Self-organized contractions in the *Hydra* spp.: a computational exploration

Konstantinos Batsis
Supervised by: Ot de Wiljes, dr. Fred Keijzer
6/7/2017
University of Groningen

Abstract

The *Hydra* genus consists of small freshwater animals. Among their behavioral repertoire they exhibit spontaneous contractions of their tubular body which are thought to be controlled by the animal's nerve net which is wrapped around this body. We propose that these contractions emerge through a process of self-organization, that is that they dynamically come about through the inherent activity and interactions of the nerve net's regular neurons. We explored the feasibility of this proposal by creating a computational model of the nerve net of Hydra based on the integrate-and-fire model of the action potential. To a large extent we were to able to reproduce contractile-like neural activity. We also demonstrated that this synchronized activity persists in a sub-region of the model's parameter space. In another part of this space the model is near the border of a phase transition between synchronized and desynchronized activity which suggests the possible existence of further self-organized behaviors.

Keywords: Hydra, contractions, self-organization, integrate-and-fire, Brian

Introduction

The Hydra species

Our research centers on a genus of small freshwater animals called Hydra (Passano and McCullough, 1964). Morphologically these organisms have a tubular body with a mouth surrounded by tentacles at one of the edges of this body (Swanson, 2012; see figure 1). They belong to the Cnidaria an animal phylum which is one of the first to have evolved with members who have nervous systems (Bosch et al., 2017; Swanson 2012). Hydras can locomote and show active feeding behavior (Passano and McCullough, 1964; Swanson 2012).

Anatomically the tubular body consists of three layers, the epidermis facing the external environment, the gastrodermis which forms an internal cavity with digestive functionality and the mesoglea layer in between those two (Johnson, 2003; Swanson, 2012; see figure 1). The body is populated by two types of neurons: sensory and motor. Sensory neurons are located at the epidermis and detect environmental stimuli. Motor neurons receive input from the sensory neurons and send their output to effector cells. The motor neurons are scattered across both the epidermis and the gastrodermis and besides their connections to the effector cells they also form connections between them. These later motor-to-motor connections are bidirectional and are served by reciprocal synapses (Swanson, 2012; see figure 1). Diffuse networks of neurons of this kind are called a nerve nets (Swanson, 2012).

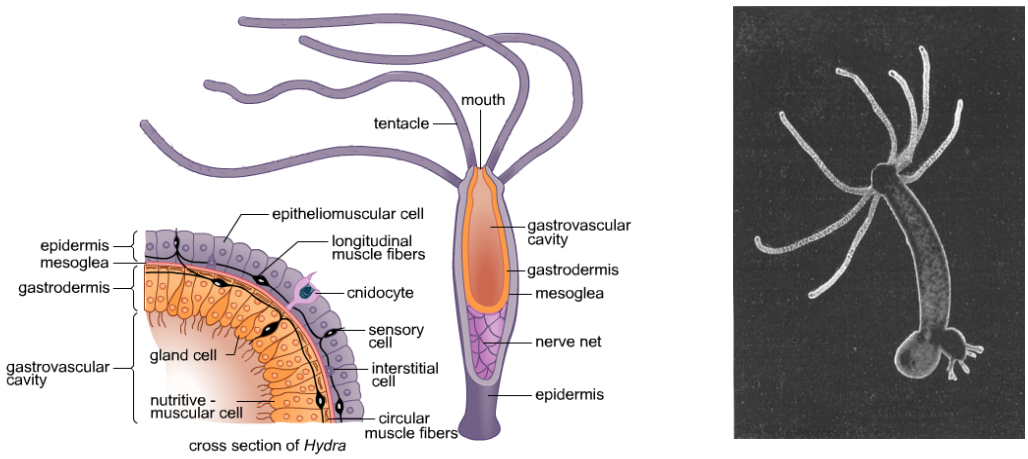

Figure 1. Left: anatomy of the *Hydra* spp. (Johnson, 2003), Right: Photograph of a Hydra.

The nerve net is thought to underlie the spontaneous periodic contractions these animals undergo during which they progressively contract to spherical shape and then extend again (Passano and McCullough, 1964) . These contractions happen every 5 to 10 minutes and this rate is affected by ambient illumination levels and light stimuli (Passano and McCullough, 1964). Each contraction consists of a number of individual contractions as revealed by electrophysiological recordings on the animal (Passano and McCullough, 1964; see figure 2). It has been suggested that the contractions are driven by pacemaker cells (Passano and McCullough, 1964; Takaku et al., 2014) but in this paper we will suggest a different proposal based on the phenomenon of self-organization.

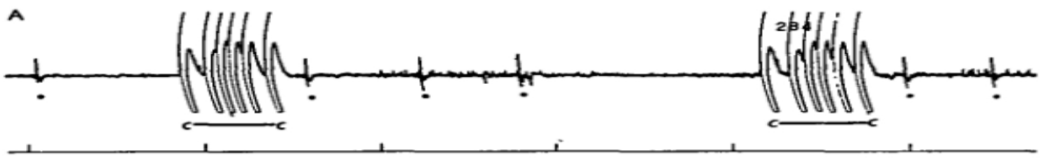

Figure 2. Electrophysiological record of spontaneous contractions in *Hydra pirardi*, lower line presents time in minutes, impulses are about 100 μV (Passano and McCullough, 1964).

Research on the *Hydra* spp. may be important since these animals possess small scale and relatively simply structured nervous systems. This allows them to be completely modeled and controlled in ways not possible with the nervous systems of more evolved organisms. By taking advantage of the tractability of Hydra we may be able to understand aspects of the operation of more complex nervous structures (Bosch et al., 2017). Considering the very early evolutionary origins of the Hydra species these organisms can also be helpful in the attempt to understand the original function and evolution of the nervous system (Bosch et al., 2017).

Self-Organization

A complex system is a system composed of many components which possibly interact with each other. In formal models of these systems their evolution is governed by rules specifying the behavior of the components and their mode of interaction (Camazine et al., 2003). The climate, the economy, an ant colony, the processes in a living cell and indeed the nerve net of *Hydra* spp. can all be considered as complex systems.

In complex systems global patterns in space and/or time can emerge. They are not explicitly encoded in the rules of the system and become apparent only in the operation and interactions of its components in time. This ordered behavior has been termed self-organization and it occurs even though the units of the system have access only to local information and there is no environmental guidance for it to occur (Camazine et al., 2003). An example of a self-organized spatial pattern is the pigmentation in the *Conus textile* shell and an example of such a temporal pattern is the synchronous flashing of fireflies (Camazine et al., 2003; see figure 3).

In this paper we propose that the nervous activity underlying the contractions of the *Hydra* spp. could possibly come about through a process of self-organization. To investigate this proposal we will use computational modeling.

Figure 3. Left: a *Conus textile* shell, it's pigmented pattern is an example of spatial self-organization (© Richard Ling), Right: The synchronous flashing of the fireflies, an example of temporal self-organization (© Judd Patterson).

Modeling of biological neural networks

Advances in mathematical and computational neuroscience have made possible the creation of realistic models of biological neural networks. At the core of these models lie mathematical models of the neuron which attempt to formally recreate the action potential. Examples of such models are the integrate-and-fire and Hodgkin-Huxley neurons (Anderson, 2014). These mathematical neurons are then arranged in space according to the anatomical features of the to be modeled organism while mathematical expressions specifying the relationships between these neurons represent the synaptic connections. Software packages that allow the easy construction of such neural network models have also been developed, major examples are Brian (Goodman and Brette, 2008), NEURON (Carnevale and Hines, 2009) and GENESIS (Bower and Beeman, 2014).

Goals of the study

As hinted in the preceding sections the goal of our study is to investigate the idea that the

spontaneous contractions of the *Hydra* spp. occur by a process of self-organization, that is that they that they are an emergent feature, arising from the combination of the nerve net topology and cell mechanism. To do this we will develop a computational model of this network. Our approach is a heuristic one and is oriented towards the exploration of possibilities rather than the testing of predictions, its aim is to create the ground for further hypothesis-based research. Thus explicit fine tuning and scanning of the parameter space of the model will be employed.

To construct our computational nerve net the integrate-and-fire action potential model (Anderson, 2012) will be used. Although this model aims at reproducing the behavior of a biological neuron it does so in a simplified way. It ignores many of the biophysical variables involved in the generation of the action potential and it does not represent its actual shape treating it rather as a point event (Anderson, 2012; see also the Method section). Once again this highlights the heuristic nature of the study. Our aim may be finally characterized as the provision of a preliminary proof of concept of the self-organized mechanism for the spontaneous contractile behavior in the Hydra species.

Method

Materials

We constructed a computational model of the nerve net of the *Hydra* spp. For its development we used Brian 2 (Goodman and Brette, 2008) running on Anaconda Python 2.7.

Spatial structure

To create the structure of our model we first randomly distribute neurons across cylindrical coordinates to mimic the tubular shape of the Hydra nerve net. The cylinder has a height of 10 and a radius of 1 in dimensionless units, this size is arbitrary since there are no distance dependent variables in the model. We estimated that Hydra has about 880 neurons by counting them in microscope photographs of the animal from Koizumi (2016). David (2012) has reported that neurons are more concentrated in the head and base of Hydra. Thus we separated the tubular nerve net into three zones with different probabilities of a neuron being created in each of them: two zones at the edges of the net (defined by the first and last 1.5 units of its height) where the probability of creating a neuron was at 0.21 in each edge and one middle zone with the probability of creating a neuron at 0.58. This creates a well shaped distribution which is depicted in figure 4. We also made sure the neurons were not overlapping by prohibiting the creation of neurons with an euclidean distance smaller than 0.2 in the middle zone and with a distance smaller than 0.1 at the edge zones.

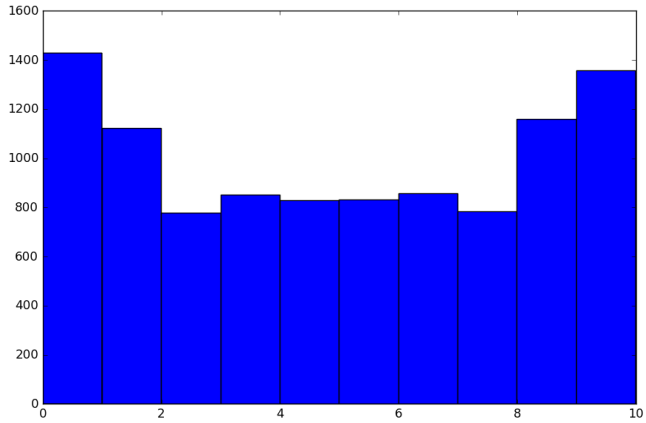

Figure 4. Well shaped probability distribution from which the probability for creating a neuron in different areas of the Hydra body is drawn, x-axis is the body length (unitless), sample size = 10000.

The next step is to create the synaptic connections. To do this we connect each neuron with all its neighbors with a euclidean distance smaller than 0.5 in the middle zone and with a distance smaller than 0.3 in the edge zones. All these synapses are bidirectional. Brian provides the functionality of

setting the probability of a specified synapse being created and this is one the parameters we varied in our simulations. Figure 5 shows a plot of the cylindrical nerve net with all the specified synapses formed.

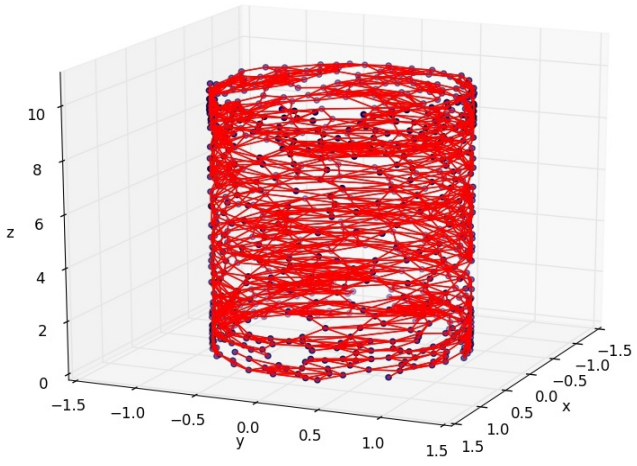

Figure 5. Plot of the spatial structure of the model with all synapses formed. The dot points represent neurons while the red lines represent the synapses. Note: the axes are scaled in respect to each other.

Dynamics

The evolution of the membrane potential for each neuron is governed by the integrate and fire (IF) equation (Anderson, 2012):

$$\tau \frac{dV(t)}{dt} = RI(t) - V(t) \quad (1)$$

Here V(t) is the membrane potential at time t, R the membrane resistance, I(t) the current that is inputted in the cell at time t and τ is the membrane time constant. For a constant input I(t) = I the natural behavior of the equation is for V(t) to asymptotically converge to the RI value. To make this behavior resemble an action potential a firing threshold (Vtr) is introduced. When this firing threshold is reached the neuron is considered to have fired and the voltage is pushed back to a reset value. Note that in the IF model the actual shape of the action potential is not reproduced, instead the action potential is treated as a point event. Aside from the firing threshold and the reset value Brian also offers the ability to make a neuron refractory by keeping it's voltage to to the reset value for a set amount of time. Figure 6 displays an example of the evolution of the membrane potential of an IF neuron.

For the Hydra model we set the RI(t) to a constant of a unitless 1. The τ was set at 70000 ms and the Vtr at 0.9987. The last two values were selected in order to slow down the firing rate of the neurons, a necessary move since the Hydra contractions occur only once every 5-10 minutes (Passano and McCullough, 1964). As is evident from equation (1) higher τ values decrease the rate of change of the membrane potential thus increasing the time it takes for the neuron to reach the Vtr. Putting the value of the firing threshold near the RI value was also critical in making the neuron slower due to the behavior of asymptotic functions which take relatively long times to assume values close to the asymptote. These optimal τ and Vtr values were decided by a manual search process where for successively increasing values of the time constant we examined the slowest firing rate that is allowed by the computational limitations that Python puts in the closest asymptotic value that Vtr can take. The reset value was set at 0 and the refractoriness value at 20 ms. Finally the voltage value for each neuron at the beginning of a simulation was randomly set with a value between 0 and Vtr. This creates the maximum possible range of differences in phase in the inherent

firing rate for each neuron.

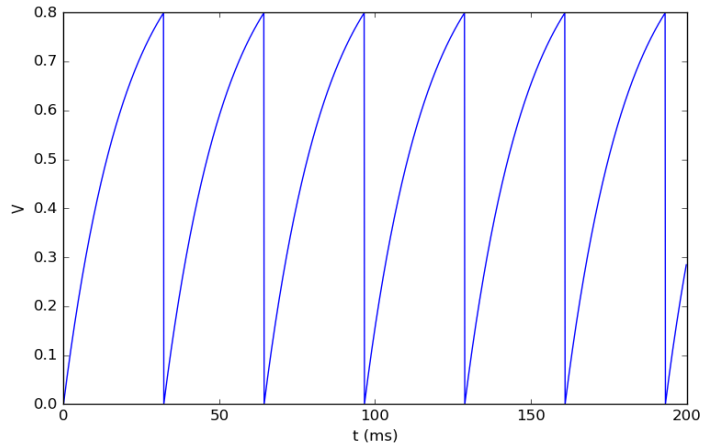

Figure 6. The time course of the membrane potential of an integrate and fire neuron, τ = 20 ms, RI(t) = 1, Vtr = 0.8, reset = 0.

Each time a neuron fires it increases the potential of the neurons with which it forms synaptic connections by an amount which was varied in our simulations. Brian offers the option to set the delay it takes for the synaptic input to take effect in the post-synaptic neuron, we also varied this parameter for our analysis.

Analysis

As was mentioned in the method section the fixed parameters of the model are: τ = 70000 ms, RI = 1 Vtr = 0.9987, reset value = 0 and the refractoriness equals 20 ms. To examine the behavior of the model we first run it by setting the probability to create a specified synapse (henceforth p(syn)) to 1, the delay to 2 ms and the weight of the synaptic connections to 0.15. For this simulation as well as for all the following ones the model variables were calculated every 1 ms by setting the Brian 2 clock accordingly. With these parameters the model falls into a periodic pattern of activity which is reminiscent of the spontaneous contractions of the *Hydra* spp. As is depicted in the raster plot of figure 7 the neurons fire in periodical columns with a period of around 7 minutes. Each of these columns in the raster plot consists of successive firings of subgroups of neurons with the whole episode lasting about 70-180 ms. Note that this time interval is much smaller than the contraction duration time that the Passano and McCullough (1964) electrophysiological measurements indicate (see figure 2).

Quantification of the network activity

The next step in the analysis was to examine the robustness of the above periodic and synchronized behavior of the Hydra model for a subregion of the parameter space. To achieve this a method to quantify this behavior was needed. For the quantification we first tried the Discrete Fourier Transform (DFT) a mathematical procedure which represents a signal that develops in time as a sum of sine waves with different phases and magnitudes (Luck, 2014; Park, 2009). High magnitudes concentrated in a narrow frequency band indicate frequencies that the time signal may actually contain (Luck, 2014).

To implement the DFT a segment of a simulation similar to that of the previous paragraph was selected with a temporal length of 2^{21} ms and from this we created a signal consisting of the counts of how many neurons had fired at each millisecond. The initial period where the system adjusts before falling into a stable periodic pattern (see figure 7) were not included in this 2^{21} ms segment. Then the mean of this signal was subtracted from each data point in the signal to remove the DC component (Park, 2009). Subsequently and in order to reduce spectral leakage the Hann window was applied to the signal (Park, 2009) which was then passed to the Fast Fourier Transform, an

algorithm for the performance of the DFT (Park, 2009). The output frequency spectrum looked like a valid representation of the input signal with a peak at the 0-20 Hz frequency range which could be interpreted to represent the slow frequency of the contraction episodes. Unfortunately the highest magnitude of this peak did not correspond the actual frequency of the appearance of the spiking columns which is around 0.0023 Hz (see figure 7) and so we abandoned this approach to quantification over concerns about its reliability when used with the model data.

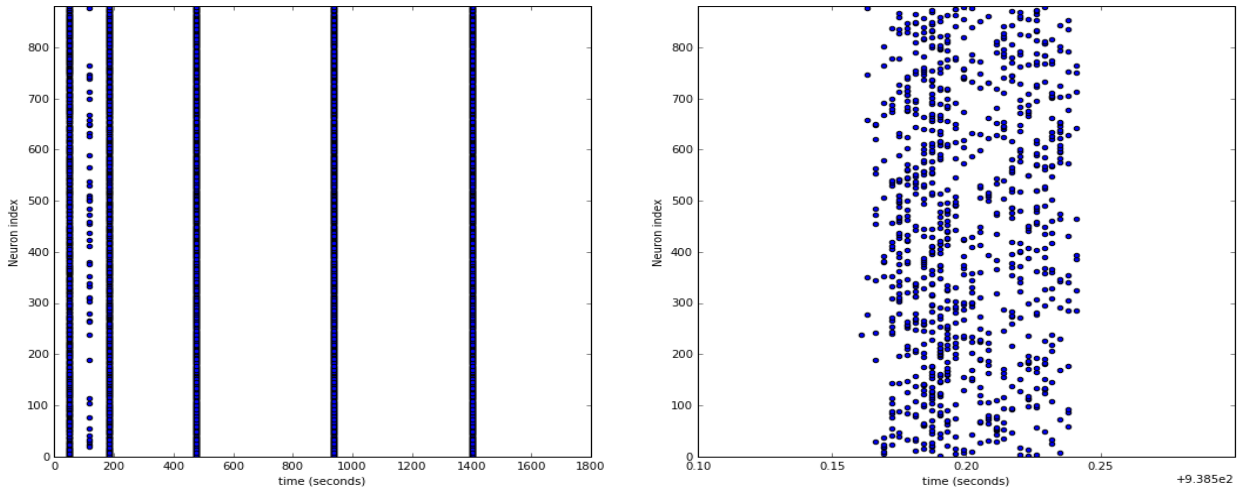

Figure 7. Left: Raster plot of a run of the Hydra model. x-axis is the time in seconds and y-axis is the index number of the neurons. p(syn) = 1, delay = 2 ms, synaptic weight = 0.15. Right: Close up of the spiking column taking place at 938 s.

We then turned to the SPIKE-distance (S-d). This is a metric which is specifically designed to detect synchronous activity in raster plot data (Mulansky and Kreuz, 2016). The S-d value interval is bounded between 0 and 1 with lower values signifying higher levels of synchrony (Mulansky and Kreuz, 2016). We should note that in contrast to the DFT this metric only signifies the presence of synchrony without giving any information about frequency values. To calculate the S-d its implementation in the PySpike python library was used (Mulansky and Kreuz, 2016).

Robustness analysis

For testing the robustness of the model's synchronized behavior we varied the p(syn) from 0.1 to 1 with a step of 0.1, the synaptic delay from 2 ms to 8 ms with a step of 2 ms and the synaptic weight from 0.1 to 0.9 with a step of 0.1 plus the Vtr since any value equal or above that always leads to exceeding the firing threshold. All possible combinations in this parameter space were examined and thus we ran in total 400 simulations, for each of those the SPIKE-distance was calculated. We decided not to use any inferential statistics since there is only one simulation run in each coordinate point of the parameter space and an exploration of a larger scope was beyond the goals of this project.

Figure 8 and table 1 depict the results of these simulations in a visual and numerical form respectively. Predictably at low p(syn) values the Hydra nerve net activity is desynchronized since most neurons fire independently and according to their inherent firing rate due to their activity not getting transferred to other cells. Despite this fact the network becomes even more desynchronized as a relatively small portion of the neurons becomes connected a situation which is apparent in the comparison between that case when p(syn) = 0.1 and the cases when p(syn) = 0.2 and p(syn) = 0.3. After the probability of forming a synapse reaches the 0.4 value the nerve net starts to become synchronized an effect which increases in magnitude as p(syn) increases.

For these cases in which the network is sufficiently connected to produce synchronized activity we observe that both higher values of delay and higher values of weight cause desynchronization. This behavior in not consistent, a fact that becomes apparent at higher levels of p(syn) where for high delay and weight values we observe both desynchronized and synchronized activity. To

explore this inconsistency further we selected the p(syn) = 1, delay = 8 ms and weight = 0.6 point in the parameter space and ran 20 simulations for it, each of those with different random values for the free model variables. In order for these simulations to be replicable we set the Numpy random seed at the value of 23. The seed was set only once, before the simulations begun, a practice justified because of the sequential and continuous way of running the simulations. The results are presented in table 2 and show that the system falls into either of two regimes: one with the network exhibiting a high level of synchronization ($N = 15$, $M = 0.0004$, $SD = 0.0005$) and one with it being desynchronized ($N = 5$, $M = 0.2269$, $SD = 0.0022$).

Thus for high delay and high weight values the possible nerve net activity is near the limit of a phase transition after which it comes desynchronized. Whether the limit is crossed depends on the non-fixed conditions of the model, that is the spatial structure of the nerve net and its synaptic connections which are randomly recreated for each simulation and the starting voltage value for each neuron which is also randomly set for each simulation (see the Method section).

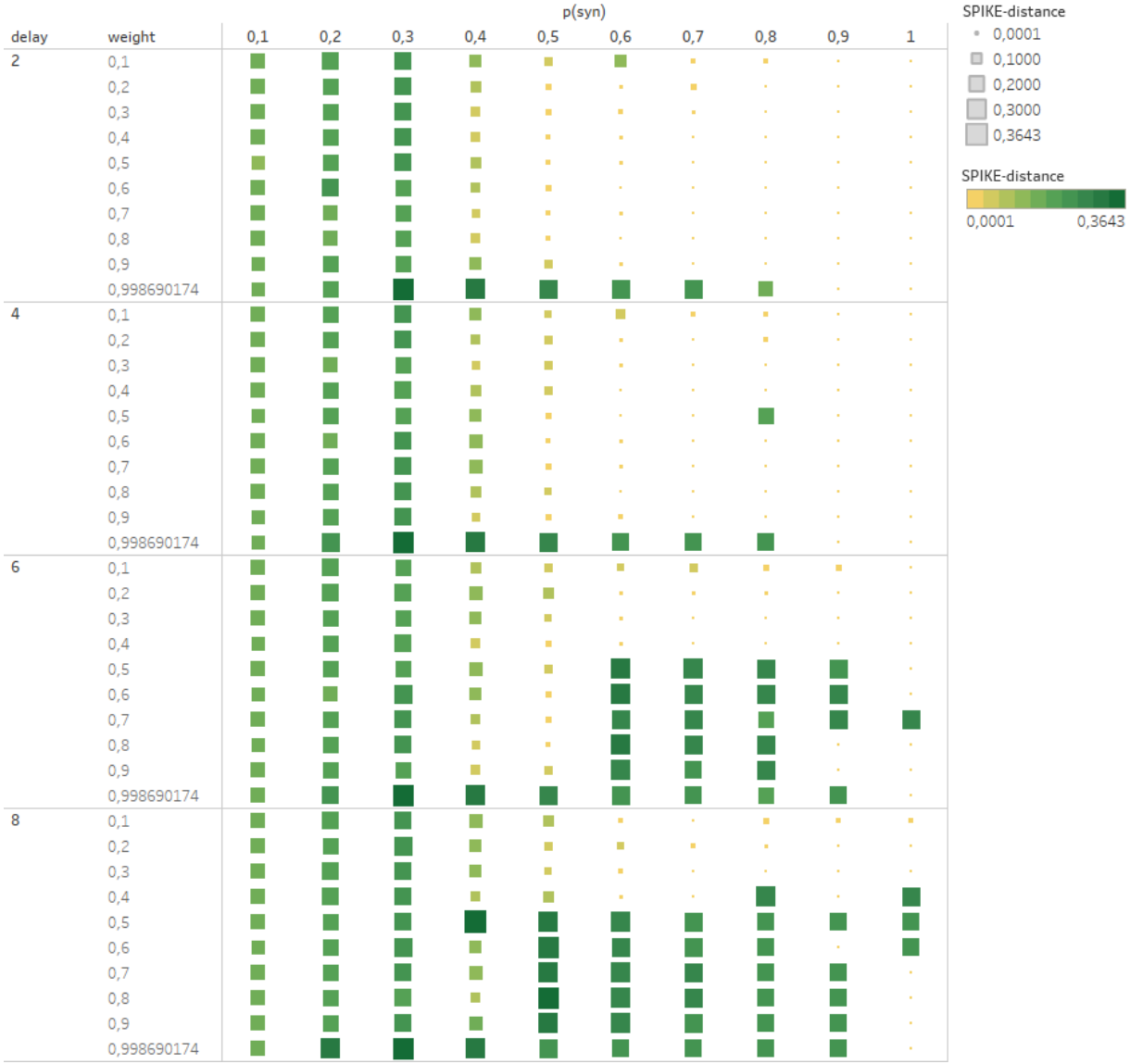

Figure 8. Synchronization levels (measured with the SPIKE-distance) for all the different combinations of the parameter space. Larger squares represent lower synchronization of the nerve net activity and the color code also reflects this. Rows list all the synaptic weight values for each value of synaptic delay (ms) while columns correspond to the probability of creating a synapse value. Yellow tinted colors indicate higher synchronization and green tinted colors lower synchronization.

delay	weight	p(syn) 0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
2	0,1	0,1598	0,2185	0,2294	0,1140	0,0560	0,1095	0,0165	0,0188	0,0049	0,0054
	0,2	0,1564	0,2055	0,2280	0,0968	0,0337	0,0104	0,0301	0,0010	0,0020	0,0027
	0,3	0,1652	0,1950	0,2303	0,0666	0,0302	0,0158	0,0084	0,0041	0,0011	0,0001
	0,4	0,1596	0,2094	0,2187	0,0707	0,0230	0,0091	0,0032	0,0002	0,0016	0,0001
	0,5	0,1439	0,1887	0,2217	0,0858	0,0229	0,0079	0,0015	0,0001	0,0001	0,0004
	0,6	0,1633	0,2206	0,2092	0,0778	0,0305	0,0059	0,0017	0,0001	0,0001	0,0001
	0,7	0,1567	0,1809	0,1943	0,0581	0,0233	0,0112	0,0017	0,0014	0,0012	0,0001
	0,8	0,1674	0,1624	0,2054	0,0702	0,0199	0,0056	0,0045	0,0001	0,0002	0,0001
	0,9	0,1502	0,1981	0,2088	0,1102	0,0557	0,0110	0,0008	0,0008	0,0001	0,0001
	0,998690174	0,1523	0,1841	0,3336	0,3095	0,2676	0,2511	0,2529	0,1760	0,0001	0,0009
4	0,1	0,1759	0,2113	0,2315	0,1215	0,0412	0,0688	0,0143	0,0192	0,0057	0,0050
	0,2	0,1608	0,1998	0,2346	0,0771	0,0505	0,0094	0,0030	0,0219	0,0008	0,0013
	0,3	0,1584	0,1770	0,1946	0,0523	0,0519	0,0093	0,0043	0,0038	0,0002	0,0016
	0,4	0,1545	0,1867	0,2140	0,0878	0,0550	0,0045	0,0036	0,0022	0,0010	0,0001
	0,5	0,1537	0,2004	0,1855	0,1128	0,0343	0,0014	0,0002	0,2115	0,0003	0,0005
	0,6	0,1584	0,1799	0,2215	0,1369	0,0213	0,0076	0,0005	0,0004	0,0008	0,0011
	0,7	0,1617	0,1881	0,2263	0,1380	0,0343	0,0078	0,0018	0,0005	0,0004	0,0001
	0,8	0,1567	0,1979	0,2293	0,1033	0,0492	0,0062	0,0026	0,0023	0,0001	0,0001
	0,9	0,1510	0,1983	0,2234	0,0493	0,0256	0,0171	0,0021	0,0009	0,0002	0,0001
	0,998690174	0,1506	0,2491	0,3454	0,2919	0,2642	0,2442	0,2427	0,2374	0,0001	0,0002
6	0,1	0,1714	0,2131	0,2023	0,0942	0,0561	0,0419	0,0509	0,0236	0,0237	0,0019
	0,2	0,1648	0,2164	0,2175	0,1351	0,0890	0,0138	0,0127	0,0137	0,0012	0,0048
	0,3	0,1669	0,2100	0,2011	0,1110	0,0424	0,0093	0,0031	0,0011	0,0010	0,0017
	0,4	0,1529	0,1882	0,2145	0,0708	0,0282	0,0130	0,0020	0,0016	0,0006	0,0001
	0,5	0,1603	0,2088	0,2027	0,1303	0,0573	0,3077	0,2810	0,2671	0,2532	0,0001
	0,6	0,1464	0,1802	0,2510	0,1206	0,0334	0,3138	0,2778	0,2571	0,2565	0,0005
	0,7	0,1750	0,2003	0,2394	0,0796	0,0283	0,2599	0,2783	0,2104	0,2631	0,2572
	0,8	0,1468	0,1904	0,2243	0,0573	0,0200	0,3099	0,2715	0,2604	0,0001	0,0001
	0,9	0,1679	0,1955	0,2065	0,0657	0,0506	0,2843	0,2244	0,2656	0,0013	0,0001
	0,998690174	0,1566	0,2274	0,3537	0,3083	0,2603	0,2546	0,2389	0,1937	0,2406	0,0001
8	0,1	0,1806	0,2134	0,2245	0,1338	0,0938	0,0206	0,0062	0,0302	0,0160	0,0152
	0,2	0,1728	0,2110	0,2471	0,1099	0,0608	0,0401	0,0222	0,0080	0,0035	0,0001
	0,3	0,1608	0,2142	0,2254	0,1183	0,0403	0,0156	0,0018	0,0037	0,0010	0,0006
	0,4	0,1635	0,2154	0,2303	0,0830	0,0857	0,0139	0,0020	0,2906	0,0006	0,2730
	0,5	0,1568	0,2007	0,2296	0,3643	0,3080	0,2837	0,2524	0,2367	0,2208	0,2242
	0,6	0,1502	0,1944	0,2447	0,1158	0,3193	0,2707	0,2522	0,2325	0,0004	0,2244
	0,7	0,1734	0,2033	0,2416	0,1347	0,3097	0,2814	0,2603	0,2400	0,2297	0,0029
	0,8	0,1558	0,1975	0,2149	0,0833	0,3348	0,2821	0,2649	0,2386	0,2334	0,0002
	0,9	0,1711	0,2113	0,2398	0,1486	0,3153	0,2861	0,2529	0,2289	0,2303	0,0002
	0,998690174	0,1635	0,3091	0,3290	0,2932	0,2489	0,2398	0,2197	0,2203	0,2220	0,0002

SPIKE-distance: 0,0001 – 0,3643

Table 1. SPIKE-distance values for all the different combinations of the parameter space. Higher values indicate lower synchronization of the nerve net activity and the color code also reflects this. Rows list all the synaptic weight values for each value of synaptic delay (ms) while columns correspond to the probability of creating a synapse value. Yellow tinted colors indicate higher synchronization and green tinted colors lower synchronization.

Simulation	S-d	Simulation	S-d
1	0,0002	11	0,0001
2	0,0003	12	0,0001
3	0,0002	13	0,2255
4	0,0008	14	0,0002
5	0,2264	15	0,0019
6	0,2294	16	0,0002
7	0,0002	17	0,0001
8	0,0010	18	0,2291
9	0,2243	19	0,0001
10	0,0002	20	0,0001

Table 2. Twenty repeated simulations for p(syn) = 1, delay = 8 ms and weight = 0.6. The nerve net activity falls either in a synchronized or a desynchronized regime depending on the initial conditions. Measurement values are in the SPIKE-distance metric, the Simulation column lists the serial number of each simulation and the color code reflects the regime the system is in: yellow for synchronized and green for desynchronized.

Discussion

In this project we explored the possibility of a self-organized mechanism for the periodic contractile behavior of the *Hydra* spp. by using computational methods. We found that at least for a specific set of initial conditions the model we developed reproduces the contractions of Hydra at the appropriate frequency and that irrespective of frequency the synchronized behavior persists in a significant region of the model's parameter space. Thus we managed to successfully provide preliminary evidence for self-organized contractions in these animals. Since these occur without any explicit rules our findings challenge suggestions that have been advanced in the literature of the contractions being driven by pacemaker cells (Passano and McCullough, 1964; Takaku et al., 2014).

Our work has several shortcomings that indicate areas of possible improvement. To begin with the Hydra model is of a heuristic nature and is based on a rather gross model of the neuron with the result of reduced biological realism. We also modeled the nerve net as a single cylinder while the actual structure is a double cylinder with neurons both in the epidermis and the gastrodermis of the animals (Johnson, 2003; Swanson, 2012) and this further reduces the realism. Furthermore each contraction episode in the model lasts far shorter than what electrophysiological data gathered from Hydras suggest (Passano and McCullough, 1964; see figure 2) and the model leaves out other aspects of the contractile behavior such as the as the sensitivity of the contraction frequency to light stimuli and illumination levels (Passano and McCullough, 1964). Finally the project was an exploratory endeavor.

Based on the above observations suggestions for future research can be made. To begin with more realistic models of the neuron could be used, for example the Hodgkin-Huxley equations that reproduce the shape of the action potential and take into account more of the related biophysical variables (Anderson, 2014). Reproducing the double cylindrical structure of the Hydra nerve net, incorporating more aspects of the contractile behavior and having an appropriate contraction duration length can also increase the realism and the scope of the model. The parameters of future models should be a-priori and empirically set thus moving the research into the hypothesis-testing domain.

The self-organized Hydra model may possibly be developed to encompass other motor behaviors of these animals such as feeding. Relevant to this kind of extension is the finding that for a region of the model's parameter space at which there were high synaptic delays and high synaptic weights the model exhibited two different phases (i.e., modes of activity). One phase in which the model goes through periodic contractions and one in which there is no such synchronized behavior. In which of these two phases the system falls in depends on two factors: a) on the random starting voltage value for each neuron and b) in the random spatial structure of the model including its random synaptic connections. Both of these factors are defined anew for each simulation. The desynchronized phase could be probed further to see if its activity is random or patterned. Should patterns be observed then their possible relation to Hydra behaviors can be examined.

We hope that by providing a proof of concept of a self-organized mechanism in Hydra we have been able to demonstrate that continuing this line of research is worth the effort. Such research can contribute to the general understanding of the Hydra nervous structures along with their activity and since the *Hydra* spp. occupy an early phylogenetic position and are computationally tractable (Bosch et al., 2017) it may ultimately help in the quest to uncover the functions and origins of the nervous system.

References

Anderson, B. (2014). *Computational Neuroscience and Cognitive Modelling* (1 edition). SAGE Publications Ltd.

Bosch, T. C. G., Klimovich, A., Domazet-Lošo, T., Gründer, S., Holstein, T. W., Jékely, G., … Yuste, R. (n.d.). Back to the Basics: Cnidarians Start to Fire. *Trends in Neurosciences*, *40*(2), 92–105. https://doi.org/10.1016/j.tins.2016.11.005

Bower, J. M., & Beeman, D. (2014). *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System* (2nd ed). Springer.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraula, G., & Bonabeau, E. (2003). *Self-Organization in Biological Systems* (1st Edition). Princeton, N.J Woodstock: Princeton University Press.

Carnevale, N. T., & Hines, M. L. (2009). *The NEURON Book* (1 edition). Cambridge: Cambridge University Press.

David, C. N. (2012). Interstitial stem cells in Hydra: multipotency and decision-making. *The International Journal of Developmental Biology*, *56*(6–8), 489–497. https://doi.org/10.1387/ijdb.113476cd

Goodman, D. F. M., & Brette, R. (2008). Brian: a simulator for spiking neural networks in Python. *Frontiers in Neuroinformatics*, *2*.

Johnson, G. B. (2003). *The Living World*. Boston: Mcgraw-Hill.

Koizumi O. (2016). in *The Cnidaria, Past, Present and Future* (1st ed.). New York, NY: Springer.

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique* (second edition edition). Cambridge, Massachusetts: A Bradford Book.

Mulansky, M., & Kreuz, T. (2016). PySpike—A Python library for analyzing spike train synchrony. *SoftwareX*, *5*, 183–189. https://doi.org/10.1016/j.softx.2016.07.006

Park, T. H. (2009). *Introduction To Digital Signal Processing*. Hackensack, NJ: World Scientific Publishing Company.

Passano L.M., & McCullough C.B. (1964). Co-ordinating systems and behaviour in Hydra I., *J. Exp. Biol., 31*, 643-664

Swanson L. (2012). in *Fundamental Neuroscience, Fourth Edition* (4th edition). Amsterdam ; Boston: Academic Press.

Takaku, Y., Hwang, J. S., Wolf, A., Böttger, A., Shimizu, H., David, C. N., & Gojobori, T. (2014). Innexin gap junctions in nerve cells coordinate spontaneous contractile behavior in Hydra polyps. *Scientific Reports*, *4*, 3573.

Appendix: Source code of the model

```
from pandas import *
from scipy.spatial import distance
from mpl_toolkits.mplot3d import Axes3D
import pyspike as psp
from brian2 import *
prefs.codegen.target = 'cython'
start_scope()

def wellProbability(edgesProb):
    '''Creates a well shaped probability distribution
    edgesProb is the probability at each of the two edges of the well
    NOTE: edgesProb values must be greater than 0.15 and with a maximum of 0.5'''
    if(edgesProb > 0.15 and edgesProb <= 0.5):
        middleProb = 1-(edgesProb*2)
        distributionValues = arange(0,1,0.001)
        pValues = repeat(edgesProb/150,150)
        pValues = append(pValues,repeat(middleProb/700,700))
        pValues = append(pValues,repeat(edgesProb/150,150))
        return choice(a=distributionValues,size=1,p=pValues)
    else:
        raise ValueError('edgesProb must greater than 0.15 and with a maximum of 0.5')

def createColumn(neuronalGroup,populationSize,columnLength):
    """Creates a cylindrical column of neurons
    neuronalGroup is a NeuronGroup object
    populationSize is the number of neurons in the column
    columnLength is the length of the column in arbitrary units
    NOTE: requires scipy.spatial/distance"""
    #adds attribute for storing the position vectors of the neurons
    neuronalGroup.add_attribute('positionVectors')
    #initialising the attribute as an array
    neuronalGroup.positionVectors = zeros((populationSize,3))
    #variables for the current provisional coordinates
    xcor = 0.0
    ycor = 0.0
    zcor = 0.0 #z is the longitudinal axis

    for i in range(0,populationSize,1):
        #while loop compares distances between neurons to make sure they are
        #not too close to one another
        tooClose = True
        while tooClose == True:
            tooClose = False
            #creation of random coordinates
            zcor = (wellProbability(0.21)*columnLength) #random length value
            angle = (rand()*(2*pi))  #random angle
            xcor = (cos(angle)) #unpacks x coordinate from angle
            ycor = (sin(angle)) #likewise for y

            if i == 0: # in this case just store the cooordinates
```

```
            neuronalGroup.positionVectors[i][0] = xcor
            neuronalGroup.positionVectors[i][1] = ycor
            neuronalGroup.positionVectors[i][2] = zcor
         else:
            #otherwise compare euclidean distances with all existing neurons
            #minimum allowed distance depends on the z-axis position of the
            #current neuron's provisional coordinates
            currentVector = [xcor,ycor,zcor] #current neuron's position vector
            for j in range(i):
               euDistance = distance.euclidean(neuronalGroup.positionVectors[j],currentVector)
               if currentVector[2] > 1.5 and currentVector[2] < 8.5:
                  if abs(euDistance) < 0.2:
                     tooClose = True #neurons too close, while loop will try again
                     break
               elif abs(euDistance) < 0.1:
                  tooClose = True
                  break
            if tooClose == False:
               #if distance ok stores coordinates
               neuronalGroup.positionVectors[i][0] = xcor
               neuronalGroup.positionVectors[i][1] = ycor
               neuronalGroup.positionVectors[i][2] = zcor
   return neuronalGroup

def plotColumn(neuronalGroup,synapseGroup=None):
   """Creates a 3D scatterplot of the column with lines indicating the
   synaptic connections (if available)
   neuronalGroup is a NeuronGroup object with coordinates created by
   createColumn()
   synapseGroup (optional) is a Synapses object"""
   pV = neuronalGroup.positionVectors
   figure1 = figure('Column')
   axes3d = figure1.add_subplot(111, projection='3d')
   axes3d.set_xlim(-1.5,1.5)
   axes3d.set_ylim(-1.5,1.5)
   axes3d.set_zlim(0,11)
   axes3d.scatter(xs=pV[:,0],ys=pV[:,1],zs=pV[:,2])
   axes3d.set_xlabel('x')
   axes3d.set_ylabel('y')
   axes3d.set_zlabel('z')

   if(synapseGroup != None): #plots the synaptic connections
      synArray = array([synapseGroup.i,synapseGroup.j])
      synArray = synArray.T
      for i in range(len(synArray)):
         axes3d.plot(xs=[pV[synArray[i,0],0],pV[synArray[i,1],0]],
                  ys=[pV[synArray[i,0],1],pV[synArray[i,1],1]],
                  zs=[pV[synArray[i,0],2],pV[synArray[i,1],2]],color='r')

def createSynapses(neuronalGroup,populationSize,synapticWeights,pValue,synapticDelay):
   '''Creates the synaptic connections in a column
   neuronalGroup is a NeuronGroup object
```

```
    populationSize is the number of neurons in the column
    synapticWeight is the weight of the synapses
    pValue is the probability of creating each synapse
    NOTE: requires scipy.spatial/distance'''
    if(populationSize > 1):
        #Creates the object that contains the synapses and their operations
        S = Synapses(neuronalGroup,neuronalGroup,'w:1',on_pre='v_post += w',
                delay=synapticDelay*ms)
        currentNeuron = 0
        pV = neuronalGroup.positionVectors

    #while loop creates a bidirectional synapse if the distance between two
    #neurons is smaller than a specified distance that depends on the z-axis
    #position of the current neuron
        while currentNeuron < (populationSize-1):
            for neighbor in range(currentNeuron+1,populationSize,1):
                euDistance = distance.euclidean(pV[neighbor],pV[currentNeuron])
                if pV[currentNeuron,2] > 1.5 and pV[currentNeuron,2] < 8.5:
                    if abs(euDistance) < 0.5:
                        S.connect(i=currentNeuron,j=neighbor,p=pValue)
                        S.connect(i=neighbor,j=currentNeuron,p=pValue)
                elif abs(euDistance) < 0.3:
                    S.connect(i=currentNeuron,j=neighbor,p=pValue)
                    S.connect(i=neighbor,j=currentNeuron,p=pValue)
            currentNeuron += 1

        S.w = synapticWeights #sets the synaptic weights
        return S
    else:
        print('createSynapses: Population size must be greater than 1')

def fftSignal(spikeTimes,runTimeTotal,adjustmentTime):
    '''Creates a signal suitable for FFT analysis
    spikesTimes: a SpikeMonitor.t object
    runTimeTotal: the total time of the simulation
    adjustmentTime: the time given to the network for adjustment
    NOTE: runTime should be a power of two
    NOTE: requires itertools/groupby'''
    #creates a dictionary with the times where firings occured as keys and
    #the counts of the firings at each of these times
    spikeTimes = array(spikeTimes/second)
    spikeTimes = list(spikeTimes)
    spikeTimes.sort()
    times = set(spikeTimes)
    times = list(times)
    for i in range(len(times)):
        times[i] = round(times[i],3)
    times.sort()
    for i in range(len(spikeTimes)):
        spikeTimes[i] = round(spikeTimes[i],3)
    freqs = []
    for i in times:
```

```
        freqs.append(spikeTimes.count(i))
    spikes = dict(zip(times,freqs))

    #fills the spikes dictionary with the rest of the zero valued times and
    #the associated zero valued counts
    timeArray = [x * 0.001 for x in range(adjustmentTime,runTimeTotal)]
    for i in range(len(timeArray)):
        timeArray[i] = round(timeArray[i],3)
    for i in timeArray:
        if i not in spikes:
            spikes[i] = 0

    #sorts and seperates the keys and the values and returns a list containing
    #them
    times = []
    freqs = []
    for key in sorted(spikes.iterkeys()):
        times.append(key)
        freqs.append(spikes[key])
    return([times,freqs])

def fftAnalysis(freqs,sRate):
    '''FFT analysis, returns the frequency bins and the associated power
    freqs: an array with the counts of firings at each sampling point
    sRate: the sampling rate based on the defaultclock.dt value
    NOTE: length of freqs should be a power of two
    code based on: https://plot.ly/matplotlib/fft/'''
    Fs = float(sRate);  #sampling rate
    Ts = 1.0/Fs; #sampling interval

    n = len(freqs) #length of the signal
    k = arange(n)
    T = n/Fs
    frq = k/T #two sides frequency range
    frq = frq[range(n/2)] #one side frequency range

    Y = fft(freqs)/n #fft computing and normalization
    Y = Y[range(n/2)]

    figure1 = figure('Spectrum')
    fplot = figure1.add_subplot(111)
    fplot.plot(frq,abs(Y),'r') #plotting the spectrum
    fplot.set_xlabel('Freq (Hz)')
    fplot.set_ylabel('|Y(freq)|')
    show('Spectrum')
    return([frq,abs(Y)])

#########################################################################
populationSize = 880 #number of neurons
columnLength = 10 #column length in arbitrary units
Vt = float64(0.998690173613014) #threshold for neuron firing
#synapticWeights = 0.15 #weight of synaptic connections
```

```
#synapticDelay = 2 #value of synaptic delay (in ms)
defaultclock.dt = 1*ms #sets the time step for the simulation(s)
runTime = 1800000 #duration of the to be analysed simulation (in ms)


eqs = '''
dv/dt = (I-v)/tau : 1 (unless refractory)
I : 1
tau : second
'''

synwArray = arange(0.1,1,0.1)
synwArray = append(synwArray,Vt)
cl = ['tau','delay','p(syn)','weight','SPIKEd','ISId','SPIKEsync']
qvalues = DataFrame(zeros((400,7)),columns=cl)
currentLine = 0
for tc in [70000]:
    for delay in (range(2,9,2)):
        for psyn in arange(0.1,1.1,0.1):
            for synw in synwArray:
                G1 = NeuronGroup(populationSize,eqs,threshold='v>Vt',reset='v=0',refractory=20*ms)
                G1.I = 1
                G1.tau = tc*ms
                G1.v = 'rand()*Vt'
                G1 = createColumn(G1,populationSize,columnLength)
                S1 = createSynapses(G1,populationSize,synw,psyn,delay)
                Sp1 = SpikeMonitor(G1)

                run(runTime*ms)

                #code below calculates and stores the pyspike metrics
                firingValuesWithUnits = Sp1.spike_trains().values()
                firingValues = []
                for i in range(len(firingValuesWithUnits)):
                    firingValues.append(array(firingValuesWithUnits[i]))
                fV = open('fv.txt','w')
                for item in firingValues:
                    item = (" ".join(map(str,item)))
                    fV.write("%s\n" % item)
                fV.close()
                spikeTrains = psp.load_spike_trains_from_txt("fv.txt",edges=(0,runTime/1000.0))
                qvalues.iloc[currentLine,0] = tc
                qvalues.iloc[currentLine,1] = delay
                qvalues.iloc[currentLine,2] = psyn
                qvalues.iloc[currentLine,3] = synw
                qvalues.iloc[currentLine,4] = psp.spike_distance(spikeTrains)
                qvalues.iloc[currentLine,5] = psp.isi_distance(spikeTrains)
                qvalues.iloc[currentLine,6] = psp.spike_sync(spikeTrains)
                currentLine += 1

                del G1
                del S1
```

```
        del Sp1
        del firingValuesWithUnits
        del firingValues
        del spikeTrains

qvalues.to_excel('qvalues.xlsx', sheet_name='Sheet1')
```