Evolving Minimal Bursting Pacemaker Neurons

Ben Axelrod

GE 485 - Genetic Algorithms in Search,
Optimization, and Machine Learning
Final Project – Fall 03

Submitted to:
Professor David E. Goldberg
deg@uiuc.edu

Abstract

A Genetic Algorithm (GA) was used to evolve and optimize the parameters of an ion channel model neuron to produce pacemaker bursting. The number of ion species was varied from 3 to 8, in order to determine the minimum species required for this task and what, if any, improvement does adding more species contribute. For each ion species, the GA optimized the reversal potential, maximum conductance, gate threshold, and time constant. Although the fitness function proved too difficult to design to provide bursting neurons, similarities in the results were encouraging. This effort laid the groundwork for an ongoing effort to evolve simple and robust artificial neurons.

Introduction

Neuron Model

Neurons are an important part of neural computation. Networks of neurons make up the human brain. They work by receiving electrical inputs from other cells through the dendrites then producing an output along the axon based upon the inputs and their weights. This simple model is utilized in artificial neural networks to do complex computation.

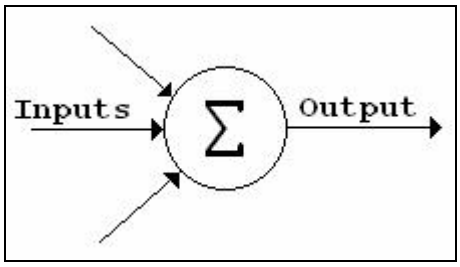

Figure 1 – Simple Neuron Model

The summing behavior, although seemingly electrical is actually a chemical process. It works by having the inputs raise or lower the membrane voltage of the cell body. The cell then opens or closes channels that let certain species of ions through. In its attempt to reach its resting voltage, the subsequent fluxes of ions cause the voltage of the cell to spike. This spike is then transmitted to another cell…

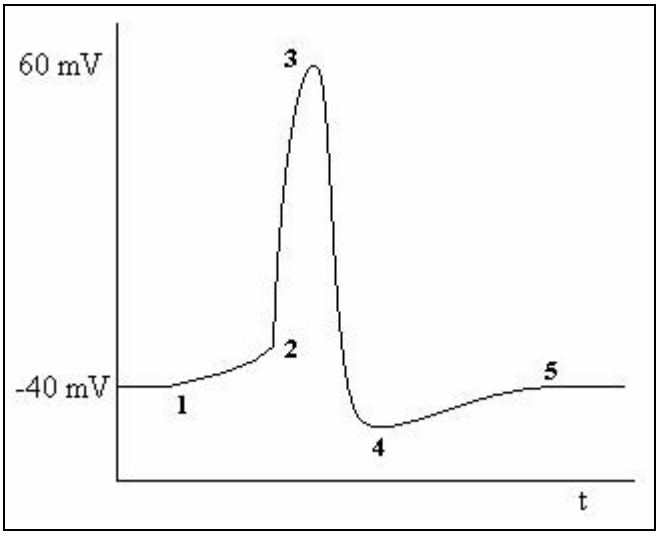

Figure 2 – Neuron Spike Creation

1. Signals from the dendrites start to raise the membrane voltage.

2. The voltage reaches the gate threshold for sodium and the gates open. Large quantities of this positively charged ion rush into the cell because of lower concentrations there, which force the voltage up.
3. The sodium gates close and the potassium gates open. This time, the positively charged potassium ions leave the cell because of higher concentrations inside the cell.
4. All gates are closed.
5. Leakage currents bring the voltage back to its resting potential.

This is a highly simplified description of how an individual spike is produced. In reality, there are many more ion species, each with its own threshold and conductance. By manipulating the thresholds, the cell can exhibit a variety of outputs: single spikes, bursting, etc. Some cells exhibit these characteristics all the time, regardless of inputs. These cells are called pacemaker cells. It has previously been assumed that it takes about 7 channels to exhibit pacemaker bursting [2].

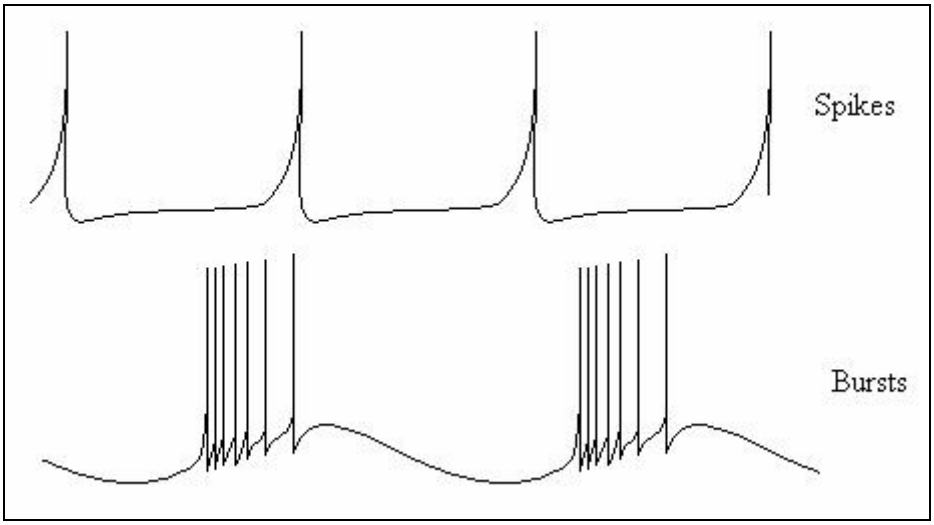

Figure 3 – Different Kinds of Neuron Output

Biologists have recognized networks of these neurons in the spinal cords of humans, which produce the rhythmic electrical pulses necessary for tasks such as walking. These networks are called Central Pattern Generators (CPGs). CPGs are of interest to the robotics community because they promise to be able to control a legged robot's gait effectively [3]. These CPGs can be simulated on a computer or hard wired with solid-state electronics. It is this reason why a simple and robust neuron model is sought.

Simulation Model

The cell can be modeled like a capacitor, with the cell membrane voltage being controlled by the currents applied to it. Each species is modeled as an input current to the cell:

$$C \cdot \frac{dV_{mem}}{dt} = \sum_{i=1}^{N} I_i \,. \tag{1}$$

Where N is the number of ion species, C is the membrane capacitance of the cell and V_{mem} is the membrane voltage. Each species current is governed by this equation:

$$I_i = g_i \cdot (E_i - V_{mem}) \,. \tag{2}$$

Here, g is the conductance of the species, and E is the reversal potential. The conductance can be thought of as the weight of that species. Using Euler's method to solve the differential equation in equation 1, yields:

$$V_{mem(t)} = \frac{\sum_{i=1}^{N} g_i \cdot (E_i - V_{mem})}{C} \cdot \Delta t + V_{mem(t-1)} \,. \tag{3}$$

In addition to this, the conductance values are modified every time step. We created a simplified, computationally tractable model of conductance variations. If the membrane voltage is above the threshold for that species, the conductance is set to the maximum. When the voltage falls below the threshold, the conductance decays exponentially with time constant τ_G.

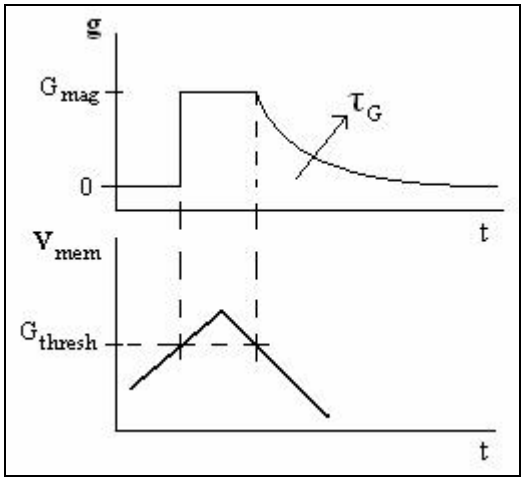

Figure 3 – Species Conductance Thresholds

Each species is characterized by four quantities: G_{mag}, G_{thresh}, E, and τ_G. The cell membrane conductance, and Δt are fixed constants. Each neuron must be simulated for a long enough time to analyze the waveform produced. During the simulation, V_{mem} and g_i are varied according to the equations detailed above. For all tests the constants used were: $\Delta t = 0.001$ seconds, and $C_{mem} = 40$ pF.

This implementation differs, perhaps, from others in the past because it does not set biologically based values for the species. Instead, the GA will choose these values in hopes to reduce the required number of species. Note: most models of neurons include a leak current and usually a signal input. This model includes neither. It does not need an input because one of the species will evolve to always be on and act as an input. The model does not need a leak term either, which would just add unnecessary complexity.

GA encoding of neural values

Our goal was to determine why natural CPGs have evolved the large number of ion species. A GA was chosen for this task because of its ability to search a highly dimensional space for an optimum solution [1].

Genetic algorithms find the solution to a problem by exploiting the Darwinian paradigm of survival of the fittest. The GA differs from other solvers in that it works with a population of possible solutions. Each individual in this population represents the possible solution through a coding called the genome. This coding is usually binary, but it can be anything. The individuals are ranked according to how well they solve the problem, and the better individuals are given precedence in mating. The basic idea is that the better solutions will have a greater chance to mate and then have a better chance of passing on its genes.

The GA determines the better individuals through a fitness function. This function assigns a fitness value based on the individual's performance in the problem to be solved. In this case, better bursting neurons get more fitness.

A GA package, called GAlib, was used to run these experiments. All tests ran using a simple GA, P_c=0.8, P_m=0.1 and a population size of 1000. The runs were stopped at 800 generations. The standard mutation and crossover were used along with

tournament selection. Some initial tests were run using a genotypic sharing function, and a "steady state" GA. Neither of these options improved the results much, if at all.

A procedure similar to simulated annealing was also tested. In this procedure, the population was run for a number of generations, then reinitialized and started over. The 10 fittest individuals from before the re-initialization were used to seed the new population. This procedure helped to keep the population from converging to a sub-optimal peak. Unfortunately, there was not enough time to fully examine the benefits of this procedure and these results are not published.

GA genome

A real coded genome was used because the values in the equations are real valued. Using a binary encoded genome for this application would add an unnecessary complexity. For each species, there are four parameters that the GA will optimize: G_{mag}, G_{thresh}, E, and τ_G. These parameters make up the genome. The parameters were grouped by species to keep a tight coding.

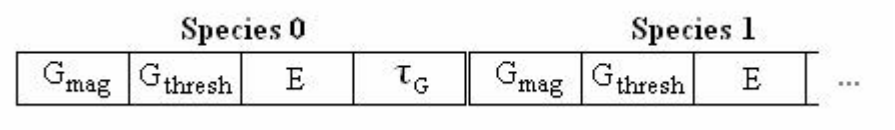

Figure 4 – Real Number Coding Genome

Bounds were then chosen based on experimentally measured values and previous models [4], but mostly what worked.

Gene	Min	Max	Units
G_{mag}	0	10	nS
G_{thresh}	-200	200	mV
E	-500	500	mV
τ_G	0.05	1	ms

Table 1 – Gene Bounds

Waveform Analysis

A large part of this experiment was to determine which quantities of the waveform to measure to assign a fitness measure. Some concerns were fast simulation times and how to determine if the neuron is bursting without imposing too many constraints.

We chose to use a "peak timer" system. It works by starting a timer every time a "good" peak is detected. Any other peaks that occur within this time zone are considered part of one burst. A good peak was determined by examining the last 5 time steps. If there was a peak with a minimum height of 60 mV and peaked above 0V, it was recorded as a good peak. Conversely, a bad peak was a small peak that did not meet the two criteria. Figure 5 illustrates an example where each burst has 2 good peaks and 1 bad peak.

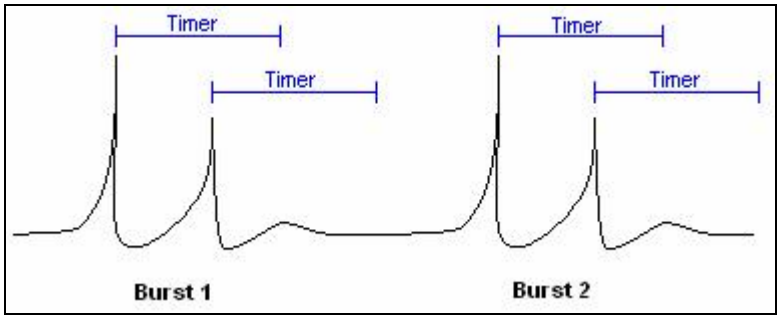

Figure 5 – Peak Timers

Now that bursts, and the number of peaks in them can be measured, many other useful quantities can be derived such as duty cycle, average value of flat time, standard deviation of flat time, and inter-spike standard deviation. This timer method does not impose too many constraints on the frequency of the output. Basically it just sets the minimum flat time.

The peak timer method allows us to quickly distinguish the type of behavior of the neuron. Such as flat lining, spiking, bursting, or transient bursting. This method also allows some analysis to be done during the simulation. This "online" analysis can be used to stop the simulation when enough data is recorded, or a major failure criteria is reached. This gave us faster simulation times. Our code was able to process between 5 and 9 generations per minute.

The Fitness Function

The largest part of this project was designing the fitness function. Without a good fitness function, a GA cannot perform well. As previously mentioned, the number of peaks per burst, bad peaks per burst, and duty cycle were measured. These quantities facilitated fitness scoring. Each neuron was simulated for a maximum of 6000 time steps

(6 seconds). If 20 bursts were recorded during this time, the simulation was stopped early. To avoid errors, the first .2 seconds of the simulation was ignored.

Initially, very simple functions were tried. These functions assigned fitness based only on number of good spikes per burst and the duty cycle or the burst. More parameters were added, and weights were changed until the final code which is detailed here.

The first part of the fitness function is meant to throw out bad cases such as when the neuron does not spike at all or when only one or two bursts are found. Zero fitness is given to flat liners. Finding only one or two bursts means that the time ran out in the middle of a burst. Because of this, other parameters such as peaks per burst, and duty cycle cannot be measured. A very low fitness is given to these individuals.

If a spiking neuron is detected, the fitness function tries to encourage the bursting behavior by awarding partial credit for qualities that might lead to bursts [5]. Partial credit is given for the average of the flat time being close to –40 mV, a small standard deviation of the flat time, using all its ion species, and greater peak to peak standard deviation. Having a greater peak-to-peak standard deviation will cause the peaks to become less regular, and more bunched, hopefully forming bursts.

The cases of two or more peaks per burst include the same partial credits with different weightings, plus a few more. These include a large penalty for having more than 2 bad spikes per burst, a reward for being close to a duty cycle of 40%, and a penalty for having a varying number of peaks per burst.

Discussion and Results

As previously mentioned, overly simple fitness functions were tried first. These runs produced nice spiking behavior and a maximum of two spikes per burst. A sample result of a run with 5 ion species can be seen in figure 6.

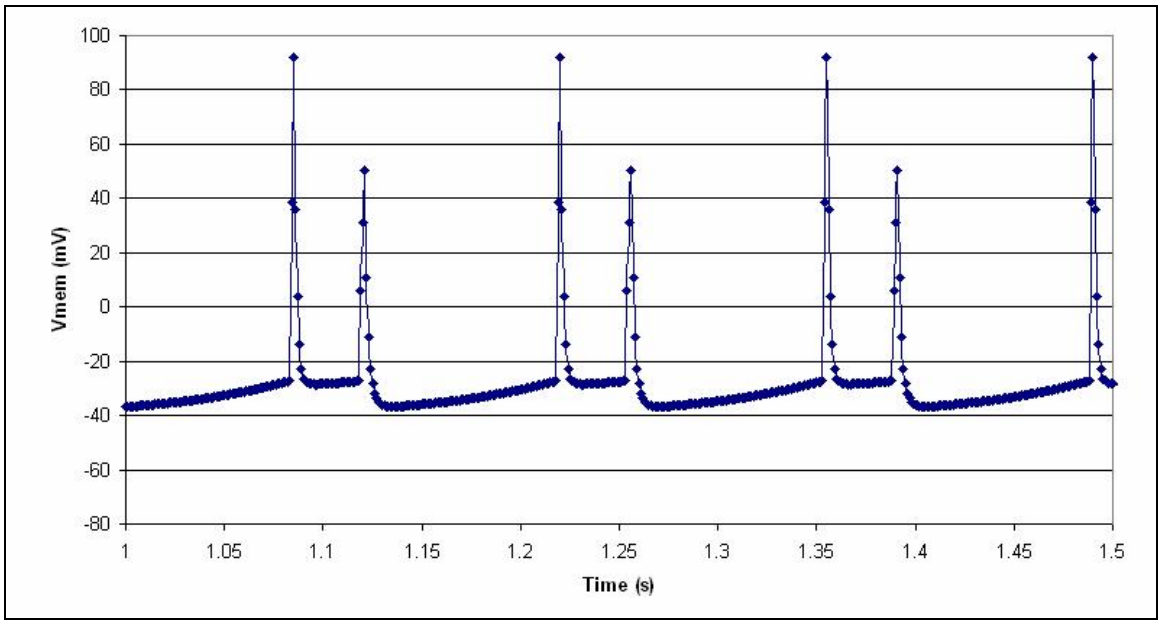

Figure 6 – Output Using Simple Fitness Function, 5 species

Clearly, the neuron can produce better results than this. The fitness function was then continually improved in an attempt to get the neuron to burst properly. Six experiments were run, one for each number of ion species. The fitness function and other GA parameters are the same across all runs. Plots of the waveforms of the fittest individual at the end of the run and the convergence data were plotted. The full data can be found in Appendix A.

The neuronal input from the run with 7 species is a good representative for the rest of the runs. Most runs produced a very high frequency spike with a slower underlying frequency. This is approximately what bursting is, with the addition of a flat period.

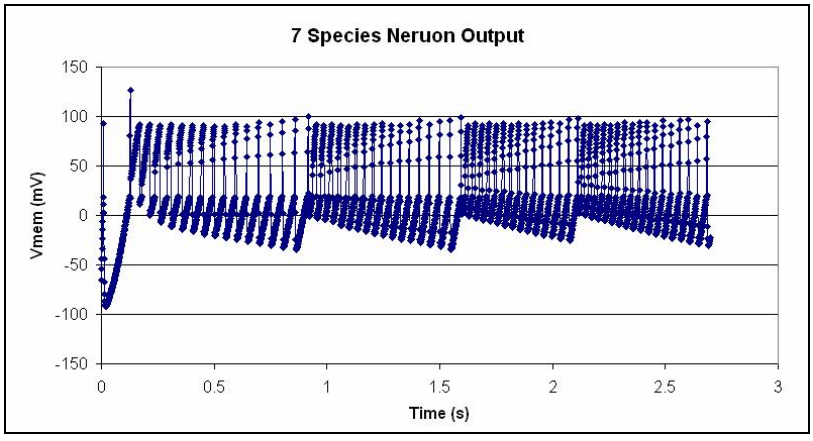

Figure 7 – Seven Ion Species Output

From analysis of the convergence plots, it is clear that adding more neurons does not add as much complexity as one might think. A good reason perhaps why biological systems have so many.

The genomes produced can also be examined. We can find similar types of ions in all the results. For example a species type with these characteristics can be found in all the run results: $G_{mag} \approx 10$, $G_{thresh} \approx 100$, $E \approx -300$, and $\tau_G \approx 0.05$. This species type is responsible for the down part of the spike. When we look for more of these similarities, we notice that the runs with a higher number of species can have multiple versions of these types. For example, the species type $G_{mag} \approx 10$, $G_{thresh} \approx 100$, $E \approx 350$, and $\tau_G \approx 0.05$, can be seen split into two types in the run with 8 species where the G_{mag}s sum to ≈ 10. This suggests that there are a certain number of important species.

Conclusion

In summary, these results are disappointing because no good bursting was produced and it is clear that either a more complex fitness function, or model is needed. However, they display similar patterns in the outputs of most runs. This is encouraging because it suggests that a neuron with a small number of species can exhibit a similar waveform to a neuron with more. This work provided a good first step into an ongoing study of evolving simple neuron models.

Acknowledgements

I would like to thank my collaborator Dr. M. Anthony Lewis for helping with the ion species model. A special thanks also goes out to Hyo-Kyung Lee for assisting with the GAlib installation and putting up with my lack of Unix skills, and Liudmila Yafremava for a fresh biological perspective on the subject.

References

1. Goldberg, David E. "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison Wesley, 1989

2. Hille, Bertil "Ionic Channels of Excitable Membranes", Sinaur Associates Inc.. 1984.

3. M. Anthony Lewis, Andrew H. Fagg and George Bekey, "Genetic Algorithms for Gait Synthesis in a Hexapod Robot", In Zheng, ed. *Recent Trends in Mobile Robots*, pp 317-331, World Scientific, New Jersey, 1994.

4. Rybak et al. "Endrogenous rhythm generation in the pre-Botzinger complex and ionic currents: modeling and in vitro studies", *European Journal of Neuroscience*, Vol. 18, pp. 239-257, 2003.

5. West et al. "Using Evolutionary Algorithms to Search for Control Parameters in a Nonlinear Partial Differential Equation", *Institute for Mathematics and Its Applications Volume on Evolutionary Algorithms and High Performance Computing (1999) Springer-Verlag, New York. pp. 33-64.*

Appendix A – Full results

3 Species:

G_{mag}	G_{thresh}	E	Tau_G
7.95403	80.2323	354.452	0.05
0.736813	-163.969	120.709	0.068703
7.36365	1.56E+02	-257.37	0.05

Table A1 – Fittest Individual with 3 Species

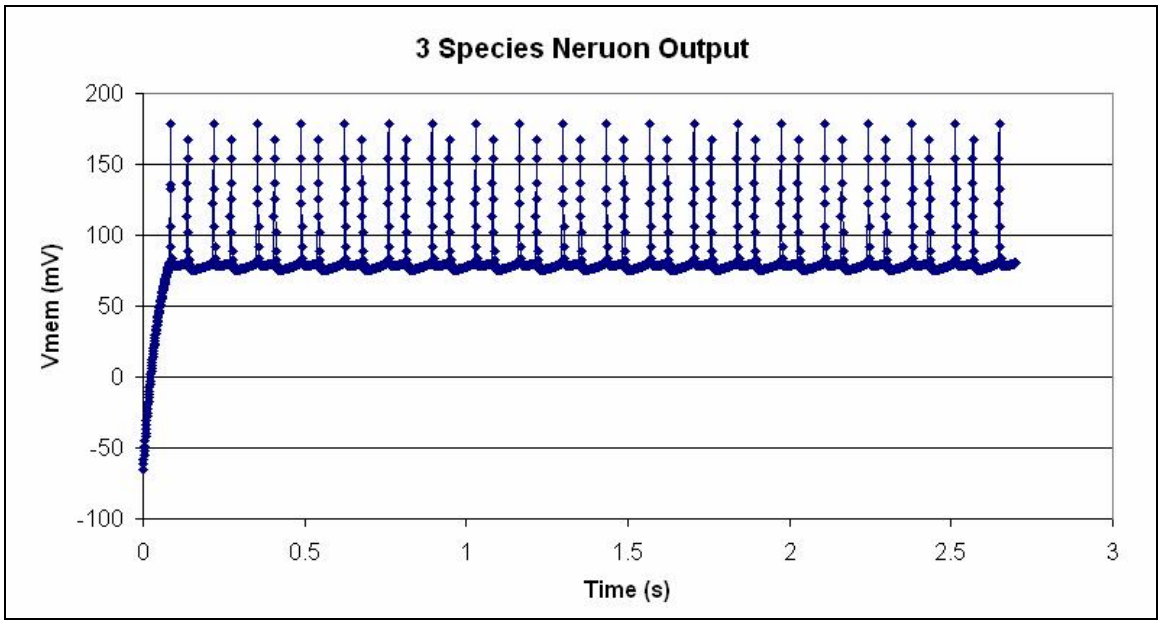

Figure A1 – Three Species Output

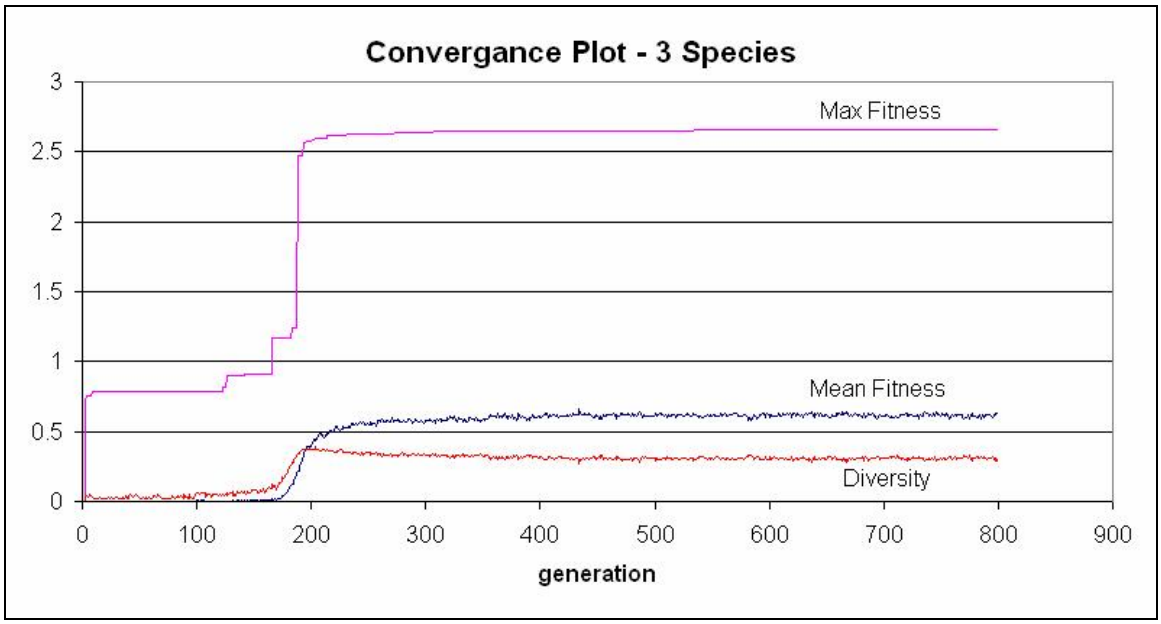

Figure A2 – Three Species Convergence

4 Species:

G_{mag}	G_{thresh}	E	Tau_G
9.74785	92.6831	-370.96	0.05
2.09512	-118.028	401.944	1
9.43245	1.17E+02	-352.42	0.39606
3.76461	3.94922	457.64	0.05

Table A2 – Fittest Individual with 4 Species

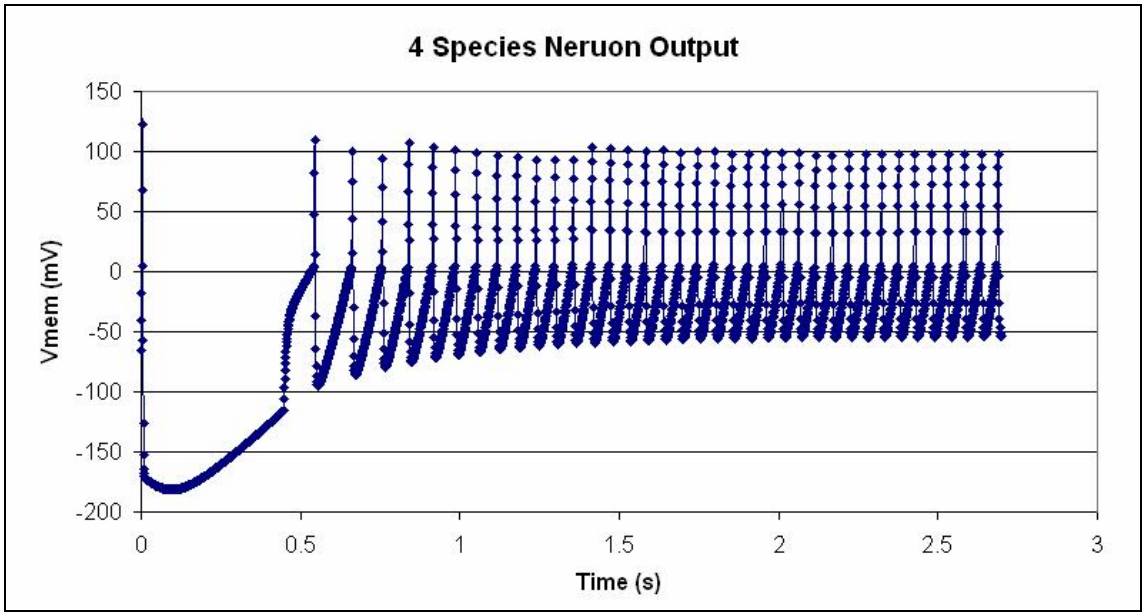

Figure A3 – Four Species Output

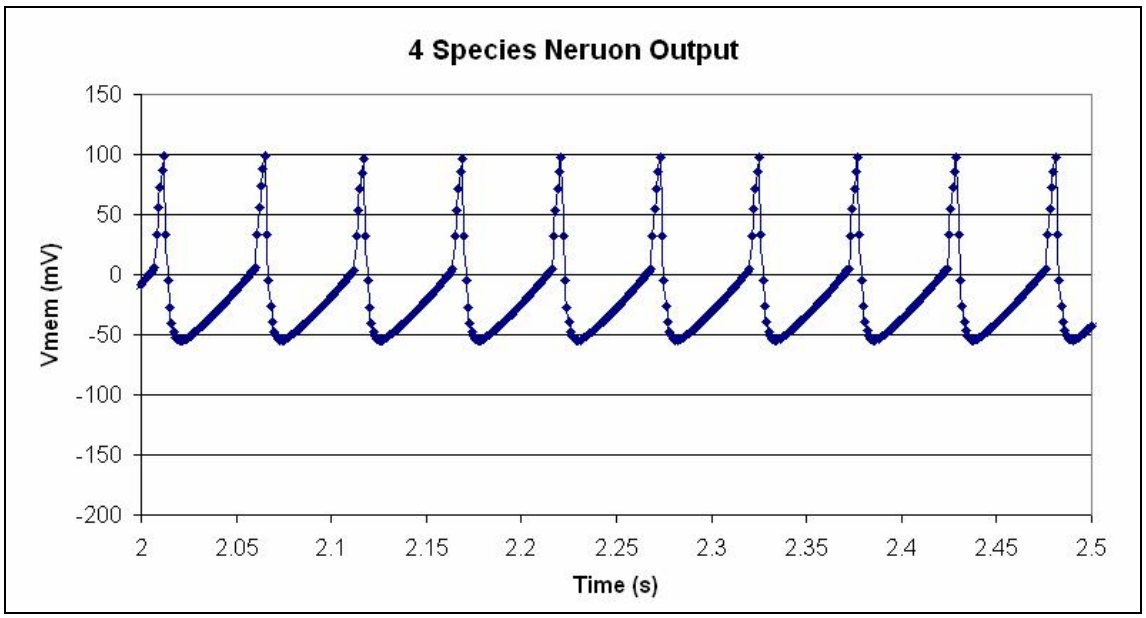

Figure A4 – Four Species Output Close-up

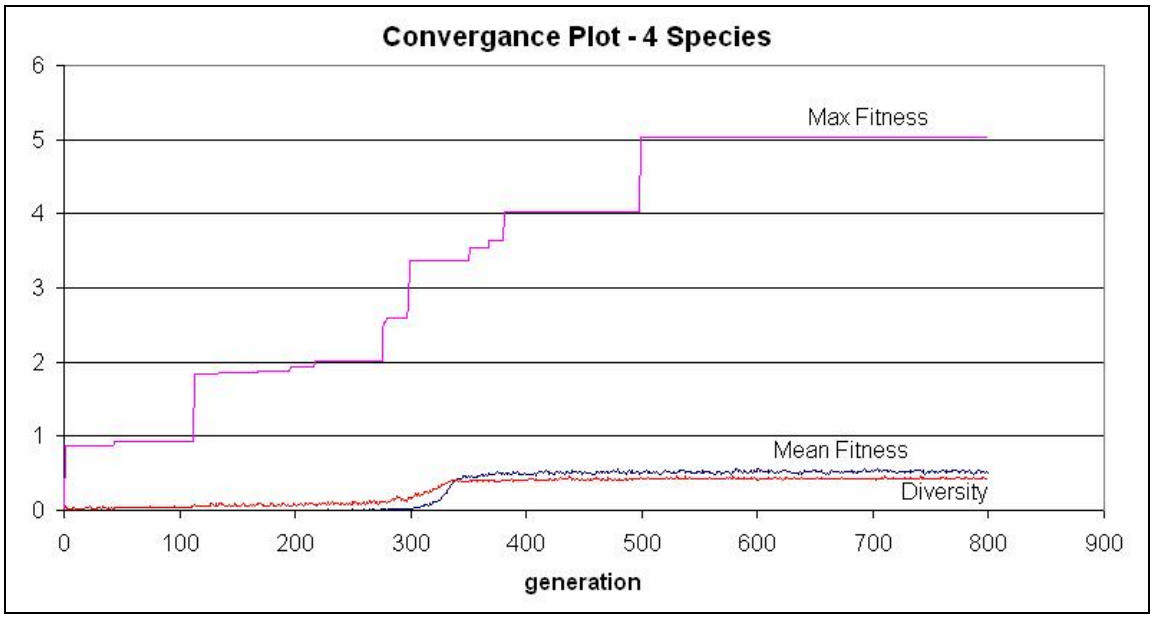

Figure A5 – Four Species Convergence

5 Species:

G_{mag}	G_{thresh}	E	Tau_G
0.01863	-186.222	42.0712	1
1.98515	5.37106	291.388	0.05
2.09791	1.26E+02	419.697	1
4.23193	53.9083	400.055	0.05
8.85349	115.351	-325.395	0.05

Table A3 – Fittest Individual with 5 Species

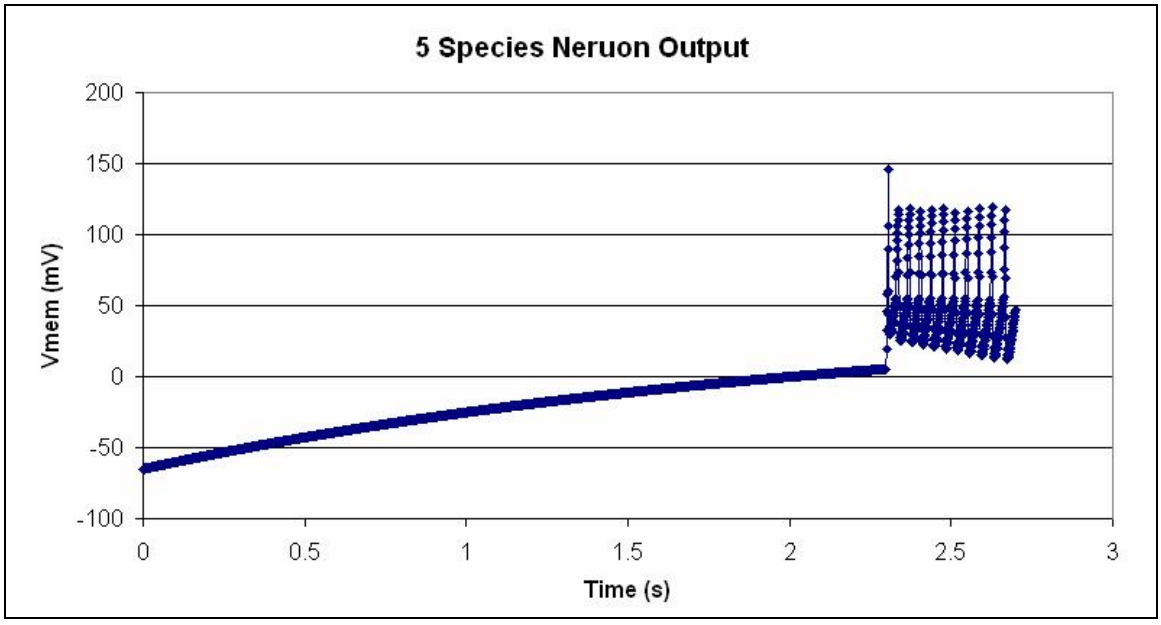

Figure A6 – Five Species Output

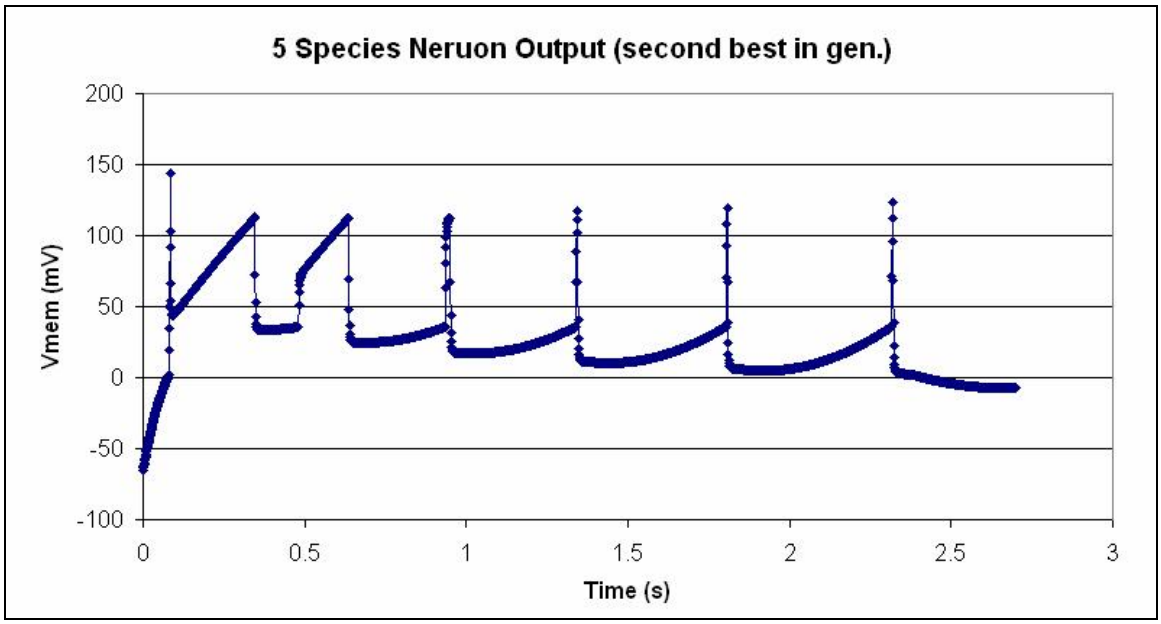

Figure A7 – Five Species Output of Second Best in Generation

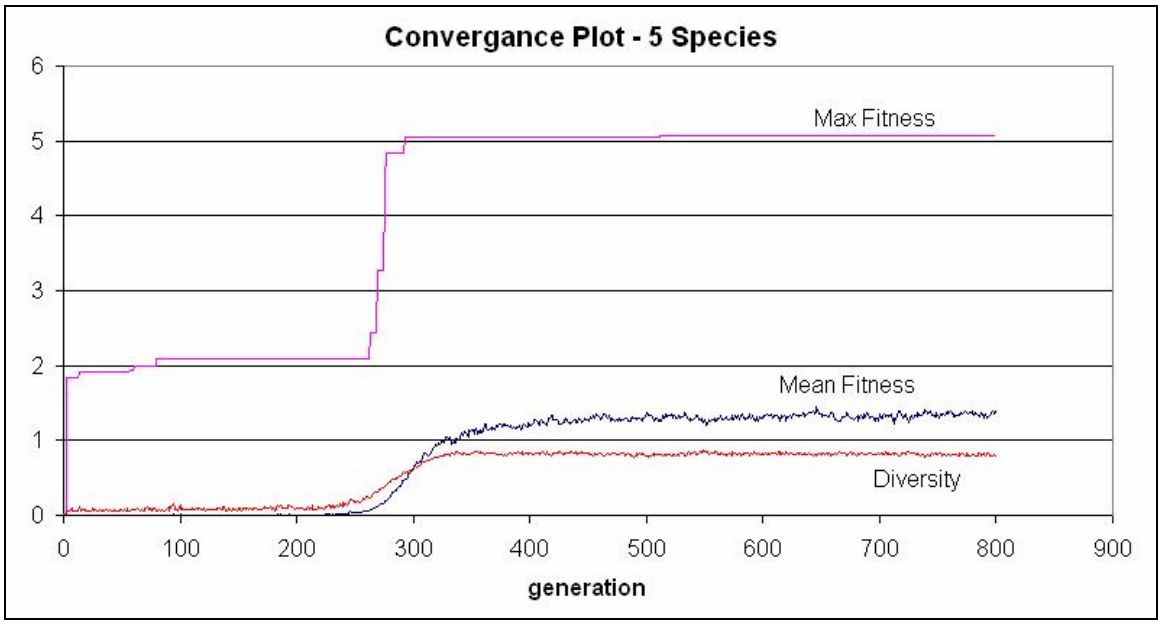

Figure A8 – Five Species Convergence

6 Species:

G_{mag}	G_{thresh}	E	Tau_G
8.99708	135.889	-320.356	1
2.75556	-196.871	395.034	0.25658
2.86E+00	34.6367	-49.1892	0.05
7.45688	49.162	312.672	0.53244
7.3317	99.9364	250.739	0.05
0.589734	163.579	-336.281	0.24599

Table A4 – Fittest Individual with 6 Species

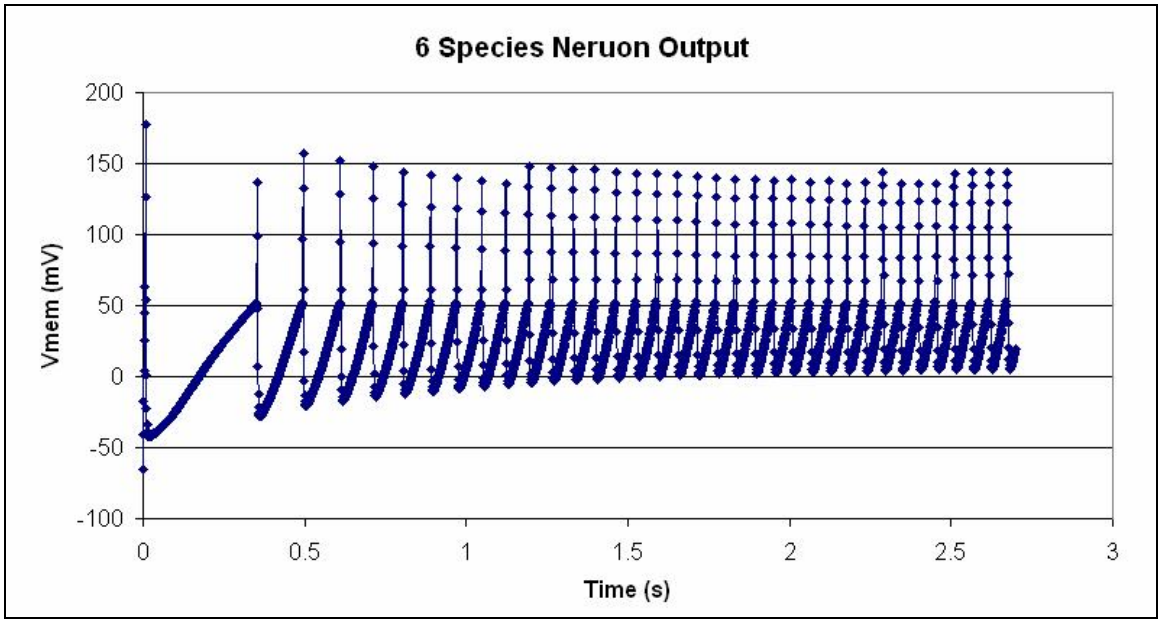

Figure A9 – Six Species Output

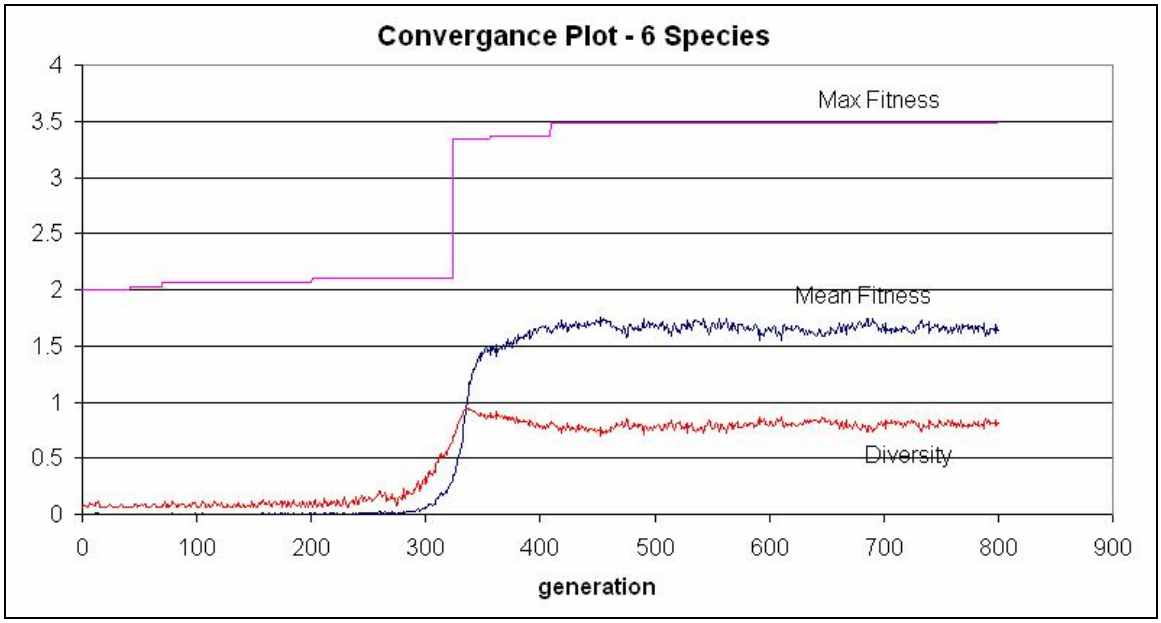

Figure A10 – Six Species Convergence

7 Species:

G_{mag}	G_{thresh}	E	Tau_G
10	90.1027	-497.066	0.05
10	117.896	181.939	0.05
9.97367	1.21E+02	196.446	1
1.68287	-125.238	196.138	0.05
3.41148	97.7241	499.111	1

7.41164	18.4538	377.021	0.05
4.2423	123.491	-489.164	1

Table A5 – Fittest Individual with 7 Species

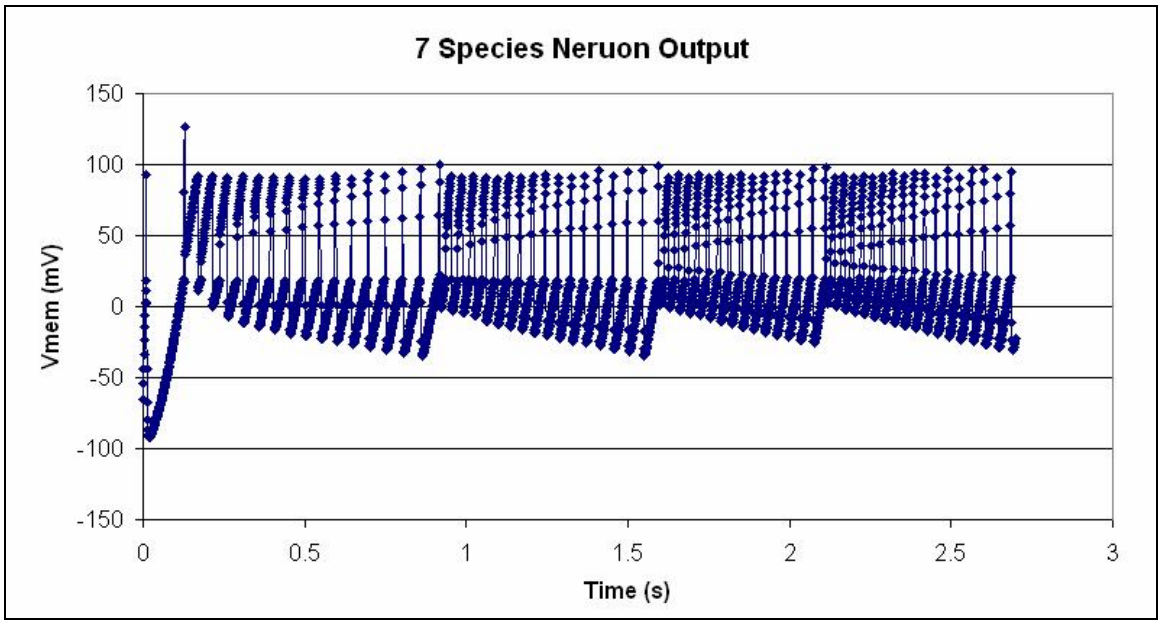

Figure A11 – Seven Species Output

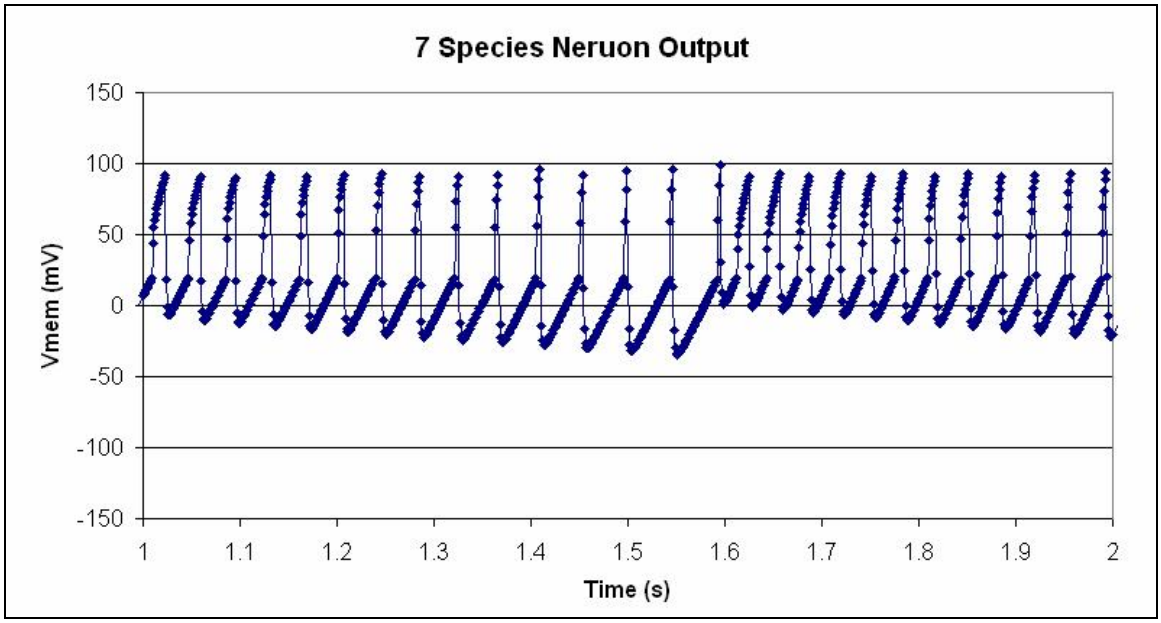

Figure A12 – Seven Species Output Close-up

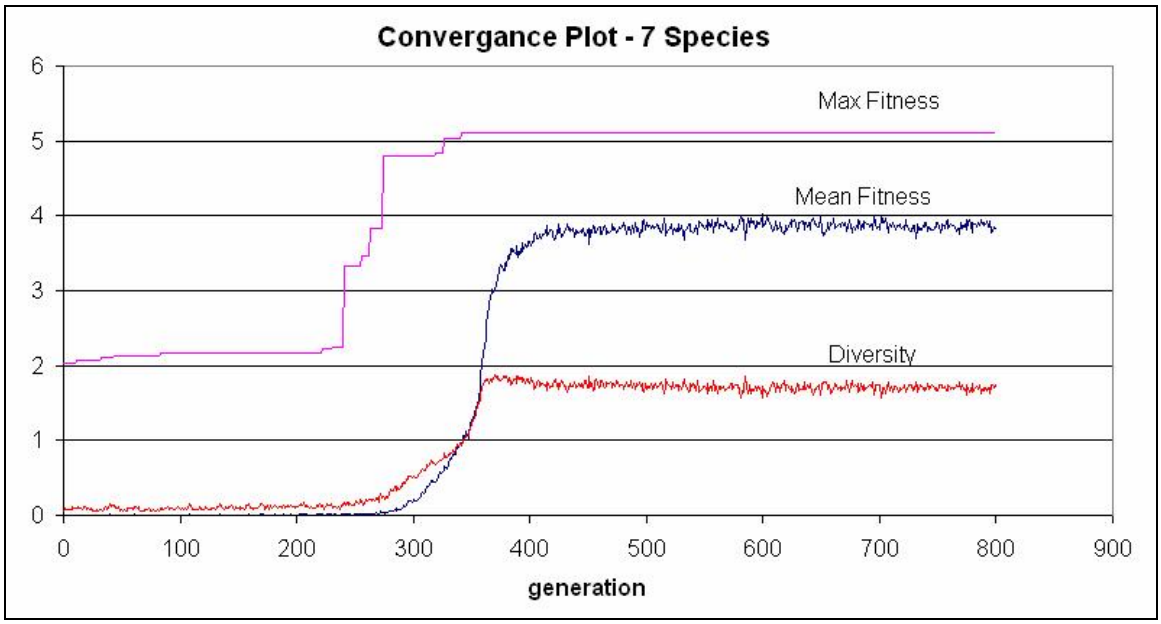

Figure A13 – Seven Species Convergence

8 Species:

G_{mag}	G_{thresh}	E	Tau_G
2.77576	9.27058	136.8	1
10	119.435	-417.285	0.05
0.855884	3.31E+01	324.023	0.05
0.0147875	-147.446	438.427	0.72246
0.595597	102.716	20.2091	0.05
1.56674	132.416	143.471	1
6.03896	37.4548	408.035	0.05
3.63265	104.296	345.533	0.05

Table A6 – Fittest Individual with 8 Species

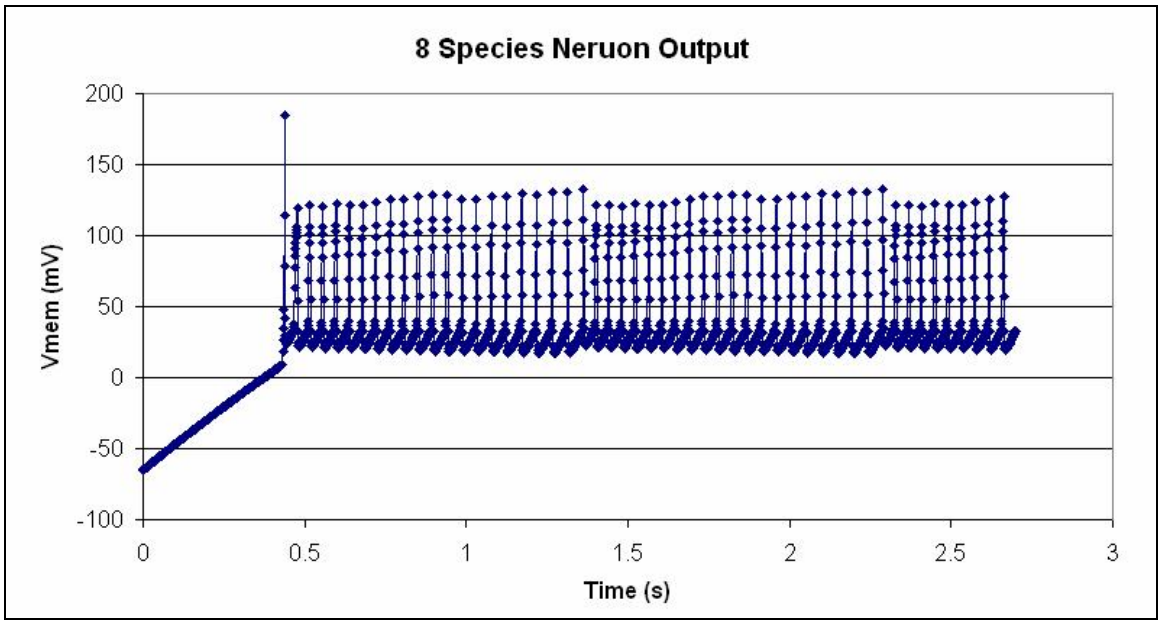

Figure A14 – Eight Species Output

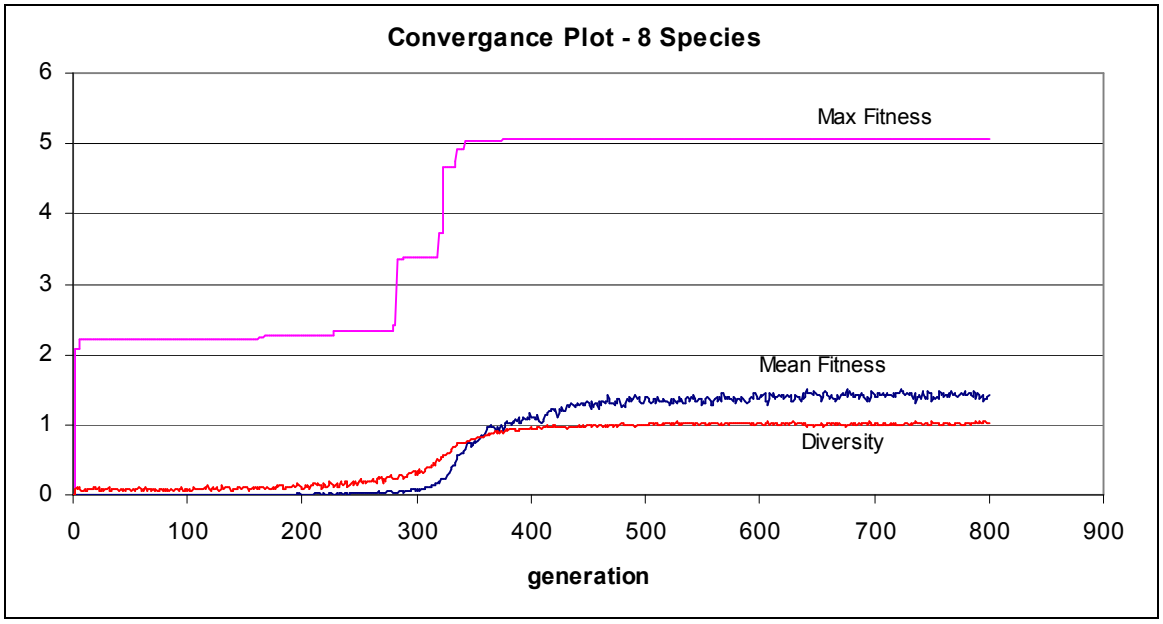

Figure A15 – Eight Species Convergence