system through a series of stable states encoding for eye positions further along that direction. The line attractor dynamics can thus facilitate velocity to position integration of inputs to the network.

Previous studies [10, 11] have employed linear neurons to construct integrator network models. Using the line attractor structure as a guide, we set out to develop a network model that incorporates biophysically realistic spiking neurons with thresholds and nonlinearities included in their response characteristics.

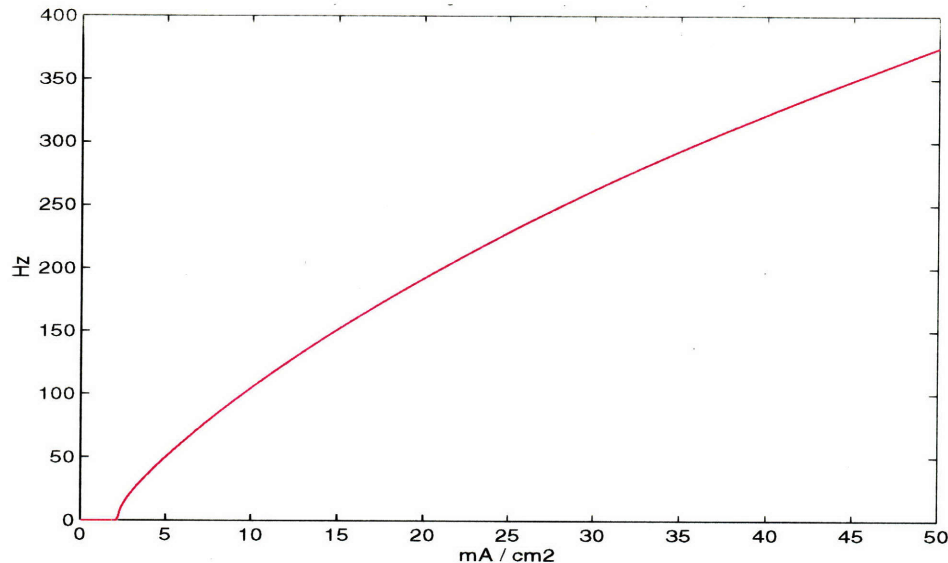
### **Learning**

In addition to enabling detailed elucidation of integrator function, this model enables investigation of integrator plasticity; the integrator has a well known capacity to adapt in order to respond to changing external conditions. Considering biological feasibility, a local learning rule must rely only on information that is actually available to each particular neuron. Previous models of local learning rules have employed linear neurons [13, 14, 15]. In this work, we have employed nonlinear neuron models to study a local learning rule and have also attempted to formulate an understanding of how such learning rules are derived.

## **II. Methods**

### **Neuron models**

A computer was used to simulate the nonlinear neuron models. This single compartment model incorporates six types of membrane channels based on equations from Hansel & Sompolinsky [16] (see appendix). The neurons have a threshold of firing above zero input current, and a nonlinearity at low frequencies in the frequency/current ( $f/i$ ) response curve. At mid-range frequencies, the  $f/i$  curve is close to linear and then begins to roll off at higher frequencies. (Figure 1).



**Figure 1. F vs. I characteristic curve from steady state data collected using a spiking neuron model. Frequency is measured in Hz, current density is in mA/cm<sup>2</sup>.**

In choosing a neuron model we wanted biophysically realistic neurons, specifically those with the characteristic nonlinearities that are found in the neurons of the integrator. The Hansel & Sompolinsky model was chosen because it had the essential features described above, and which are also evident from the detailed study of MVN neurons [17, 18, 19, 20].

### Network models

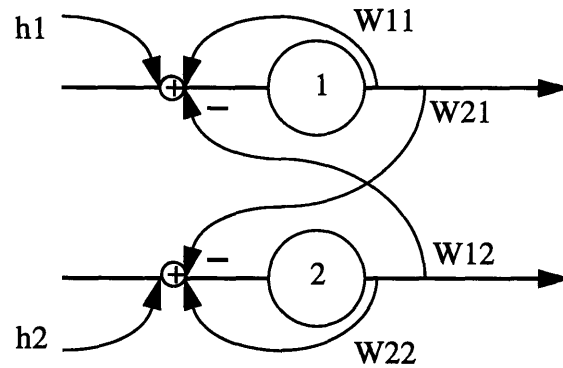
For our network models, we used fully connected networks of neurons with lateral inhibition and excitation, as well as external feed-forward input currents. A neuron's input current depends on the weighted linear summation of all the neurons' firing activities plus its external feed-forward input current.

Apart from external stimulus pulses, the neurons of the integrator network also receive constant tonic inputs from the vestibular and visual systems. The tonic input is incorporated into holding currents which are used to give individual neurons different thresholds, by offsetting the  $f/i$  curve of each neuron accordingly.

The synaptic weights are kept in a weight matrix of current amplitudes. Our model assumes linear summation of currents at the postsynaptic cell. This assumption affects both spatial summation (two simultaneous stimuli) and temporal summation (two stimuli in series) in the postsynaptic cell.

In the simulations, a neuron is said to 'fire' if its membrane potential crosses -20 mV with an upward slope. The neuron then initiates a synaptic current into all neurons onto which it synapses, according to the corresponding synaptic weights. The timecourse of the synaptic currents is modeled using a double exponential 'alpha' function with a 20 msec rise time and 150 msec decay. Please see the appendix for technical details of implementation.

Biologically we see diversity not only in the neurons' thresholds and tonic inputs, but also in the actual shape of the  $f/i$  curve[17, 18, 19, 20], and in the timecourse of the synaptic currents [21]. Our model did not include a diversity of the latter two properties.



**Figure 2. Two neuron network with lateral connections and external stimulation.  $W$  is the weight matrix and  $W_{ij}$  is the amplitude of the current that is injected into neuron  $i$  whenever neuron  $j$  fires.  $h_i$  is the external holding current that determines the threshold of neuron  $i$ . The network in this particular example displays lateral inhibition and self stimulation.**

### Rate model

In order to perform analysis on the various network models and to determine the weights and the holding currents, an analytic rate model was developed that incorporated the steady state nonlinearity in the frequency vs. current ( $f/i$ ) characteristic of the system. A function  $g(u)$  was set up to model the  $f/i$  relationship, and the system dynamics were analytically modeled accordingly:

A fully connected network of neurons with lateral inhibition and excitation between neurons and external holding currents producing different thresholds can be modeled as:

$$u_i + \tau_s \dot{u}_i = \sum_j W_{ij} g(u_j) + h_i \quad (\text{EQ 1})$$

where  $u$  represents the currents of the neurons and  $g(u)$  is a function relating a neuron's steady state firing rate to its input current (the development of such a function is described in the appendix).  $\tau_s$  is the timescale of all the synapses.  $h$  represents the sum of the holding currents and feed forward inputs to each of the neurons, and  $W_{ij}$  is the connection matrix of weights between the neurons. (Figure 2).

Even though it did not model the individual action potentials and synaptic currents of the nonlinear spiking neurons, this rate model was sufficient for determining the fixed points of the network. This seemed reasonable since the synaptic currents had a slow response time on the order of 150 ms, and therefore the net synaptic current into a neuron appeared smooth despite the discreteness of the action potential outputs of the presynaptic cells.

### III. Constructing the Networks

We set out to use spiking non-linear model neurons to construct a network with dynamics that approximates some form of line attractor. Our strategy focuses on constructing a line attractor by placing many fixed points along a line in the state space of a system, or even only a few fixed points with very long relaxation times.

## Networks

In constructing a network, the task is to determine the set of connection strengths, or weights, of a fully connected set of similar neurons, as well as the external holding currents feeding the neurons in the network. A weight matrix along with a vector of holding currents fully describe any one of the networks discussed in this paper.

We will first formalize a system for developing weight matrices and then attempt to guide that development with the goals of many fixed points and/or long relaxation times.

### Networks with a rank one weight matrix

In order to construct a line attractor, we have chosen to examine only networks in which the weight matrix has a rank of one -- that is, any network in which one eigenvalue is close to unity and the others are all close to zero. In this type of network, the set of solutions at equilibrium approaches a line in state space.

Also, there is an added convenience in terms of visualization. It is, in general, difficult to visualize system dynamics in the state space of networks that have more than two neurons. For networks with rank one weight matrices, however, the dynamics of the system in N-dimensional space is reducible to one variable, and thus easy to visualize.

To constrain our network to rank one, we construct the weight matrix from the outer product of two vectors:

$$W_{ij} = \xi_i \eta_j \quad (\text{EQ 2})$$

Since  $\dot{u} = 0$  at equilibrium, we rewrite EQ 1 as:

$$u_i = \xi_i \sum_j \eta_j g(u_j) + h_i \quad (\text{EQ 3})$$

For convenience, we now wish to reduce the system to one variable,  $E$ . Defining:

$$E \equiv \sum_j \eta_j g(u_j) \quad (\text{EQ 4})$$

Plugging back in for  $E$  in EQ 3 we get:

$$u_i = \xi_i E + h_i \quad (\text{EQ 5})$$

Plugging back in for  $u_j$  in EQ 3 yields:

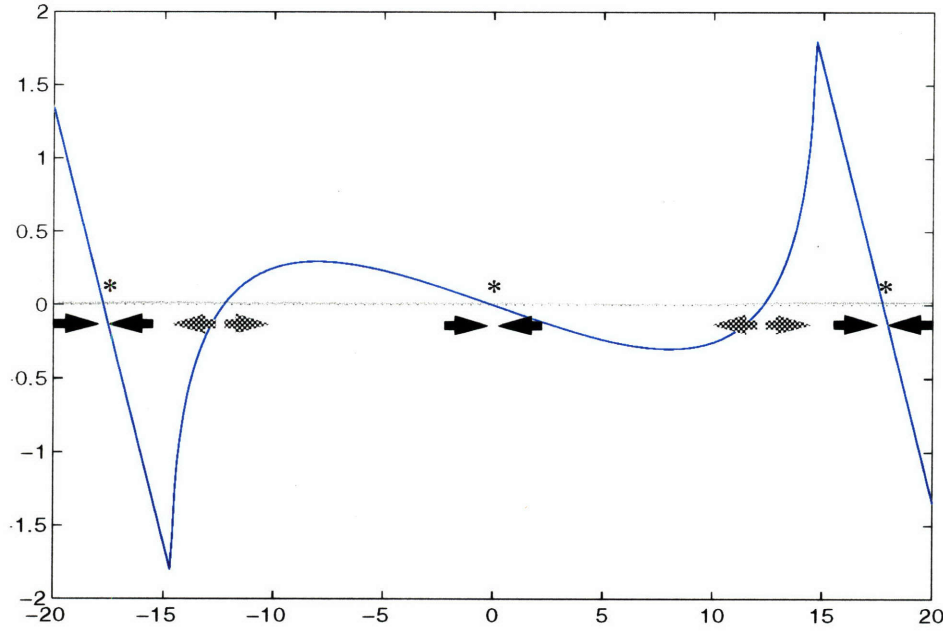
$$u_i = \xi_i \sum_j \eta_j g(\xi_j E + h_j) + h_i \quad (\text{EQ 6})$$

Finally, setting EQ 5 equal to EQ 6, we arrive at a self consistent definition for  $E$ :

$$E = \sum_j \eta_j g(\xi_j E + h_j) \equiv f(E) \quad (\text{EQ 7})$$

This is the equilibrium expression derived from the original EQ 3 in terms of the new variable  $E$ . We have reduced the n-dimensional state space of the system to one dimension, so  $E$  is now the state variable of the system. When  $E = f(E)$ , the system is in equilibrium.





**Figure 3. Dynamics of a two neuron network reduced to one variable. The figure is a plot of  $f(E) - E$  vs.  $E$ . Crossings represent fixed points, as the system is in equilibrium whenever  $f(E) = E$ . The dynamics (indicated by the arrows) dictates that crossings with negative slope are stable fixed points and that crossings with positive slopes are unstable fixed points. This system has three stable fixed points (asterisks). The parameters used are:**

$$\vec{\xi} = [0.197 \ -0.197], \vec{\eta} = [0.197 \ -0.197], \vec{h} = [5.068 \ 5.068], \text{ or, } \vec{\theta} = [-14.624 \ 14.624].$$

For purposes of identifying neurons on various plots, we introduce the variable  $\theta$  to represent the value of  $E$  at which neuron  $i$  has its threshold. The threshold of the  $f/i$  function  $g(u)$  is measured at 2.187. We set

$$\theta_i = \frac{2.187 - h_i}{\xi_i} \quad (\text{EQ 8})$$

Plugging this into EQ 7 yields a more useful form for plotting purposes which allows for easier identification of neurons and their thresholds:

$$E = \sum_j \eta_j g(\xi_j E - \xi_j \theta_j + 2.187) \equiv f(E) \quad (\text{EQ 9})$$

## Energy landscape of a network

To construct the networks, we choose a random set of initial values for the weights ( $\vec{\eta}, \vec{\xi}$ ) and thresholds ( $\vec{h}$ ) of the system. We then use a non negative least squares minimization based on EQ 7 to adjust these parameters. The procedure selects a subset of neurons from an original randomly generated set, and adjusts the selected weights in order to minimize error  $f(E) - E$  over a specified range of  $E$ . If the error were zero along the whole range, the network would be a perfect line attractor, with no motion along the line itself. This is not possible with the intrinsic nonlinearities of the neurons, and instead, the algorithm ends up with a network that has small values of  $f(E) - E$  over the specified range of  $E$ . As an example, let us consider a simple two neuron network (Figure 2) with each neuron making an excitatory connection onto itself and an inhibitory connection onto the other neuron. (This system breaks Dale's law, but is nevertheless instructive.)

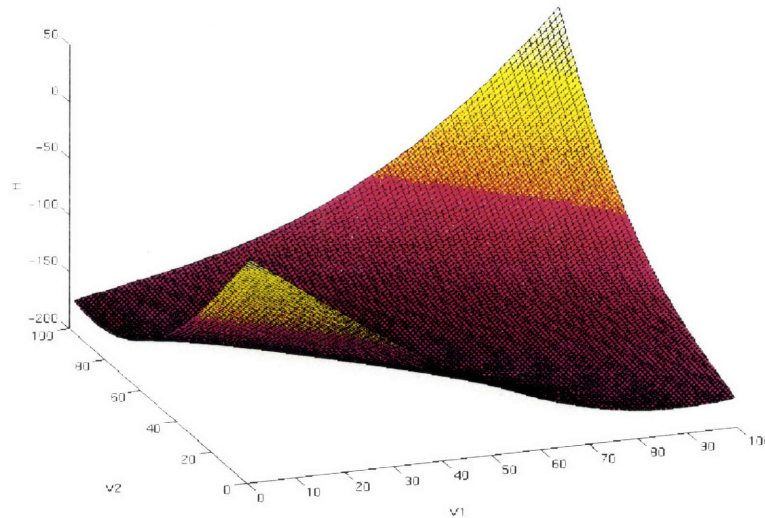
Most solutions include a number of crossings where  $f(E) - E = 0$  -- these are the fixed points of the system. The dynamics of the system is driven by the difference  $f(E) - E$ . If the difference is large, the system moves quickly toward the fixed point. If the difference is small, the relaxation time of the system becomes very long. Because we set out to achieve long relaxation times, we try to minimize the overall error of the system; if we cannot form a fixed point ( $f(E) - E = 0$ ), we at least try to make the relaxation time to the nearest fixed point as long as possible ( $f(E) - E$  small).

To determine which fixed points are stable we look at the difference  $f(E) - E$ . If the difference is positive, the state of the system heads towards positive values of  $E$ , and if the difference is negative, the state system heads towards negative values of  $E$ . Therefore, the stable fixed points are located wherever the line crosses the axis with a negative slope, while the positive slope crossings represent unstable fixed points. Everywhere else along the line, the system slowly relaxes to the nearest stable fixed point. (Figure 3)

So far we have examined a 1 dimensional energy function of the system, in the one dimensional  $E$ -space. The energy function of the original system can be converted to a Lyapunov function  $H(\vec{v})$ , where  $\vec{v}$  is a vector of the firing rates of the neurons in the network.  $H(\vec{v})$  is plotted for this two neuron case in Figure 4. Note the trough running along the middle of the graph. This is a typical approximation of a line attractor. When the system's state is at a point distant from the trough line, fast relaxation towards a fixed point can be represented by the motion down the steep energy gradient leading to a point on the trough line.

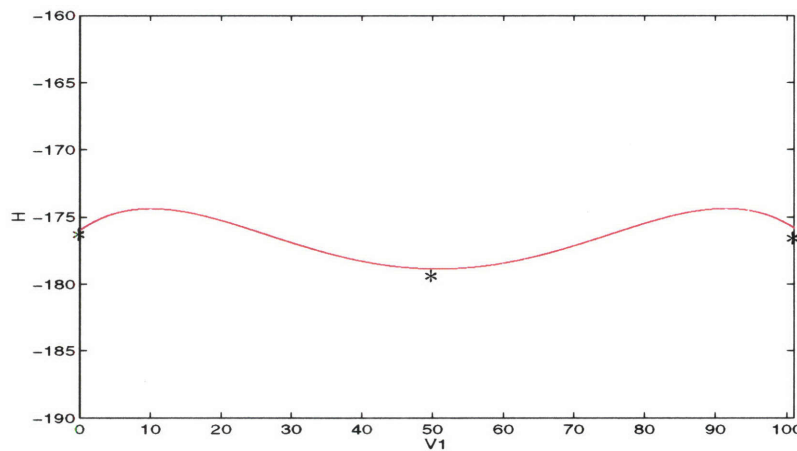
Upon closer examination, however, one notices that the base of the trough is not perfectly flat. (Figure 5). It contains local depressions, or energy minima, corresponding to the stable fixed points of the system. In this example, one of the fixed points is reached when both neurons are firing at the same rate, while the other two fixed points occur when one of the neuron is firing at a high rate and the other is quiescent.

All other points along the trough line are situated on very shallow gradients leading to nearby local depressions, which manifest themselves with long relaxation times along the trough.

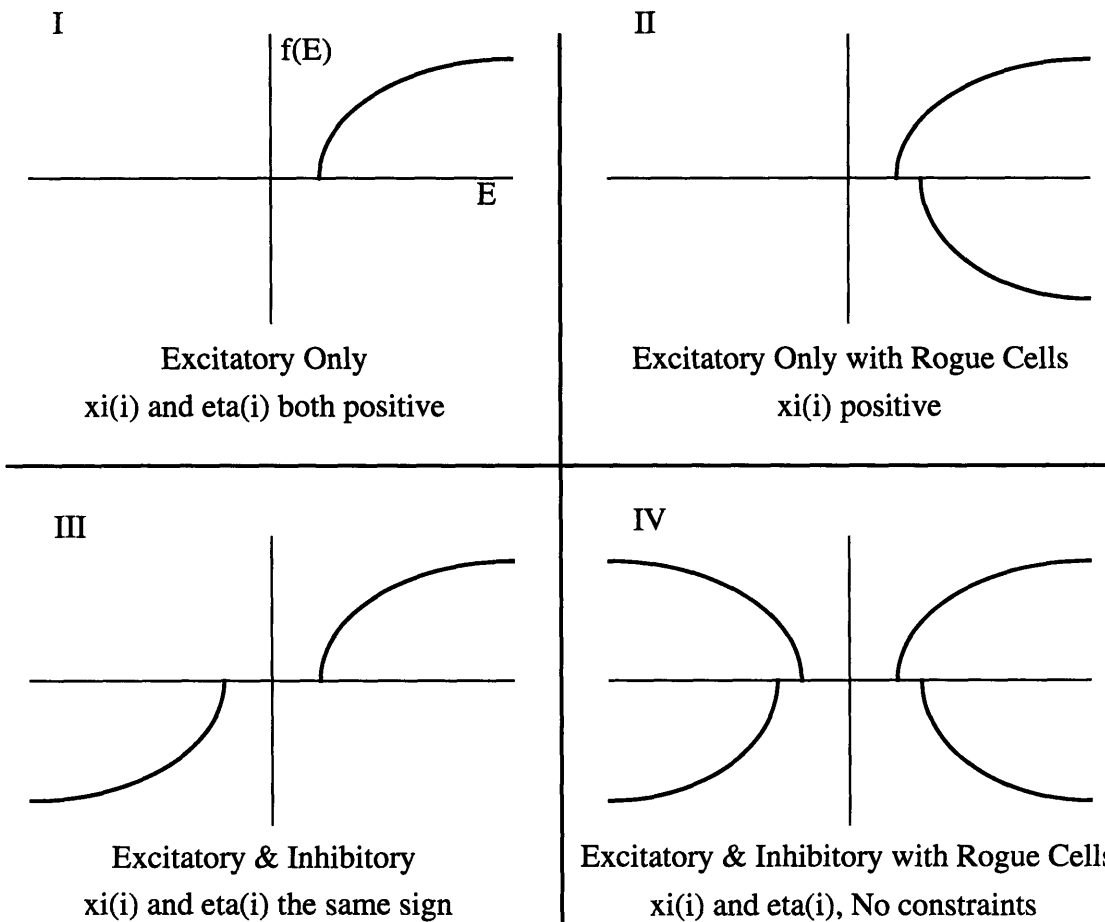


**Figure 4. Surface plot of a Lyapunov function describing the system dynamics. The energy of a state is plotted against the firing rates of the two member neurons in the network described in Figure 3. Note the major trough (low energy line) along the line  $V1+ V2= 100.15$ . Relaxation is fast from a point off the line into the trough. The**

**Lyapunov function used is  $H = -0.5\check{v}W\check{v} + \sum_i^{v_i} \int_0^{v_i} g^{-1}(s)ds - \check{h}\check{v}$ , where  $v_i = g(u_i)$ . The firing rates are  $\check{v}$ , and  $g^{-1}$  is the inverse of the frequency current relation function.**



**Figure 5. Terrain along the base of the valley in the energy diagram in Figure 4. Note the three energy minima (asterisks) at  $(V1,V2) = (0.0, 100.15)$ ,  $(50.07, 50.07)$ , and  $(100.15, 0.0)$ . The line attractor is not perfectly flat, but rather consists of a few fixed points with very slow relaxation times along the line. It is important to note that the energy differences along the line, shown here, are two orders of magnitude smaller than the energy differences of the steep trough walls shown in Figure 4.**



**Figure 6. Characteristic response curves of neurons and corresponding mathematical constraints in four network types examined. See text for detailed explanations of each network type. Each graph is a plot of  $E$  vs.  $f(E) = \sum_j \eta_j g(\xi_j E - \xi_j \theta_j + 2.187)$ .**

## IV. Network Types

We classify four network types according to biological attributes and examine their construction. We then evaluate the quality and robustness of the various network types and compare their properties.

### Network constraints

Before constructing a network, it is imperative to consider all biological constraints and assumptions underlying the system being modeled. It is important to first determine the excitatory and/or inhibitory attributes of the connections between neurons. Excitation occurs when the presynaptic neuron's direction of excitement matches that of the postsynaptic neuron; namely, when the first neuron is excited, the system will be driven in a direction that favors excitation of the second.

Mathematically, according to our rank-one assumptions, the weight matrix  $W$  is the outer product of the two vectors of  $\vec{\eta}$  and  $\vec{\xi}$ . Therefore, the sign of each weight  $W_{ij}$  is determined by the similarity of the signs of the specific elements of  $\vec{\eta}$  and  $\vec{\xi}$  that combined to form it. The direction in which a presynaptic neuron  $j$  drives the system is determined by  $\eta_j$ . The orientation of postsynaptic neuron  $i$  (i.e. its ON direction) is determined by the sign of  $\xi_i$ . (This is because  $\xi_i$  is the coefficient of  $E$  inside the  $g$  function.)

Therefore, as long as the direction in which the firing presynaptic neuron  $j$  drives the system corresponds with that of the postsynaptic neuron  $i$ , an excitatory connection is formed. Mathematically, as long as  $\xi_i$  and  $\eta_j$ , are of the same sign,  $W_{ij}$  is positive resulting in excitation in the dynamics described in EQ 1.

Another issue is the presence of rogue cells in the system. A rogue cell is defined as a cell that when injected with current, causes the eye to move in its OFF direction. -- i.e. even though it fires faster as the system moves in one direction, its own firing causes the system to move in the opposite direction.

The presence of rogue cells depends on whether or not the signs of  $\eta_i$  and  $\xi_i$  are similar for any neuron  $i$ . Self-weights ( $W_{ii}$ ) with negative signs yield rogue cells, whereas self-weights with positive signs yield normal cells.

Figure 6 shows the characteristic neuron response curves allowed by each of four network types that we examined.

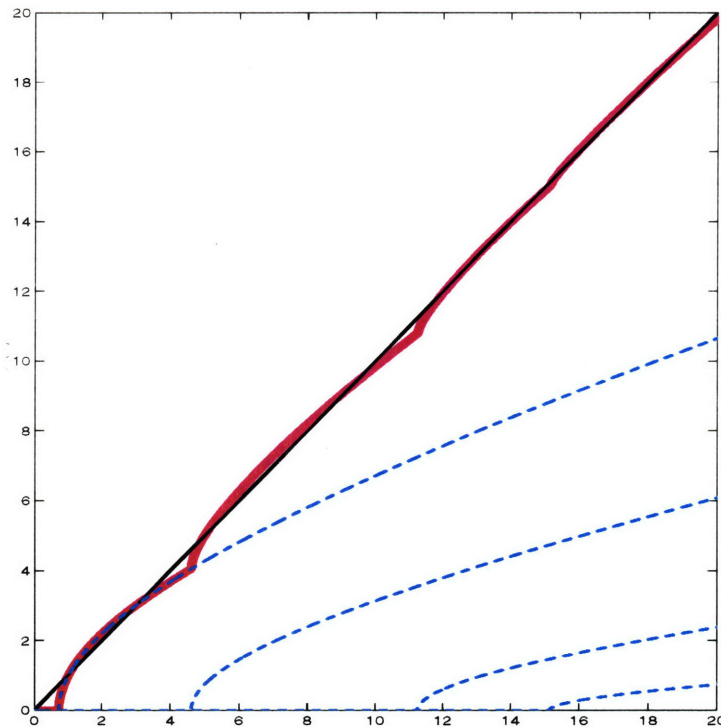
Type I allows for excitatory cells only. All the cells have the same orientation, and drive the system in the same direction, thus exciting each other. Type II is similar to Type I in that all the cells have the same orientation, but unlike Type I, rogue cells are allowed. Here, the cells depicted in the upper right quadrant excite all the neurons in the system, whereas the rogue cells depicted in the lower right quadrant inhibit all the neurons in the system.

In a Type III network, the equivalents of two type I networks with opposing orientations are combined. All the cells within each quadrant excite themselves and inhibit those in the other quadrant. Type IV allows for both inhibitory and excitatory connections between neurons, but also allows for rogue cells.

Types I and II obey Dale's law, which states that neurons can be either excitatory or inhibitory to other neurons in the network, but not both. In networks of type III and IV, cells that have one effect on cells of one orientation, have the opposite effect on cells of another orientation, and therefore do not obey Dale's law.

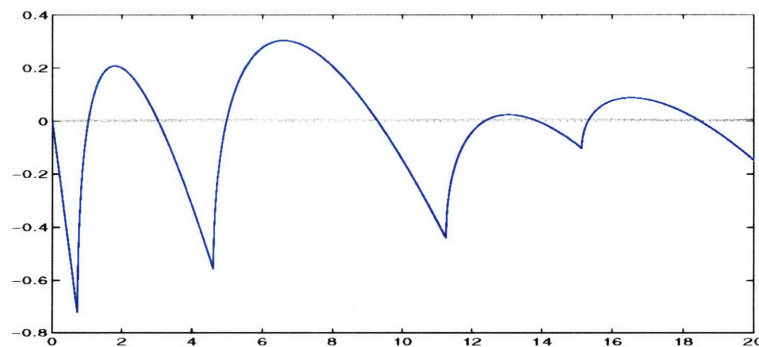
### **Type 1: All excitatory connections**

Baker [personal communication] reports that only excitatory neurons are found in the goldfish integrator, since immunochemical studies failed to detect any inhibitory neurotransmitters in the region believed to perform the integrator function. According to this assumption let us examine the make up of a network consisting only of excitatory neurons.



**Figure 7.** Analysis of the fit to  $f(E) = E$ , [thin line], using four excitatory neurons, [dashed lines]. The thick line represents the sum of the contributions by the four neurons in the network to the expression  $f(E) = \sum_j \eta_j g(\xi_j E - \xi_j \theta_j + 2.187)$ . The plots are of  $E$  vs.  $f(E)$ .

Mathematically,  $f(E)$  is a sum of  $j$  terms describing the contributions of the  $j$  neurons in the network. The line  $f(E) = E$  slopes straight up through the 2-d plot in Figure 7. The goal in constructing a network is to have the combined contributions of the various neurons add up as close as possible to the line  $f(E) = E$ , thus minimizing  $f(E) - E$ . With excitatory connections only, the fit of  $f(E)$  to  $E$  depends on using the neurons with higher thresholds to compensate for the roll-off of the neurons with lower thresholds.



**Figure 8.** The difference  $f(E) - E$  derived from the network in figure.

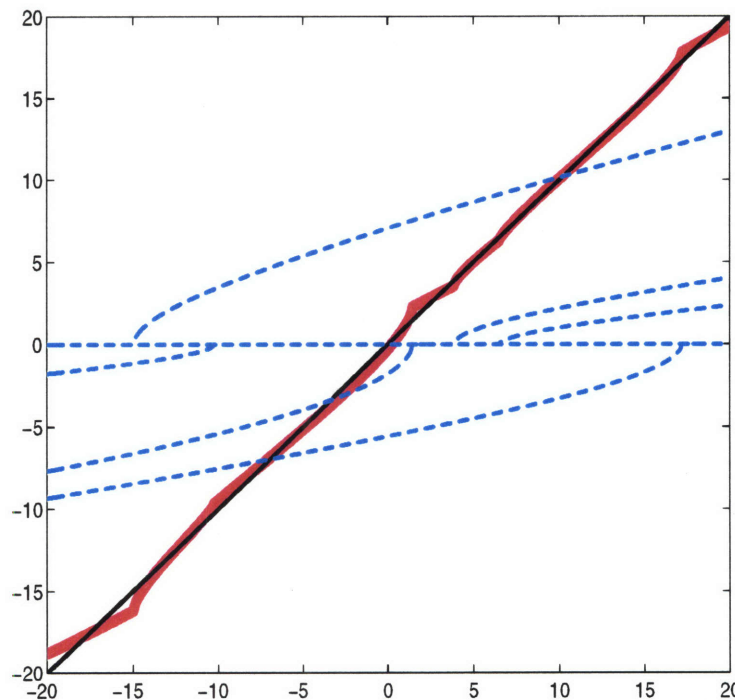
Figure 8 shows the resulting  $E$ -space dynamics from the network in Figure 7. As before, the stable fixed points are at the negative sloped zero crossings of the plot.

Considering the additivity of excitatory effects, the closer to linear the  $f/i$  characteristic is, the fewer neurons are required to construct the network; if the  $f/i$  curve were in fact more linear, the neurons with lower thresholds have less rolloff and require less compensation by addition of neurons with higher thresholds. If more high threshold neurons are added to the system, they will, in turn, have smaller weights assigned to them.

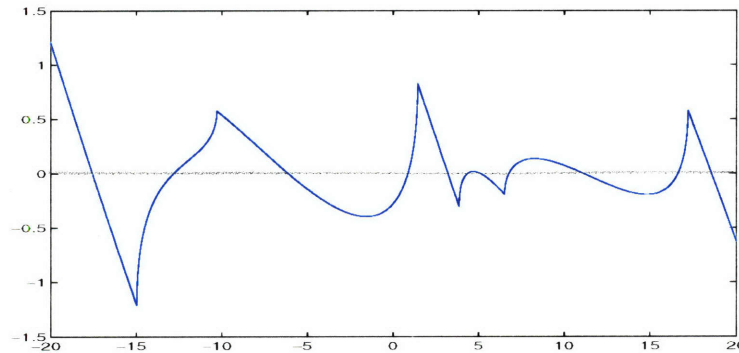
In goldfish, the feed-forward inputs from the other parts of the brain to the integrator network have been experimentally observed to be all-excitatory. This lowers the threshold of the neurons, shifting them to the left in  $E$  space. With positive feed-forward inputs, we predict that the neurons must have high enough intrinsic thresholds so that they can be lowered by the feed forward inputs and yet still lie in positive  $E$ -space in order to properly construct the fit  $F(E) = E$  in the all excitatory case. Said differently, any network with all excitatory connections must have all its neurons silent in the absence of synaptic inputs. This is not the case with neurons in mammalian integrators, which appear to possess spontaneous pacemaker activity [22].

### Type 2: Excitatory connections with rogue cells

In discussions with Baker, he claims he does not see any rogue cells in the areas associated with the integrator. The construction of Type 2 networks is similar to those of Type 1, except that neurons with negative  $\eta$  values are also allowed. The rogue cells allow for a less structured fit, and this has a bearing on the robustness of the networks. See the discussion under the section, Evaluation of networks.



**Figure 9.** Analysis of the fit to  $f(E) = E$ , [thin line], using six excitatory and inhibitory neurons, [dashed lines] with no rogue cells. The thick line represents the sum of the contributions by the six neurons in the network to  $f(E)$ .



**Figure 10.** The difference  $f(E) - E$  derived from the network in Figure 9.

### **Type 3: Excitatory and inhibitory connections**

In networks with both excitatory and inhibitory connections, the neurons form two groups with opposing orientations. The neurons in each group excite the neurons in their own group and inhibit the members of the other group. Figure 9 shows a fit constructed with excitatory and inhibitory connections. Note that the inhibition allows for more flexibility in constructing the fit. Neurons can be positioned to cancel each other by mutual inhibition, whereas in the all-excitatory case, neurons could only be added to counteract the roll-off of earlier neurons. This means that additional neurons can be added freely to the network, without need to attenuate their weights. For example, mutually inhibitory neurons can be added in pairs to produce wiggles in the  $E$ -space dynamics.

Figure 10 shows the fit attained by the network in Figure 9. Note that fixed points can occur where two mutually inhibitory neurons are placed close by each other. This can be done ad infinitum in order to add more fixed points to the system. In fact, we found that given ideal placement (pairing up) of  $N$  neurons, a network could have a total of  $2N+1$  fixed points,  $N+1$  of which are stable. We report this as the upper limit of the number of stable fixed points per number of neurons in the network.

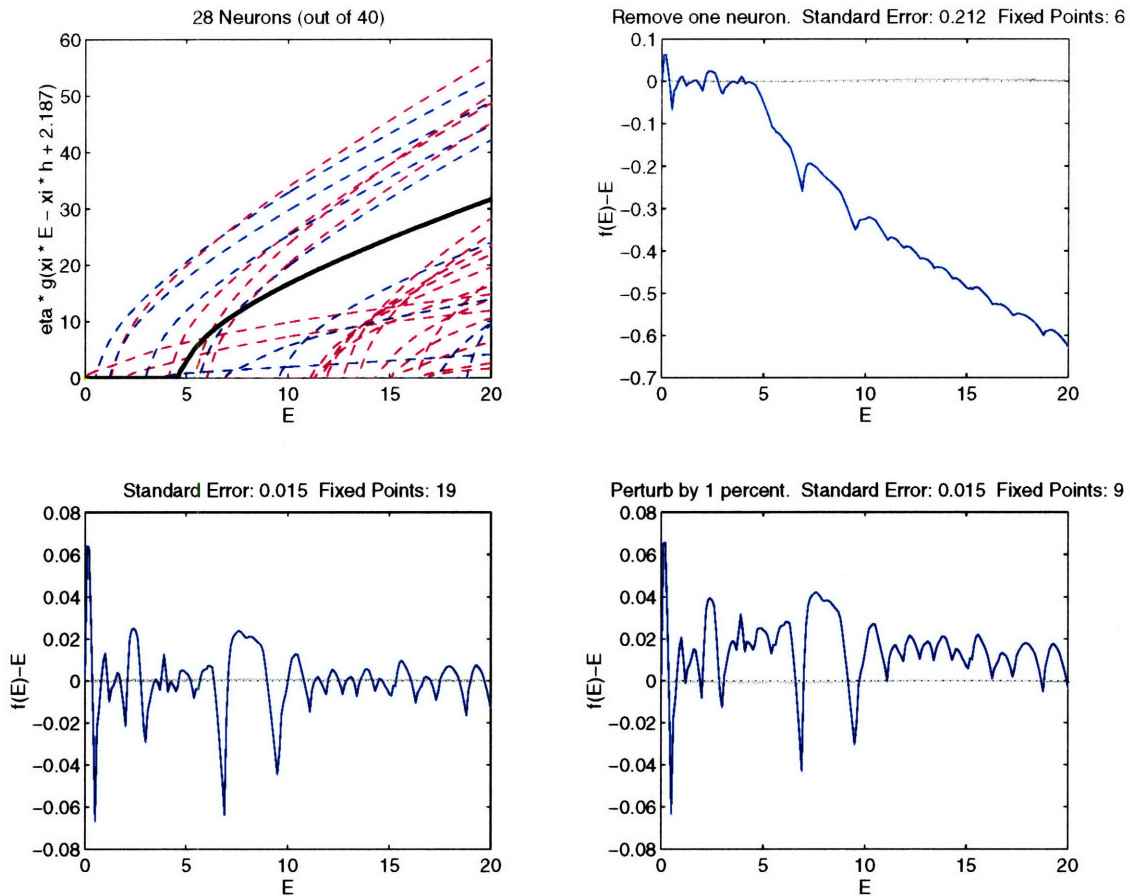
### **Type 4: Both excitatory and inhibitory connections plus rouge cells**

See discussion below under Evaluation of networks.

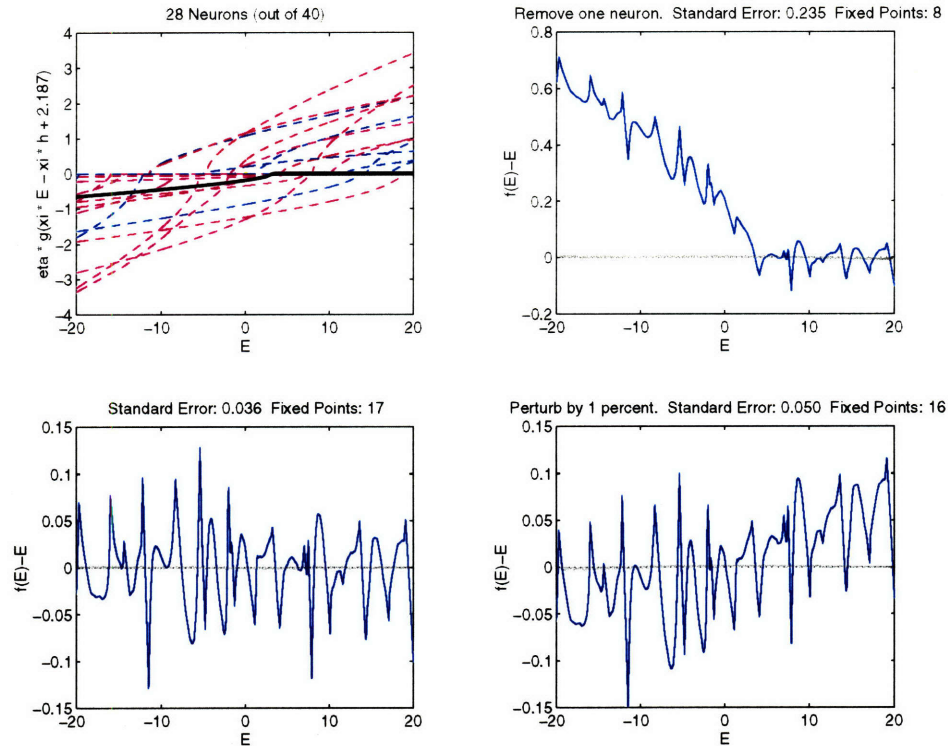
### **Evaluation of networks**

We compare different network configurations for robustness and efficiency and evaluate the effect of various parameters on network properties.



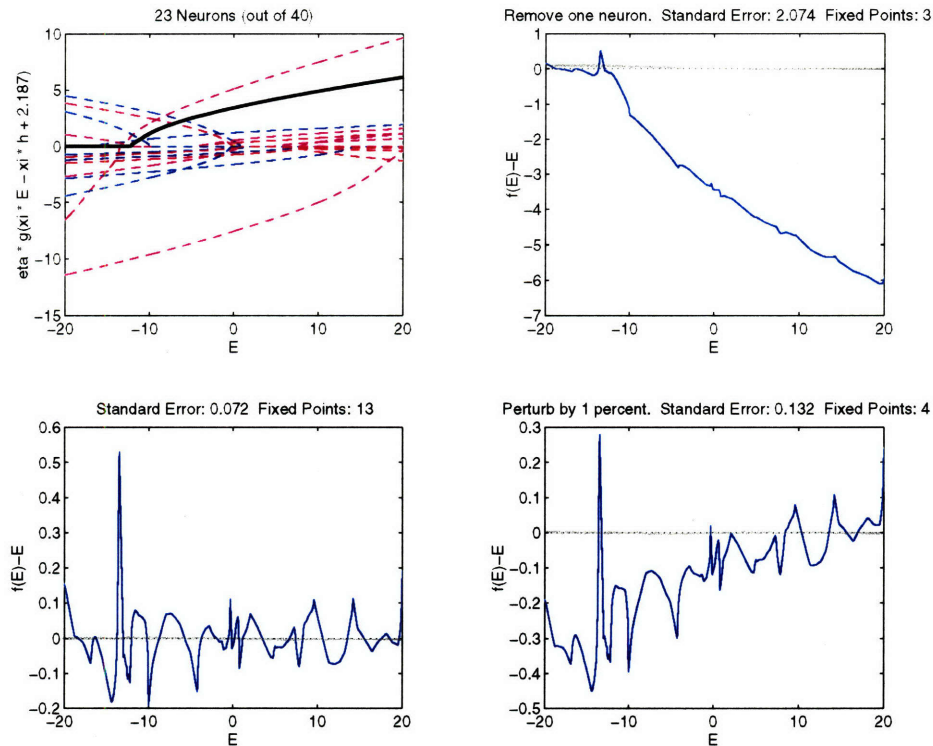


**Figure 11. An example of lesion and robustness testing. This all excitatory network (Type I) was constructed by a non negative least squares minimization on an original set of parameters. The top left plot shows the contributions of the various neurons to the network. The bottom left plot shows the energy diagram of the network. This network was subjected to a lesion test in which one neuron (drawn as a thick black line) was removed. The energy diagram (top right plot) reveals that the network was affected drastically to the right of the neuron's threshold, but not at all to its left. Note that many fixed points were lost, and the overall error was significantly degraded. In addition, the original unlesioned network was also subjected to a 1% gaussian perturbation of the synaptic weights (lower right plot). The energy diagram reveals that a few fixed points were lost, and that the overall error was degraded slightly.**



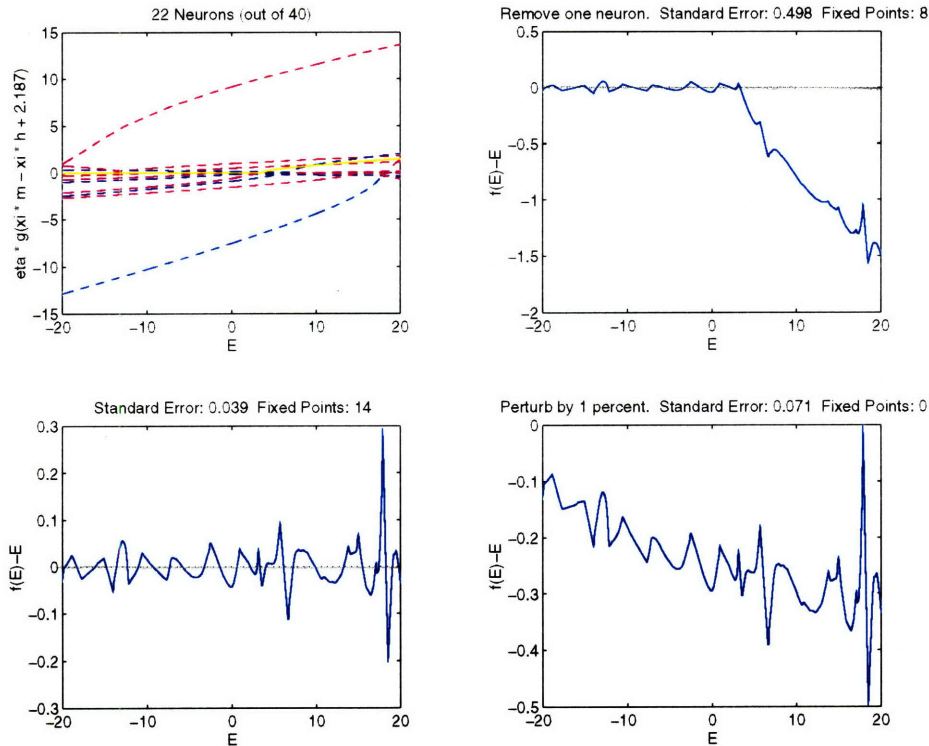
**Figure 12. Robustness testing of a network with excitatory and inhibitory connections and no rogue cells (Type III).** As before, removing a neuron causes a drastic change over the region of activity of that neuron (see text). In this network, however, the perturbation of the weights does not seem to have had a major effect on the network, as all but one fixed points remain intact and the degradation of error was minor.

To test the various network types for robustness to neuron damage, each network was subjected to a lesion test. A neuron was picked at random and removed from the network, leaving the rest of the network untouched. In most cases, where a reasonably tight fit with many wiggles had been achieved, this removal had a drastic effect on the network. Each neuron contributes to  $f(E)$  for all values of  $E$  where the neuron is above its threshold. Removing the neuron caused everything in the neuron's ON direction, beginning at its threshold, to be shifted up or down. This usually meant that all fixed points in a neuron's ON domain, beginning at its threshold, were destroyed when the neuron was removed. The overall error of the network also suffered significantly, depending mostly on how much of the affected region of state space overlapped with the region in which the minimization was being performed.



**Figure 13. Robustness testing of a network with excitatory and inhibitory connections, and rouge cells (Type IV). This network performed significantly worse than the non-rouge cell network in Figure 12. Here, almost all fixed points are destroyed by perturbation, even though the degradation of error is about the same. This suggests that the fixed points in this network are less robust to perturbation than those in Figure 12.**

In addition to lesion studies, robustness tests were performed on the networks. The weights in the network being tested were perturbed by a factor determined by a Gaussian spread with a mean of 1 per cent. After the perturbation, the networks were examined again for overall error and number of fixed points. The type III networks with excitatory and inhibitory connections fared quite well, retaining most of their fixed points and showing little degradation of error. (Figure 12). The worst hit were the type IV networks with rouge cells. (Figure 13) It seems that since the fit of these networks was constructed in a less orderly and structured fashion than type III networks, the even perturbation to the weights caused an uneven (unevenly-compensated-for) perturbation to the fit of the network.



**Figure 14. Robustness testing of Type III network. This network displays the observed behavior of incorporating two dominant ‘Uber-neurons’. The two prominent neurons in the upper left plot have large weights associated with them, and due to their symmetric placement, together form a good first order approximation to the line  $E = f(E)$ . The other neurons in the network act to fine tune the rough fit of the Uber-neurons. It is worthy to note that although close fits are attainable using this strategy, the network becomes very susceptible to catastrophic damage if one of the Uber neurons is affected.**

Sometimes the minimization yielded a solution where a few neurons would dominate by having much larger weights than the rest of the system. (Figure 14) These ‘Uber-neurons’ would usually be symmetrically placed about the center of the  $E$  region, and be of opposite orientation, together providing a first order fit to the straight line  $E = f(E)$ . The other neurons, which had much smaller weights, were in effect responsible for making second order corrections to the first order fit of the Uber-neurons, thus causing many oscillations and many fixed points, along with a good overall fit. The trade-off with these solutions is that the system becomes very vulnerable -- it would undergo a drastic change if one of the dominant neurons were affected.

### Networks without rank one constraint

So far we have examined only networks with a weight matrix of rank one. One cannot be sure that this mathematical constraint does not also place significant constraints on the biological functioning of the network. It is therefore necessary to explore networks of full rank as well.

One way to get around the problem of visualizing a full rank state space is to visualize only one line in the space that is of interest -- the line along which the fixed points lie. This makes it possible to perform a minimization on a network without the rank one constraint and then to plot the dynamics along the predicted line of fixed points.

In this case, though, one needs to check for spurious fixed points. In the presence of spurious fixed points, a system can saccade to and get stuck at a fixed point that is not on the intended line of fixed points. This can be tested for by running the simulations and checking for fixed points that don't appear on the line.

To construct a network without the rank one constraint, we start out by repeating EQ 1.

$$u_i + \dot{u}_i = \sum_j W_{ij}g(u_j) + h_i \quad (\text{EQ 10})$$

For convenience, we perform a change of variable at the onset:  $c_i \equiv u_i + h_i$  and  $\dot{c}_i = \dot{u}_i$ . So now EQ 10 can be rewritten as

$$\dot{c}_i + c_i = \sum_j W_{ij}g(c_j + h_j) \quad (\text{EQ 11})$$

Now we pick a line in the  $i$ -dimensional space over which to perform the minimization:

$$c_i = E\xi_i \quad (\text{EQ 12})$$

$E$  here is a parameter used to sweep along the line described by  $\vec{\xi}$ . So at equilibrium, plugging EQ 12 into EQ 11, we have:

$$E\xi_i = \sum_j W_{ij}g(E\xi_j + h_j) \equiv F(E) \quad (\text{EQ 13})$$

We now perform a non negative least squares minimization over this line, with  $E\xi_i - F(E)$  as the error function.

With simulations in the full rank case, we read the state of the system by using the complete vector of currents,  $\vec{u}$ . To check for spurious fixed points, we run the rate model and allow the system to settle to a fixed point after being perturbed. Since  $\vec{u}$  might be off the plane  $E\vec{\xi} + H\vec{h}$ , we define a vector  $L\vec{\zeta}$  to be the perpendicular distance to  $\vec{u}$  from  $E\vec{\xi} + H\vec{h}$ :

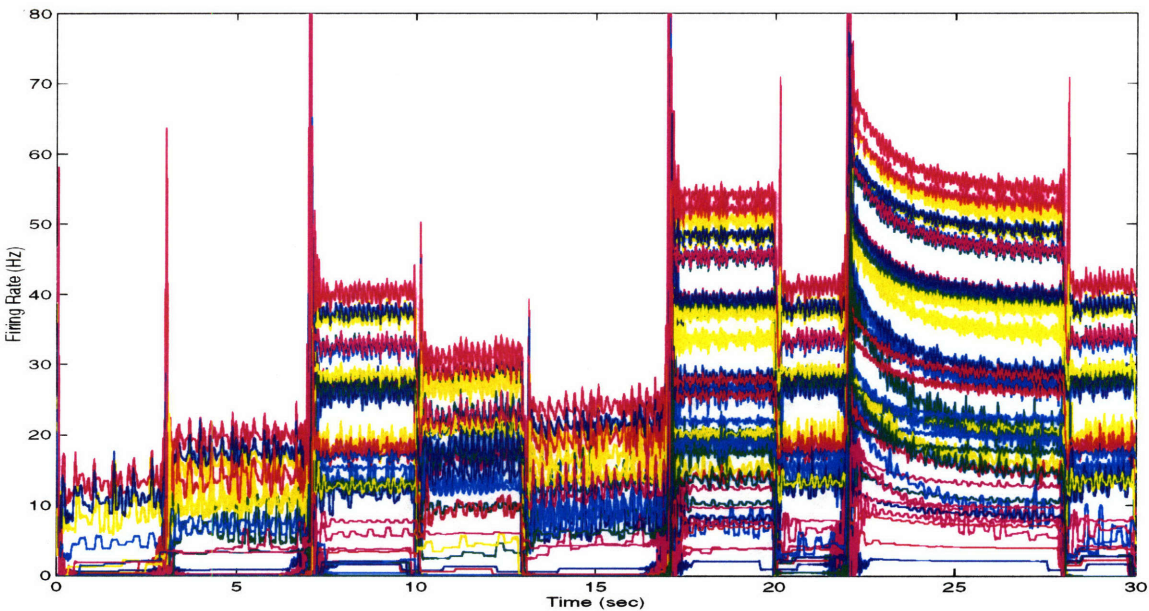
$$\vec{u} = E\vec{\xi} + H\vec{h} + L\vec{\zeta} \quad (\text{EQ 14})$$

After the system settles, we solve for both  $E$  and  $H$  by projecting  $\vec{u}$  onto the plane defined by  $\vec{\xi}$  and  $\vec{h}$ .

$$\xi^2 E + \xi h H = u \cdot \vec{\xi} \quad (\text{EQ 15})$$

$$\xi h E + h^2 H = u \cdot \vec{h} \quad (\text{EQ 16})$$

We then solve for  $L$  by plugging in to EQ 14.



**Figure 15. Firing rate traces of a 41 neuron network simulated using the spiking model. Brief 50 msec current pulses are injected into the system at 2 - 5 second intervals to produce the saccade-like shifts. After each pulse the system undergoes fast relaxation to a fixed point and remains there until the next stimulus. The major drift at 22 seconds is due to the system being stimulated above its highest fixed point, and to the long relaxation time needed to get to the closest fixed point.**

## V. Simulations Results

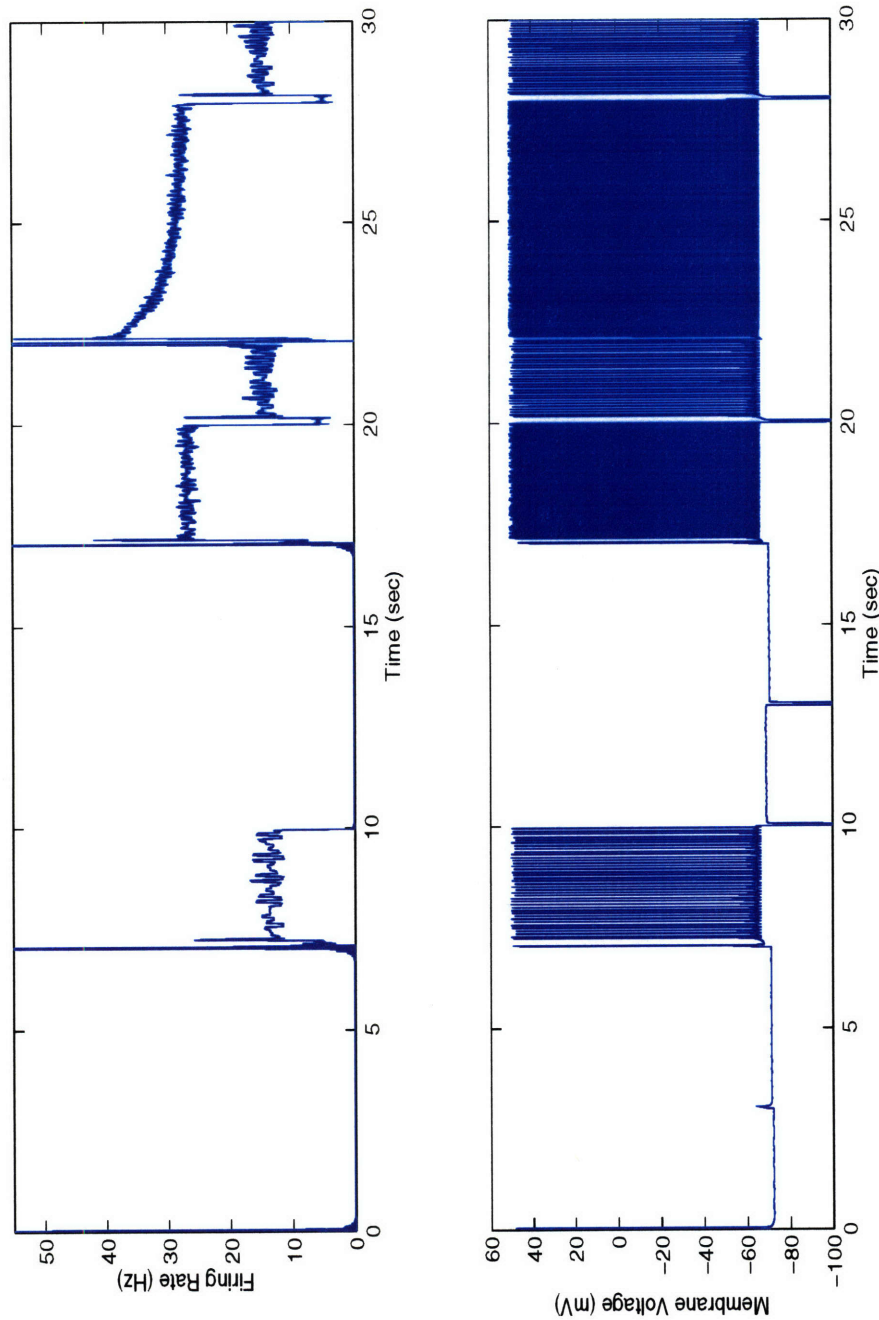
After the various networks were modeled and analyzed, simulations were run to directly test the properties of the networks.

### Spiking Simulation

A 41 neuron network subjected to saccadic stimulation over a 30 second period was simulated using the spiking model (Figure 15). The results show the firing rates approaching fixed points in state space, precisely as predicted. The network is capable of maintaining multiple persistent patterns of activity. The persistence times of the network are longer than the intrinsic time constant of the neurons, demonstrating that this can be done successfully with positive feedback, despite the intrinsic nonlinearities of the neurons.

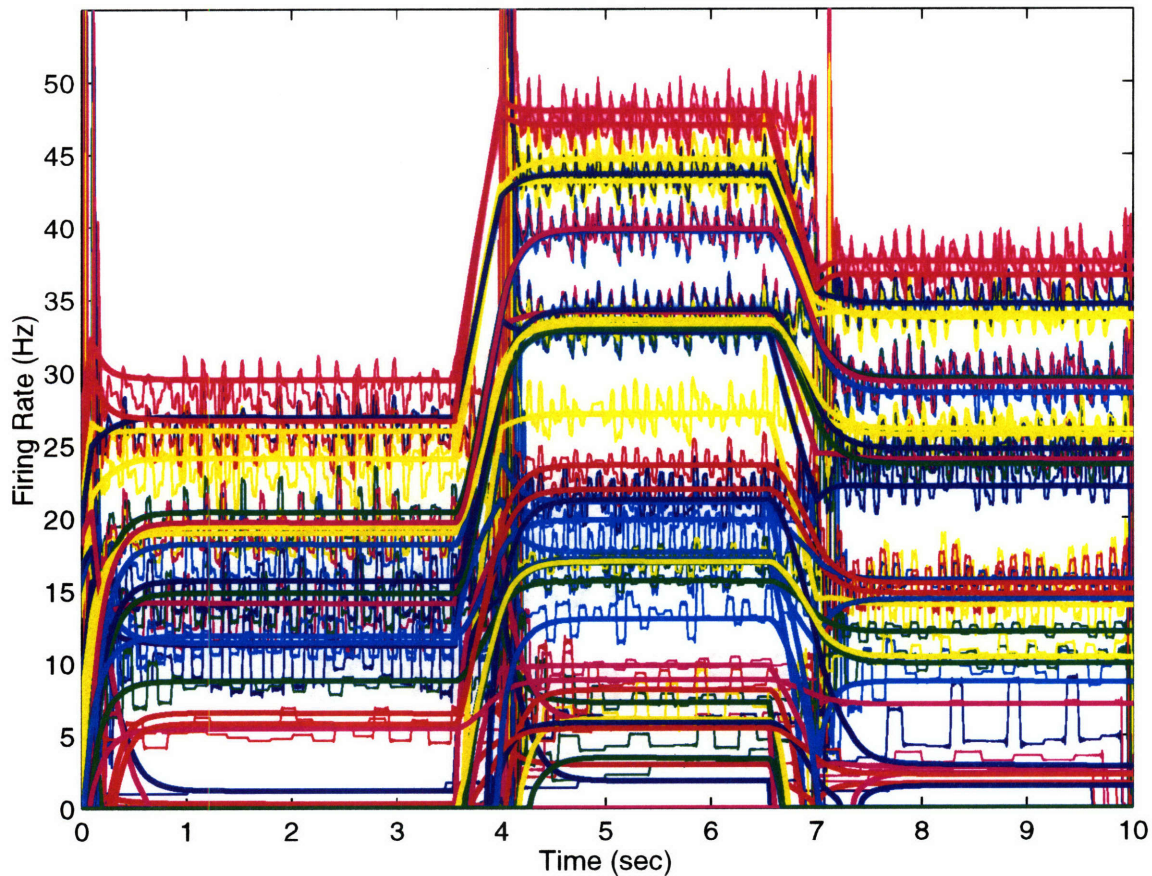
During each current injection, the firing rates of the neurons rise sharply, and then begin to relax to their new steady state level. This transient behavior is called spike frequency adaptation.

Late in the simulation, the system is stimulated to saccade above its highest stable fixed point. The dynamics directs the system to relax to the nearest stable fixed point. In this case it is far away, and so the relaxation time is long.



**Figure 16.** The first trace shows the firing rate of neuron #9 during the simulation of the 41 neuron network in Figure 15. The second trace shows the membrane voltage of neuron #9, also taken from the same simulation.

The membrane voltage of an individual neuron from the same simulation is plotted in Figure 16. Note that this neuron, like many others, goes silent for some intervals during the simulation. From a state space point of view, the neuron's threshold lies at a certain value of the state variable  $E$ . If the state of the system rises above that value, the neuron is activated at an appropriate firing rate; below that value, it remains quiescent.



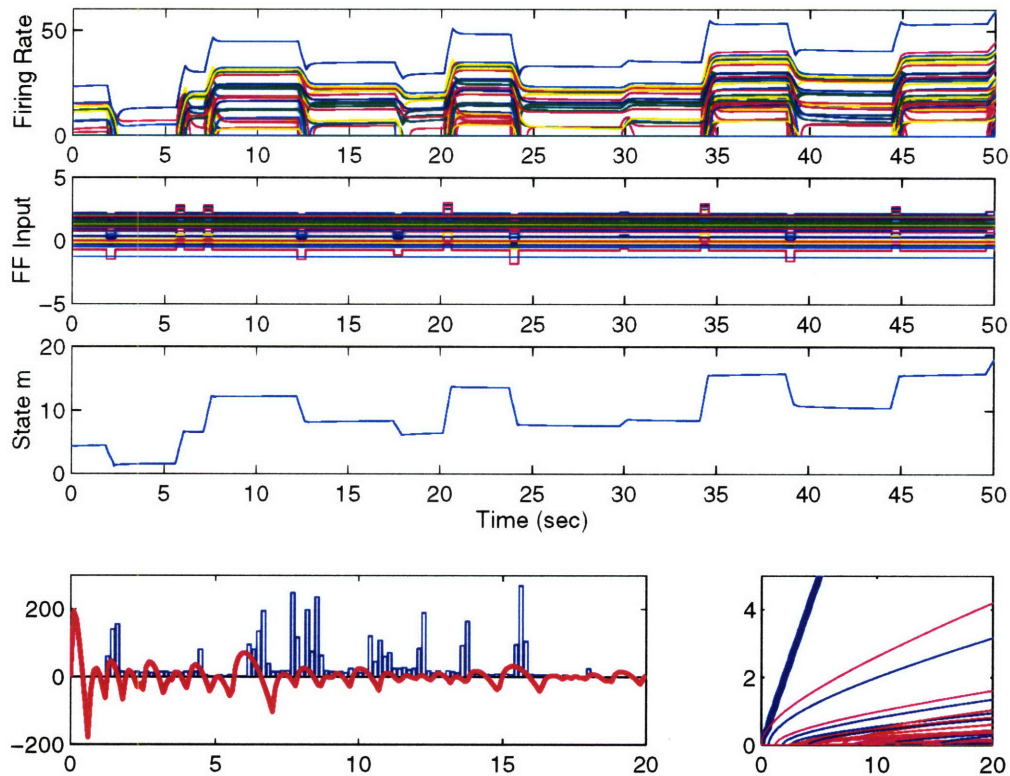
**Figure 17. Comparison of the spiking model with the rate model: A ten second simulation with three saccades was run using the spiking model. The same simulation was also run using the rate model (overlaid smooth lines). Note that the rate model does not incorporate many of the transient properties of the spiking model.**

### Comparison of rate and spiking models

Figure 17 depicts a ten second simulation involving three saccades. This simulation was run on both the spiking model and the rate model, and the results are plotted together for comparison. The results show that a rate model can be used successfully to predict steady state dynamics in the nonlinear case.

While the rate model is indeed sufficient for the purposes for which it was used, the spiking model reveals a host of intricacies not included in the rate model. In the rate model, we have modeled the nonlinearities of the neurons with respect to their steady state response to changes in current ( $f/i$  curves). However, the rate model does not incorporate the transient nonlinearities of the system in responding to change in input current (i.e. spike frequency adaptation transients)



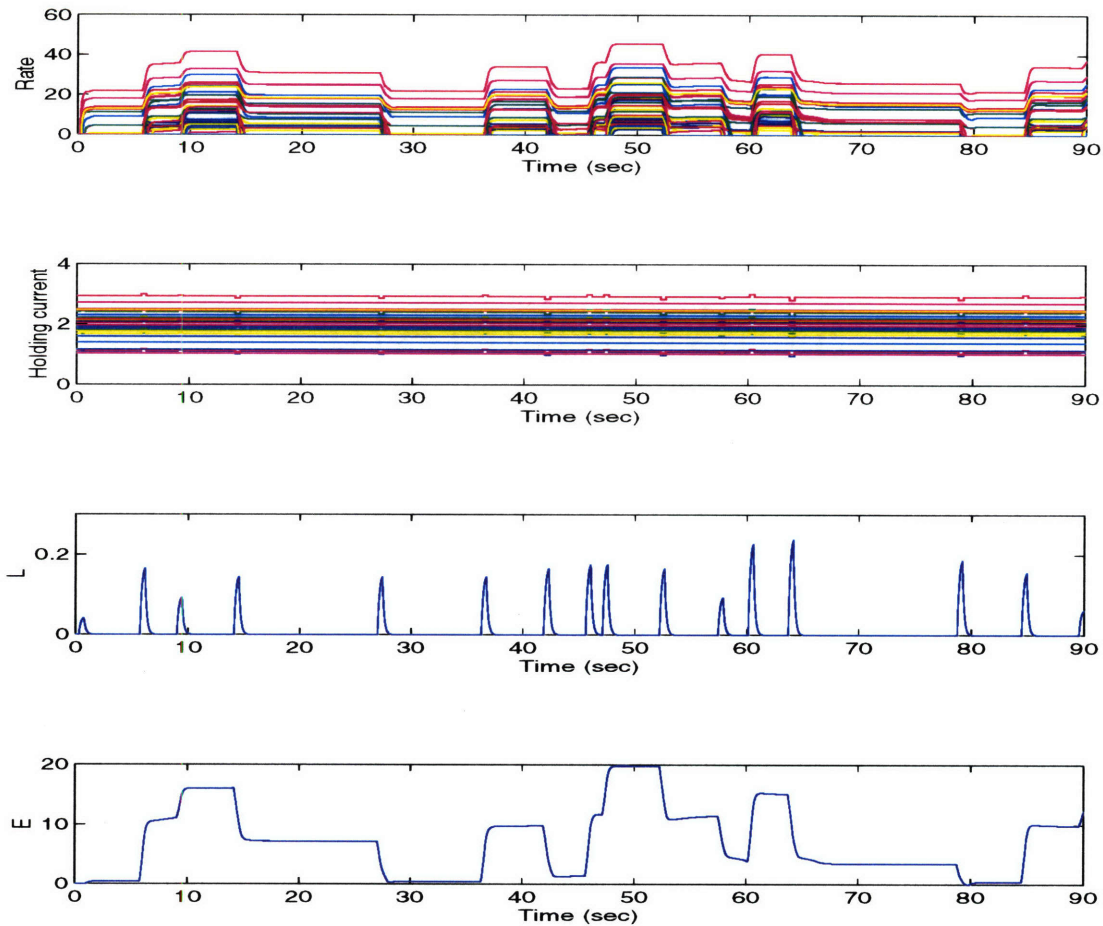


**Figure 18.** The top plot shows a rate-model simulation of an all excitatory 41 neuron network. The network was subjected to a series of external stimulus bursts representing the feed-forward inputs responsible for saccades (second plot). The firing rates of the neurons settle to fixed patterns of activity when saccading to the various fixed points of the network. The third plot shows changes in the state of the system,  $E$ , which can also be regarded as a measure of eye position. The bottom left plot shows a histogram of the values of  $E$  during the simulation, superimposed over the energy diagram for  $E$ . As expected, during most of the simulation, the system remains at or near the fixed points represented by the negative-slope zero crossings of the energy diagram. The bottom right plot shows the structure of the network, with the contribution of each of the 41 constituent neurons (thin lines) to the fit function  $f(E) = E$  (thick line).

### Fixed points and system dynamics

To gauge the correlation between the predicted fixed points and the actual fixed points, the rate model was run through 90 seconds of saccadic stimulation (Figure 18). In the rate model, it is possible to measure the state variable  $E$  directly. The time course of  $E$  reveals a correlation between the states of the system, and the various patterns of persistent activity seen in the firing rates.

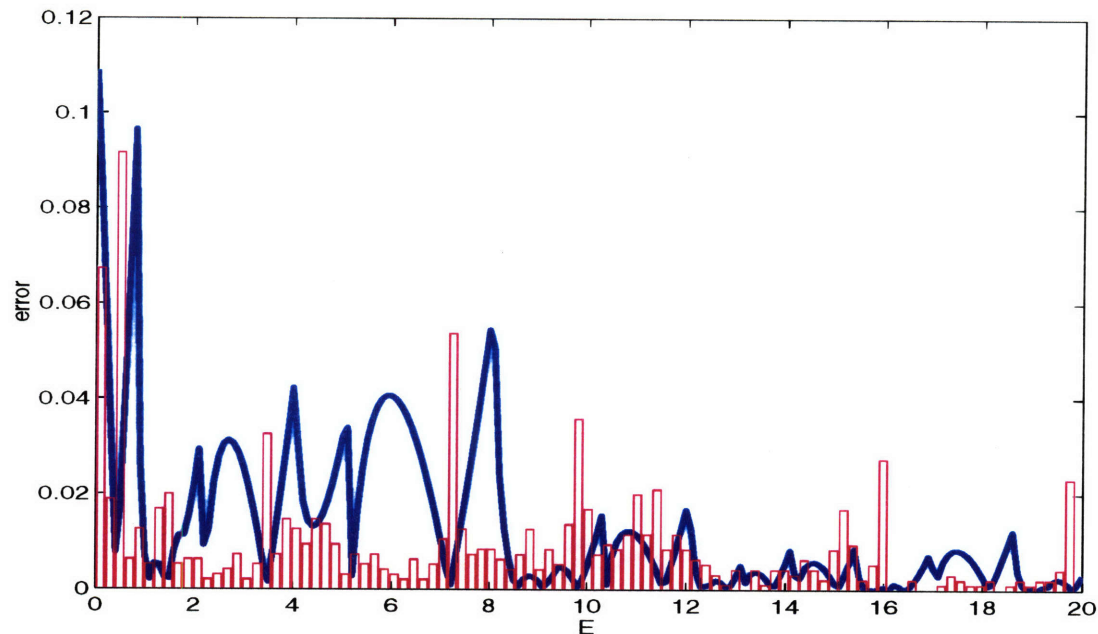
A histogram of the timecourse of  $E$  is plotted on top of the energy diagram of the network ( $F(E) - E$  vs.  $E$ ). The plot demonstrates that the system resides for the longest duration at points predicted by the energy diagram to be stable fixed points of the system.



**Figure 19.** A rate model simulation of a network with a full-rank weight matrix. The topmost figure shows the firing rates, and the second figure shows the feed forward stimuli to the system. Since the dynamics are not reducible to one variable, the error off the line attractor is given as the perpendicular distance  $L$  from the line to the current state of the system (third figure from top). The position along the line is given by  $E$ , which is the best one-variable representation of the state of the system (lower figure). Note that  $L$  is zero whenever the system seems to have reached a steady pattern of activity, implying that this fixed point lies along the line. Many simulations were run and no spurious fixed points (persistent patterns of activity coded for by points off the line) were found.

### Full rank runs

Based on the full rank model for weight matrix construction, a simulation was run to test this line attractor construction and to check for spurious fixed points. Figure 19 shows the rate model simulation along with the values  $L$  and  $E$  plotted over time.  $E$  is the parameter along the attractor line in  $N$  dimensional space, and serves as a good measure of eye position.  $L$  is the length of the perpendicular vector connecting the current state of the system to the line.



**Figure 20.** The thick dark line shows the error for states along the line  $E\xi_i$  plotted against  $E$ . A histogram of the  $E$  values from the simulation in Figure 19 is overlaid on this plot. Note that the peaks of the histogram occur precisely at zero points of the error generated when constructing the network.

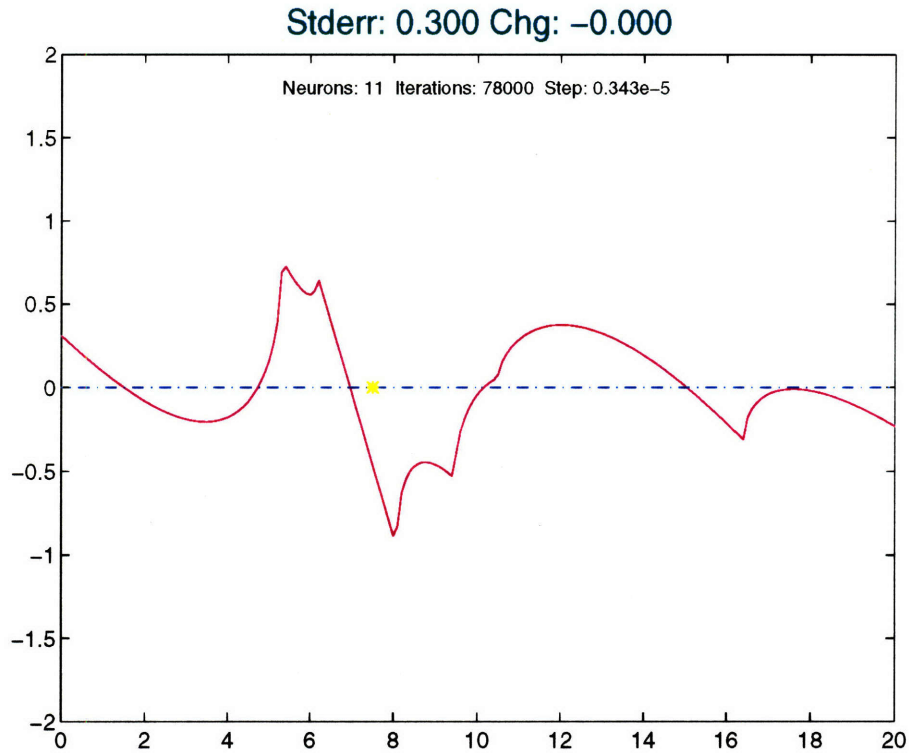
The time course of  $L$  reveals that the system strays off the line when perturbed by input pulses, but then undergoes fast relaxation back to the line and the nearest fixed point. Since  $L$  never settles at a value other than zero, it can be inferred that all the fixed points reached during this simulation are indeed on the line. The simulation does not seem to have settled at any spurious fixed points. Similar simulations were performed with different networks and saccade patterns, and no spurious fixed points were found.

Figure 20 plots a histogram of the timecourse of  $E$  on top of a plot of the stability of the system for states along the line. This figure is a dramatic demonstration of the extent to which our simulation results corresponded to the predicted behavior. The distinct tall peaks in the histogram lie directly in the middle of the energy wells along the line attractor of the network. Note that the larger energy wells are better able to stabilize the system and hold it at an appropriate fixed point. Accordingly, the tall histogram bars are positioned in between the tall error peaks.

## VI. Local learning rule

For a network to construct itself initially and adapt itself for purposes of plasticity a biologically feasible local learning rule is required. We analyze the rank one case.

Each of the neurons in a network has access only to limited information about the system as a whole. During the learning process the neurons need to take advantage of any information available to them in order to modify their own connection weights in a manner that will improve the network's overall performance. It stands to reason that



**Figure 21. A fit achieved by a local learning rule. The plot is of  $f(E) - E$  vs.  $E$ . The learning rule simulation allows each neuron to access information about the current state of the system (eye position) and about its own firing rate, and to accordingly tweak its own weights to improve the overall dynamics of the system.**

information regarding eye position is available to each neuron. Eye position may be considered in terms of a summary variable that reflects the overall state of the system. Indeed, the range of values of  $E$  can be said to correspond to the range of values of eye position. It is therefore reasonable to assume that each neuron has access to  $E$ , the state variable of the system.

The state of the system  $E$  changes as:

$$\dot{E} = \vec{\eta} g(\vec{\xi}E + \vec{\xi}h) - E \quad (\text{EQ 17})$$

As the system moves through phase space, the  $h$ 's and  $\eta$ 's for each neuron are updated based on the current value of  $E$  and their own firing rate, as follows:

$$\dot{\eta}_i = -s\dot{E}g(\xi_i E + \xi_i h) \quad (\text{EQ 18})$$

$$\dot{h}_i = -s\dot{E}\eta_i \xi_i g'(\xi_i E + \xi_i h) \quad (\text{EQ 19})$$

The  $\xi$ 's remain fixed.  $s$  is the modification step size which decays over time to allow the solution to converge, and  $g'()$  is the derivative of the  $f/i$  relationship  $g()$ .

The intuitive reasoning behind these relations is straightforward. The rate of change of each neuron's parameters is determined by how far the system is from stability ( $\dot{E}$ ), and by the

extent of that neuron's contribution to the current unstable state (the extent is reflected in the neuron's firing rate).

At first, the learning rule simulation utilized an on-line minimization, in which the system evolved according to its dynamics, saccading to random points every few seconds. In this simulation, the system ended up avoiding the regions in state space that needed the most adjustment -- namely, those regions that could be transformed into additional stable fixed points. Instead, the system spent most of the time drifting to and redundantly perfecting stable fixed points that had already been minimized. This behavior introduces an inherent difficulty in the understanding of the learning problem -- how can learning be accomplished effectively while the system is evolving according to initially primitive system dynamics? This problem may be mitigated by assuming that all stable fixed points are present from the start, and require only minor adjustments.

The same learning simulation was run with a random traversal of the state space, instead of evolving according to system dynamics. It yielded better results (Figure 21) that compare reasonably well with the global minimizations discussed above.

## VII. Conclusions

We have shown that a network with stable states approximating a line attractor can be constructed from nonlinear neurons. Our work suggests that such a network would have stable fixed points a long a line in its state space, with slow relaxation along the line. In addition, we have demonstrated that a local learning rule can be used to configure such a network at the individual neuron level.

From the network architectures we examined, we report that an integrator network with  $N$  neurons can support a maximum of  $N+1$  stable fixed points. Our work also suggests that neurons in an all excitatory integrator network would be silent in the absence of external input.

It is reasonable to assume that a network in which a series of fixed points combine to approximate a line attractor would be more robust in resisting the effects of noise than an ideal line attractor. Since in an ideal line attractor the line is perfectly flat and no position along it is more stable than its neighbors, even the slightest noise perturbations would kick the system's state freely along the line. Therefore, once this system has relaxed to the line, gaze stability must be assumed to be poor in the presence of noise. Conversely, in the fixed point approximate line model, even though a small amount of noise would perturb the state of the network slightly off the fixed point, the dynamics of transitions along the unsmooth line would then bring it back to the original point of gaze.

We note that the approximate line model of the VOR, although presumed to be more robust in the presence of noise, possesses only a finite repertoire of stable gaze positions, whereas a perfect line attractor allows infinite resolution. This implies that robustness is gained at the expense of resolution. While measurements are still needed to test for fixed point structure, we speculate that enough fixed points could provide adequate position resolution for the system, (provided, of course, that enough neurons are available in the network), while also providing gaze stability in the presence of noise.

In this project we have successfully modeled a system with dynamics that approximate a

line attractor. In order to construct a model of a complete integrator, one would need to interface both the feed-forward inputs and the readouts to and from each of the individual neurons. One would then need to balance the overall gain of the system in a manner that would provide a uniform linear response. Furthermore, the transient nonlinearities of the system in responding to change in input current (i.e. spike frequency adaptation) are crucial for balanced input/output and linear response, and would therefore need to be incorporated into the rate model used for network analysis.

### **Future experiments, recordings from the goldfish integrator**

One may test for fixed points by trying to discern quantized positions of saccades using a scleral search coil method for accurate eye position measurements. Additionally, by removing one or more neurons from the network, one could study the effects on integrator function and on fixed point structure. (i.e. does the system lose all fixed points higher than threshold of removed neuron?) Possible pharmacological treatments could also correlate to a predictable change in the model. (i.e. perturb the synapse weights with drug and test the robustness of the fixed points). One could, in principle, even test the local learning rule directly by comparing the changes in synaptic strength over a period of plasticity with those predicted from the measured recordings of firing rates and eye positions, in accordance with the learning rule.

One potential problem with experimental observation is that the more isolated fixed points have slower relaxation times associated with them, making them harder to pinpoint. The fixed points with faster relaxation times are correspondingly closer together, and therefore more difficult to differentiate. (lower right trace in Figure 18).

## **References**

- [1] D. A. Robinson. Integrating with Neurons. *Ann Rev Neurosci.*, 12:33-45, 1989.
- [2] J. L. McFarland and A.F. Fuchs. Discharge patterns in nucleus prepositus hypoglossi and adjacent medial vestibular nucleus during horizontal eye movement in behaving macaques. *J. Neurophysiol.*, 68:319-332, 1992.
- [3] J. M. Lopez-Barneo, C. Darlot, A Berthoz, and R. Baker. Neuronal activity in prepositus nucleus correlated with eye movement in the alert cat. *J. Neurophysiol.*, 47:329-352, 1982.
- [4] A. M. Pastor, R. R. De La Cruz, and R. Baker. Eye position and eye velocity integrators reside in separate brainstem nuclei. *Proc. Natl. Acad. Sci. USA*, 91:807-811, 1994.
- [5] C. Tiliket, M. Shelhamer, D. Roberts, and D. S. Zee. Short-term vestibuloocular reflex adaptation in humans. I. Effect on the ocular velocity-to-position integrator. *Exp. Brain Res.*, 100:316-327, 1994.
- [6] J. J. Hopfield and D. W. Tank. Computing with neural circuits: a model. *Science*, 233:625-633, 1986.
- [7] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc Natl. Acad. Sci USA*, 79:2554-2558, 1982.
- [8] D. Amit. *Modeling brain function*. Cambridge University Press, Cambridge, 1989.
- [9] H. S. Seung, How the brain keeps the eyes still. Submitted to PNAS, 1996.

- [10] Cannon S. C., Robinson D. A., Shamma S. A proposed neural network for the integrator of the oculomotor system. *Biol Cybern* 49:127-136, 1983.
- [11] Cannon S. C., Robinson D. A. An improved neural network model for the neural integrator of the oculomotor system: More realistic neuron behavior. *Biol Cybern* 53:93-108, 1985.
- [12] A. K. Moschovakis and S. M. Highstein. The anatomy and physiology of primate neurons that control rapid eye movements. *Annu. Rev. Neurosci.*, 17, 1994.
- [13] D. B. Arnold and D. A. Robinson. A learning network model of the neural integrator of the oculomotor system. *Biol. Cybern.*, 64:447-454, 1991.
- [14] D. B. Arnold and D. A. Robinson. A neural network model of the vestibulo-ocular reflex using a local synaptic learning rule. *Phil. Trans. R. Soc. of London*, B337:327-330, 1992.
- [15] Arnold D.B., Robinson D.A. A neural network model of learning in the vestibuloocular reflex (in progress)
- [16] Hansel D., Sompolinsky H. Chaos and synchrony in a model of a hypercolumn in visual cortex (in progress)
- [17] M. Serafin, C. de Waele, A. Khateb, P. P. Vidal, and M. Muhlethaler. Medial vestibular nucleus in the guinea pig. I. Intrinsic membrane Properties in brainstem slices. *Exp. Brain Res.*, 84:417-425, 1991.
- [18] A. R. Johnston, N. K. MacLeod, and M. B. Dutia. Ionic conductances contributing to spike repolarization and after-potentials in rat medial vestibular nucleus neurones. *J Physiol.*, 481.1:61-77, 1994.
- [19] S. du Lac and S. G. Lisberger. Membrane and firing properties of avian medial vestibular nucleus neurones in vitro. *J Comp. Physiol.*, A176:641-651,1995.
- [20] Quadroni R., Knopfel T. Compartmental models of type A and type B guinea pig medial vestibular neurons. *Journal of Neurophysiology* 72,4:1911-1924, 1994.
- [21] G.A. Kinney, B. W. Peterson, and N. T. Slater. The synaptic activation of N-methyl-D-aspartate receptors in the rat medial vestibular nucleus. *J. Neurophysiol.*, 72:1588-1595, 1994.
- [22] Y. Lin and D. O. Carpenter. Medial vestibular neurons are endogeneous pacemakers whose discharge is modulated by neurotransmitters. *Cellular and Molecular Neurobiology*, 13:601-613, 1993.

## A. Technical Appendix

### Simulation Capabilities

The simulation software written for this project is capable of loading in a network consisting of weights and holding currents, and running a simulation of a desired length with a specified pattern of external stimulation. It records membrane voltages of each of the neurons, as well as firing rates, synaptic currents, and overall currents.

### Channels

The channel equations for the neurons were taken from [Sompolinsky & Hansel]. The model has five channels plus a leakage current. They are:

*I<sub>na</sub>* -- Sodium current

*I<sub>k</sub>* -- Delayed rectifier potassium current

*I<sub>a</sub>* -- A-type potassium current

*I<sub>nap</sub>* -- Persistent sodium current

*I<sub>z</sub>* -- Slow Potassium current

*I<sub>l</sub>* -- leakage current

*I<sub>l</sub>* is independent of the membrane voltage  $V_m$ , and determines the passive properties of the cell near resting potential. The rest of the channels are dependent on  $V_m$ . *I<sub>na</sub>* and *I<sub>k</sub>* form the typical Hodgkin Huxley delayed rectifier Na/K combination of channels. *I<sub>nap</sub>* enhances the excitability of the cells near threshold, extending the dynamic range of the neurons to frequencies close to zero. *I<sub>a</sub>* reduces the gain of the neurons, limiting their maximal rate, while *I<sub>z</sub>* reacts slowly to input current steps and contributes to the frequency adaptation behavior of the cell. The total current coming into the cell was measured as:

$$I_{tot} = I_{syn} + I_{ext} + I_{channels}$$

As in [Sompolinsky & Hansel], currents are measured throughout in units of current density, (milliamps/cm<sup>2</sup>). Therefore, the size of the cell is not relevant.

### Synapses and networks

The weights generated by the minimizations of the rate model had to be converted in order to be used in the spiking simulations. This entailed converting the values in  $W_{ij} = \xi_i \eta_j$  into current amplitudes of responses between neurons. The time course of the synaptic currents  $I_{ij}$  in the spiking model was defined as:

$$I_{ij} = I_{ij}^0 \left( e^{-\frac{t}{\tau_1}} - e^{-\frac{t}{\tau_2}} \right) \quad (\text{EQ 20})$$

where  $\tau_1 = 20ms$  and  $\tau_2 = 150ms$ .  $I_{ij}^0$  is the amplitude of the response in neuron  $i$  whenever neuron  $j$  fires. Integrating this over all positive time, yields the charge  $Q_{ij}$  delivered during each synaptic current pulse:



$$Q_{ij} = I_{ij}^0(\tau_2 - \tau_1) \quad (\text{EQ 21})$$

Looking back at EQ1,  $W_{ij}$  has units of charge. where the contribution of neuron  $j$  to the current of neuron  $i$  is given by  $W_{ij}g(u_j)$ . (The function  $g$  takes the current  $u$  and returns a frequency in  $\text{sec}^{-1}$ .  $W$  is then in millicoulombs, yielding millicoulombs/sec = milliamps.)

Therefore, to convert from the  $W_{ij}$  values in the analytic model to the current amplitude values  $I_{ij}^0$  in the spiking simulation, the following was done:

$$I_{ij}^0 = \frac{W_{ij}}{(\tau_2 - \tau_1)} \quad (\text{EQ 22})$$

plugging in for  $\tau_1$  and  $\tau_2$ :

$$I_{ij}^0 = \frac{W_{ij}}{0.130} \quad (\text{EQ 23})$$

### Integration method and equations

The integration was performed using a fourth order Runge Kutta method on 5 variables:  $h$  used for the sodium channels,  $n$  for the potassium channels,  $b$  for the A current channels,  $z$  for the slow potassium channels, and the membrane voltage  $V_m$ .

### Optimization of code

The integration was computationally very intensive, and simulations with multiple neurons over substantial periods of time could take very long. In order to make the simulation run faster, certain parts of the code were redesigned. One of the more expensive computations involved initially was the exponentials that were used to determine alpha and beta values for channels based on the membrane voltage. To circumvent this, a look-up table was generated for the full sweep of normal membrane voltages and the actual values were linearly interpolated from the two nearest values stored in the lookup table in memory. In addition to this, often used variables were kept in registers, and computationally expensive divisions were rewritten as multiplications. These optimizations speeded up the simulation about 5 or 6 - fold.

### Robustness testing of code

Before the data could be trusted, the simulation was checked for robustness and accuracy. Different integration step sizes were tried, and 0.01 msec was determined to be the largest time step that could be used without data drifting significantly after a 5 second simulation. Smaller step sizes took longer without changing the reported data values significantly, while larger step sizes sped up the integration, but caused the data to drift and sometimes even explode. (go out of bounds)

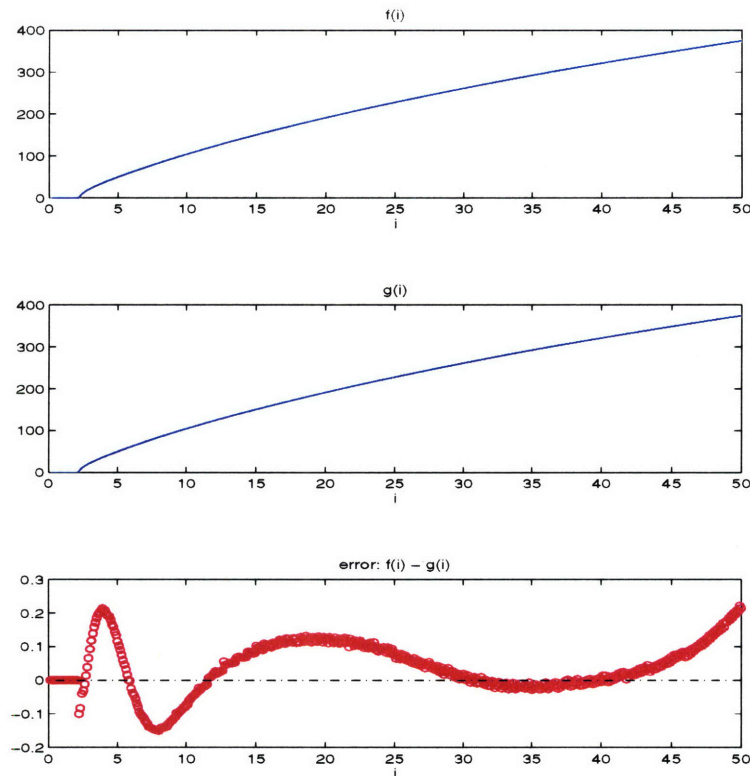
We also dealt with the issue of the precision of the simulation -- the question was whether to use floats or double precision floats. A comparison of simulations at different degrees of precision showed no significant difference from one to the other at the end of 5 seconds of simulation. Standard floating point variables were therefore used. The capability to change the degree of precision by changing only one line of code was added to the software.

### Development of $g$ function approximation.

In order to develop an  $f/i$  characteristic relation function to use in the rate model, the spiking simulation was run with one neuron and a constant external current. After the neuron reached steady state, the average firing rate was recorded. This was done for over 500 different input currents. The results compared favorably with the  $f/i$  data given to us by [Sompolinsky], attesting to the fact that our simulation is consistent with theirs.

For most purposes a linear interpolation of the data points was used. In a few instances, where speed was important and accuracy was not, the following fit was found to be sufficiently accurate:

$$g(i) = \frac{56.76 \times i + 186.1}{i + 11.13} (i - 2.187)^{0.5238} \quad (\text{EQ 24})$$



**Figure 22.** The upper figure shows the  $f/i$  curve  $f(i)$  collected from steady state data from the spiking simulation. The middle figure shows the fit function  $g(i)$  described in EQ 21. The error, plotted in the lower figure, lies mostly within 0.1 percent.

### Simulation platform

Simulations were initially implemented using the package Nodus 3.3 for Power Macintosh. Nodus was easy to use and was sufficiently powerful for the early part of the study. However, in order to compute the larger and more complex simulations, (i.e. the speed needed to model all the synaptic currents and network connections, and keep track of multiple channels in the neuron models,) we wrote a C-code implementation to run simulations on more powerful Sun workstations. (SPARCstation 20).

## B. Simulation code: spikingsim.c

```

/* program settings: choose FIDATA or NORMAL */
#define NORMAL
/* percision of data types: choose float or double */
#define percisiontype float

/* integration parameters in msec */
#define Int_time 30000.0
#define h 0.01
#define MajorTimeStep0.1

/* number of neurons */
#define NN 41

/* synapse settings */
#define BIGTAU 150
#define SMALLTAU20
#define Vsyn 0
#define LENGTH 10000
#define LENGTHH 10001
#define SN 8 /* special neuron -- monitor voltage*/

#define DURATION 5000
/* network settings */
percisiontype hc[NN];
percisiontype W[NN][NN];

/* time constants */
#define TBa 10
#define TZ 60

/* conductances */
#define GNA120
#define GK 10
#define GA 60
#define GNAP0.5
#define GZ 10
#define GL 0.1

/* reversal potentials */
#define Vna 55
#define Vk -70
#define Vl -80
#define Vz -70
#define Va -75
#define Vnap 55

/* capacitance */
#define C 1

/* constants */
#define PHI 4

/* initial conditions */
#define restingVm -70

/* include files */
#include <stdio.h>
#include <math.h>
#include <time.h>
#define CLOCKS_PER_SEC 1000000L

/* global variables */
percisiontype Vm[NN], Hna[NN], Nk[NN], Ba[NN], Z[NN];
percisiontype SPIKETIMES[NN][LENGTHH], VOLTAGE[LENGTH];
percisiontype Iext[NN], Iext2[NN], X, IE, Itot, ff;
percisiontype avgper, mtstep, count, spikecount, oldcount[NN], fire[NN];
percisiontype Hainterp[2000], Hbinterp[2000], Binterp[2000];
percisiontype Nainterp[2000], Nbinterp[2000], Zinterp[2000];
percisiontype Mainterp[2000], Mbinterp[2000], Ainterp[2000], Sinterp[2000];
percisiontype AMPL[2000], TIME[2000];
percisiontype ZETA[NN];
percisiontype Vmpercentlow, Vmpercenthigh;
percisiontype hh, hhh, tba, ttz;

double bt, st, ibt, ist; /*helpers to speed computation */
int Vmhigh, Vmlow, stimcount, fcount[NN], ffc[NN], nn; /*helpers to speed computation*/

FILE *fptr, *fptr1, *fptrv;
char fname1[NN][30];

```

```

void set_initial_values()
{
    percisiontype aHna, bHna, aNk, bNk;

    mtstep= MajorTimeStep / h ;
    hh=h/2;
    hhh=h/6;
    bt=exp(hh/-BIGTAU);
    st=exp(hh/-SMALLTAU);
    ibt=exp(h/-BIGTAU);
    ist=exp(h/-SMALLTAU);
    ttba=1.0/TBa;
    ttz=1.0/TZ;
    setvbuf(stdout,NULL,_IONBF,2);

    for(nn=0;nn<NN;nn++)
    {
        Vm[nn]=restingVm;
        fcount[nn]=0;
        ffc[nn]=0;

        Iext[nn]=0.0;
        Iext2[nn]=0.0;

        aHna= 0.07 * exp( (Vm[nn] + 55) / (-20) );
        bHna= 1 / (1 + exp((Vm[nn] + 25) / (-10) ));
        Hna[nn]= aHna / (aHna + bHna);

        aNk = 0.01 * (Vm[nn] + 45) / (1 - exp((Vm[nn] + 45) / (-10)));
        bNk = 0.125 * exp((Vm[nn] + 55) / (-80));
        Nk[nn]= aNk / (aNk + bNk);

        Ba[nn]= 1 / (1 + exp( (Vm[nn] + 70) / 2));

        Z[nn] = 1 / (1 + exp( (-0.7) * (Vm[nn] +30) ));
    }
}

void setup_interp_tables()
{
    register int i;
    percisiontype val;
    for (i=0;i<2000;i++)
    {
        val=(i-1000.0)/10.0;

        Hainterp[i]= 0.07 * exp( (val + 55) / (-20) );
        Hbinterp[i]= 1 / (1 + exp((val + 25) / (-10) ));

        Nainterp[i]= 0.01 * (val + 45) / (1 - exp((val + 45) / (-10)));
        Nbinterp[i]= 0.125 * exp((val + 55) / (-80));

        Binterp[i] = 1 / (1 + exp( (val + 70) / 2));

        Zinterp[i] = 1 / (1 + exp( (-0.7) * (val +30) ));

        Mainterp[i]= 0.1 * (val + 30) / (1 - exp((val + 30) / (-10)));
        Mbinterp[i]= 4.0 * exp((val + 55) / (-18));

        Ainterp[i] = 1 / (1 + exp( (val + 50) / (-4) ));

        Sinterp[i] = 1 / ( 1 + exp((-0.3) * (val+50) ));

    }
    Nainterp[550]=0.100;
    Mainterp[700]=1.000;
}

void load_weights()
{
    int wi,wj;
    FILE *wfptr;

    printf("loading weights...\n");
    wfptr=fopen("newnet41e","r");

    for (wi=0;wi<NN;wi++)
    {
        fscanf(wfptr,"%f",&hc[wi]);
    }
}

```

```

        printf("%f\n",hc[wi]);
    }
    for (wi=0;wi<NN;wi++)
    {
        for(wj=0;wj<NN;wj++)
        {
            fscanf(wfptr,"%f",&W[wi][wj]);
            printf("%f ",W[wi][wj]);
            W[wi][wj]=W[wi][wj]/0.130;
        }
        printf("\n");
    }
    fclose(wfptr);
}

void load_ff_weights()
{
    int fi;
    FILE *fffptr;

    printf("loading feed-forward weights...\n");
    fffptr=fopen("ZETASIM41","r");

    for (fi=0;fi<NN;fi++)
    {
        fscanf(fffptr,"%f",&ZETA[fi]);
        printf("%f\n",ZETA[fi]);
    }
    fclose(fffptr);
}

void load_stimuli()
{
    int si;
    FILE *sfptr;

    printf("loading stimulus data..\n");
    sfptr=fopen("STIM2","r");

    for (si=0;si>-1;si++)
    {
        if(
            (fscanf(sfptr,"%f %f",&AMPL[si],&TIME[si]) < 1)
            break;
        TIME[si]=TIME[si];
        printf("%f %f\n",AMPL[si],TIME[si]);
    }
    fclose(sfptr);
}

percisiontype eval_v()
{
    percisiontype Ina, Ik, Inap, Ia, Iz, Ii;
    percisiontype aMna, bMna, Mna, Aa, Snap;
    int q;

    aMna= Vmpercenthigh*Mainterp[Vmhigh] + Vmpercentlow*Mainterp[Vmlow];
    bMna= Vmpercenthigh*Mbinterp[Vmhigh] + Vmpercentlow*Mbinterp[Vmlow];
    Mna= aMna / (aMna + bMna);
    Ina= GNA * (Mna*Mna*Mna) * Hna[nn] * (Vm[nn] - Vna);

    Aa = Vmpercenthigh*Ainterp[Vmhigh] + Vmpercentlow*Ainterp[Vmlow];
    Ia = GA * Aa * Ba[nn] * (Vm[nn] - Va);

    Snap = Vmpercenthigh*Sinterp[Vmhigh] + Vmpercentlow*Sinterp[Vmlow];
    Inap = GNAP * Snap * (Vm[nn] - Vnap);

    Ik= GK * (Nk[nn]*Nk[nn]*Nk[nn]*Nk[nn]) * (Vm[nn] - Vk);

    Ii= GL * (Vm[nn] - Vi);

    Iz= GZ * Z[nn] * (Vm[nn] - Vz);

#ifdef NORMAL
    IE=hc[nn];

```

```

if (count>TIME[stimcount] && (count<TIME[stimcount]+DURATION))
    {
        IE=hc[nn]+AMPL[stimcount]*ZETA[nn]*0.29;
    }
/*
if (count>100 && count < 50000 && nn==2)
    IE=hc[nn]+30.0;
if (count< 50000 && nn==2)
    IE=hc[nn]-2.0;
else if (count< 500000)
    IE=hc[nn];
else if (count< 550000 && nn==2)
    IE=hc[nn]+0.8;
else if (count<1000000)
    IE=hc[nn];
else if (count<1050000 && nn==2)
    IE=hc[nn]+1.0;
else if (count<1500000)
    IE=hc[nn];
else if (count<1550000 && nn==2)
    IE=hc[nn]+0.8;
else if (count<2000000)
    IE=hc[nn];
else if (count<2050000 && nn==2)
    IE=hc[nn]+0.8;
else if (count<2500000)
    IE=hc[nn];
else if (count<2550000 && nn==2)
    IE=hc[nn]+0.2;
else if (count<3000000)
    IE=hc[nn];*/
X = Iext[nn] - Iext2[nn];
#endif

#ifdef FIDATA
    X=IE;
#endif

Itot = ( X - Ina - Ik - Ia - Inap - Iz - Il + IE ) / C;
return(Itot);
}

percisiontype eval_h()
{
    register percisiontype aHna, bHna;

    aHna =Vmpercehigh*Hainterp[Vmhigh] + Vmpercentlow*Hainterp[Vmlow];
    bHna =Vmpercehigh*Hbinterp[Vmhigh] + Vmpercentlow*Hbinterp[Vmlow];

    return(PHI * (aHna - ( Hna[nn] * (aHna + bHna))));
}

percisiontype eval_n()
{
    register percisiontype aNk,bNk;

    aNk =Vmpercehigh*Nainterp[Vmhigh] + Vmpercentlow*Nainterp[Vmlow];
    bNk =Vmpercehigh*Nbinterp[Vmhigh] + Vmpercentlow*Nbinterp[Vmlow];

    return(PHI * (aNk - ( Nk[nn] * (aNk + bNk))));
}

percisiontype eval_b()
{
    register percisiontype Bainf;

    Bainf =Vmpercehigh*Binterp[Vmhigh] + Vmpercentlow*Binterp[Vmlow];
    return( (Bainf - Ba[nn]) * ttba);
}

percisiontype eval_z()
{
    register percisiontype Zinf;

    Zinf =Vmpercehigh*Zinterp[Vmhigh] + Vmpercentlow*Zinterp[Vmlow];
    return((Zinf - Z[nn]) * ttz);
}

void interp()
{

```

```

register percisiontype Vmtemp;

    Vmtemp=floor(Vm[nn]*10.0);
    Vmpercenthigh= (Vm[nn]*10.0 - Vmtemp) ;
    Vmpercentlow= 1.0 - Vmpercenthigh;
    Vmlow= (int)(Vmtemp) + 1000;
    Vmhigh= Vmlow + 1;

    if (Vmlow<0)
        {
            Vmlow=0;
            Vmhigh=1;
        }

    if (Vmhigh>1999)
        {
            Vmlow=1999;
            Vmhigh=2000;
        }

}

void update_Vm()
{
percisiontype olderVm[NN],oldVm[NN],oldHna,oldNk,oldBa,oldZ,oldIext,oldIext2,Itemp;
percisiontype v1,v2,v3,v4;
percisiontype h1,h2,h3,h4;
percisiontype b1,b2,b3,b4;
percisiontype z1,z2,z3,z4;
percisiontype n1,n2,n3,n4;
int fc,i,pre,post;

    for(nn=0;nn<NN;nn++)
    {

        oldHna=Hna[nn];
        oldNk=Nk[nn];
        oldBa=Ba[nn];
        oldZ=Z[nn];
        olderVm[nn]=oldVm[nn];
        oldVm[nn]=Vm[nn];
        oldIext=Iext[nn];
        oldIext2=Iext2[nn];
        interp();

        n1=eval_n();
        v1=eval_v();
        h1=eval_h();
        b1=eval_b();
        z1=eval_z();

        Hna[nn]=oldHna+hh*h1;
        Nk[nn]=          oldNk+hh*n1;
        Ba[nn]=          oldBa+hh*b1;
        Z[nn]=          oldZ+hh*z1;
        Vm[nn]=          oldVm[nn]+hh*v1;
        Iext[nn]=Iext[nn]*bt;
        Iext2[nn]=Iext2[nn]*st;
        interp();

        n2=eval_n();
        v2=eval_v();
        h2=eval_h();
        b2=eval_b();
        z2=eval_z();

        Hna[nn]=oldHna+hh*h2;
        Nk[nn]=          oldNk+hh*n2;
        Ba[nn]=          oldBa+hh*b2;
        Z[nn]=          oldZ+hh*z2;
        Vm[nn]=          oldVm[nn]+hh*v2;
        Iext[nn]=Iext[nn]*bt;
        Iext2[nn]=Iext2[nn]*st;

        interp();

        n3=eval_n();
        v3=eval_v();
        h3=eval_h();
        b3=eval_b();
        z3=eval_z();
    }
}

```

```

Hna[nn]=oldHna+h*h3;
Nk[nn]=      oldNk+h*n3;
Ba[nn]=      oldBa+h*b3;
Z[nn]=      oldZ+h*z3;
Vm[nn]=      oldVm[nn]+h*v3;
lext[nn]=lext[nn]*bt;
lext2[nn]=lext2[nn]*st;

interp();

n4=eval_n0;
v4=eval_v0;
h4=eval_h0;
b4=eval_b0;
z4=eval_z0;

Hna[nn] = oldHna + hhh * (h1 + 2*h2 + 2*h3 + h4);
Ba[nn] = oldBa + hhh * (b1 + 2*b2 + 2*b3 + b4);
Z[nn] = oldZ + hhh * (z1 + 2*z2 + 2*z3 + z4);
Nk[nn] = oldNk + hhh * (n1 + 2*n2 + 2*n3 + n4);
Vm[nn] = oldVm[nn] + hhh * (v1 + 2*v2 + 2*v3 + v4);

#ifdef FIDATA
    if(Vm[nn] > -20.0 && oldVm[nn] <= -20.0)
        {
            spikecount+=1.0;

            if (spikecount>0)
                avgper+=count-oldcount[nn];

            oldcount[nn]=count;
        }
#endif

#ifdef NORMAL
    fire[nn]=0.0;
    if (nn==SN)
        {
            if ( (int)(count/mtstep) == (count/mtstep) )
                {
                    VOLTAGE[fcount[nn]]=Vm[nn];
                    fcount[nn]=fcount[nn]+1;
                    if (fcount[nn]==LENGTH)
                        {
                            fptrv=fopen("VOLTAGE8","a");
                            printf(".");
                            fwrite(VOLTAGE,
                                sizeof(percisiontype),
                                LENGTH,fptrv);
                            fclose(fptrv);
                            fcount[nn]=0;
                        }
                    /*printf("%f %f %f %f %f\n",
                        count,Vm[nn],oldVm[nn],fcount[nn],VOLTAGE[fcount[nn]]);
                }
            }
        }

    if (Vm[nn] > -20.0 && oldVm[nn] <= -20.0)
        {
            /*
                printf("spike %d at %f \n",nn, count);*/
            fire[nn]=1.0;
            SPIKETIMES[nn][ffc[nn]]=count;
            ffc[nn]=ffc[nn]+1;
        }

    if (ffc[nn]==LENGTH)
        {
            fptr1=fopen(fname1[nn],"a");
            fwrite(SPIKETIMES[nn],sizeof(percisiontype),LENGTH,fptr1);
            fclose(fptr1);
            ffc[nn]=0;
        }
}

#endif

lext[nn] = oldlext * ibt;

```



```

Iext2[nn] = oldIext2 * ist;
}

for(post=0;post<NN;post++)
{
    Itemp=0.0;

    for(pre=0;pre<NN;pre++)
    {
        Itemp += ((fire[(pre)]) * (W[(post)][(pre)]));

        /*if ( count>149500.0 && (fire[0]==1 || fire[1] ==1))
        {printf("%f %d %d %f %f %f %f\n",count,pre,post,
        fire[(pre)],Itemp,(fire[(pre)]*(W[(post)][(pre)]),
        Vm[(post)]);*/
    }

    Iext[(post)]+=Itemp;
    Iext2[(post)]+=Itemp;
}

if (count>TIME[stimcount] +DURATION)
    stimcount+=1;
}

#endif FIDATA

void main() /* ficurve main routine */
{
    percisiontype max;

    max = Int_time/h;
    printf("success: %d\n",fptr=fopen("newfidata","w"));
    printf("\n");
    printf("Simulation of %d neurons using %.4f msec intervals,\n",NN,h);
    printf("Duration %.2f msec, %.1f integration loops.\n\n",Int_time,max);

    printf("starting...\n");
    setup_interp_tables();

    for(IE=20.0; IE<50.0; IE+=0.1)
    {
        spikecount = -7;
        avgper= 0;
        set_initial_values();

        for( count = 0.0 ; count < max ; count +=1.0)
            update_Vm();

        avgper=avgper/spikecount;
        printf("spikes: %f avg: %f Iext: %f\n",
            spikecount,avgper,IE);
        fprintf(fptr," %f %f\n",avgper,IE);
    }

    fclose(fptr);
}

#endif

#endif NORMAL

void main() /* normal main routine */
{
    percisiontype max;
    time_t t1,t2,t3;
    /*char fname1[30];
    char fname2[30];
    char fname3[30];
    char fname4[30];*/
    int i,j,jj,ffccount;
    percisiontype Ffff[NN][LENGTHH],f,t;

    printf ("start\n");
    max = Int_time/h;

    set_initial_values();
    setup_interp_tables();
    load_weights();

```

```

load_ff_weights();
load_stimuli();

printf("\n");
printf("Simulation of %d neurons using %.4f msec intervals,\n",NN,h);
printf("Duration %.2f msec, %.1f integration loops.\n\n",Int_time,max);

printf("\nData files: ");
for(nn=0;nn<NN;nn++)
    {
        sprintf(fname1[nn],"B%dst",nn);
        fptr1=fopen(fname1[nn],"w");
        fclose(fptr1[nn]);
        printf("%s ",fname1[nn]);
    }
fptrv=fopen("VOLTAGE8","w");
fclose(fptrv);

time( &t1 );
printf("\n\nBegin time: %s",ctime(&t1));
printf("Running");
clock();

for( count = 0.0 ; count < max ; count +=1.0)
    update_Vm();

time( &t2 );
printf("\nEnd time: %s\n",ctime(&t2));
printf("Total simulation time: %d seconds.\n\n",t2-t1);
printf("Actual processor time used: %d seconds.\n",
        clock()/CLOCKS_PER_SEC);
/*for (i=0;i<ffc[SN];i++)
    {
        printf("%f\n",SPIKETIMES[SN][i]);
    }*/
for(nn=0;nn<NN;nn++)
    {
        fptr1=fopen(fname1[nn],"a");
        fwrite(SPIKETIMES[nn],sizeof(percisiontype),ffc[nn],fptr1);
        fclose(fptr1);
        if (nn==SN)
            {
                fptrv=fopen("VOLTAGE8","a");
                printf(".");
                fwrite(VOLTAGE, sizeof(percisiontype),
                    fcount[nn],fptrv);
                fclose(fptrv);
            }
    }
}
#endif

```