Modelling Circadian Rhythms in Nengo

Department of Psychology, University of Waterloo

Authors: Natasha Thomare, Parmandeep Chaddha, Saad Arif Qadeer

Course: PSYCH 420: An Introduction to Computational Neuroscience Methods

Instructor: Dr. Britt Anderson

Submission Date: April 12, 2022

Submission Location: LEARN Dropbox

Table of Contents

1.0 An Introduction to Nengo

Modelling the human brain is a significant challenge that many researchers are attempting to solve. A detailed model of a region of the brain will give us imitations of different neural activities such as neuron spikes and action potentials (Humphries, 2021). The more accurate the neural activity, the more realistic the model becomes. By modelling parts of the brain or its processes, we can test mechanistic hypotheses to improve our understanding of human behaviour. This leads to the possibility of virtual experiments which can be done faster and more ethically compared to experimenting on biological life (Humphries, 2021). It also leads to a greater amount of control as scientists gain the ability to manipulate certain neurons to observe virtual behavior changes.

The Neural Engineering Framework (NEF) is a method that can be used to create neural simulations (CNRG Lab, n.d.). It can showcase neural representations of objects such as vectors, scalars, and functions. NEF can incorporate populational (signals of joint activities of multiple neurons) and temporal (focuses on the exact timing of action potentials) encoding of neurons to compute linear and nonlinear functions (CNRG Lab, n.d.). A significant challenge for the NEF is the ability to build larger-scale models. Nengo is a software that can build large-scale models using the NEF foundation.

The first version of Nengo was created in 2003 by Chris Eliasmith and Charles H. Anderson, which details a framework for generating models of spiking neurons and provided a collection of MATLAB scripts called NESim (Neural Engineering Simulator) (Bekolay et al., 2014). There was also a basic Graphical User Interface (GUI) which users could use to set the parameters of their model.

Presently Nengo's application programming interface can run various models on the backend and deliver an improved front-end experience (Bekolay et al., 2014). It utilizes a NumPy-backed reference simulator. It is capable of building large models and interacting with other neuroscience packages written in Python.

Some present-day applications of Nengo include deep learning using recurrent neural networks and online learning. Deep learning is a sub section of machine learning which focuses on algorithms based on artificial neural networks. Online Learning is a form of artificial learning that updates network parameters with each data point.

A previous use of Nengo was creating a simulation for an inverted pendulum which is a pendulum that has its center of mass above its pivot point. It is inherently unstable and must be kept stable by using a control system to monitor the angle of the pole and move the pivot point horizontally back under the center of mass when it starts to fall over, keeping it balanced. This can be done by applying a torque at the pivot point.

Figure 1: An example of an inverted pendulum, the pivot point is able to move horizontally balancing the pendulum (Wikimedia, 2022).

This example demonstrates the use of an adaptive controller to control a weighted inverted pendulum that is affected by gravity. The controller used is a combination of a standard Proportional-Derivative (PD) controller with an adaptive neural ensemble. Together, these two components implement a Proportional-Integral-Derivative (PID) controller, where the integral term is adaptive and compensates for changing steady-state error as gravity acts on the system.

There are two inputs that the user can control the target pendulum angle which is the desired angle of the pendulum and the mass of the pendulum. In the simulation which uses the adaptive controller or PID controller, the actual position of the pendulum converges to that of the target angle. In comparison the same pendulum when controlled by a standard PD controller the actual position of the pendulum never converges to the desired target angle.

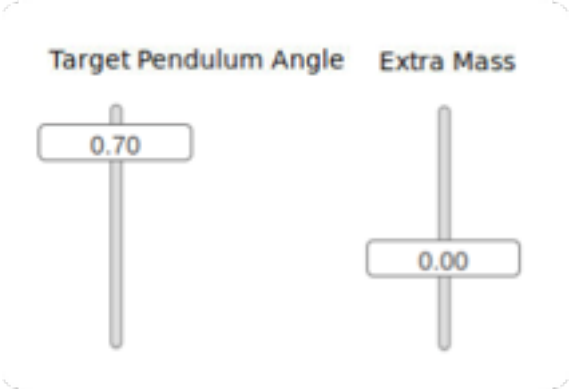

Figure 2: User inputs of Target Pendulum Angle and mass of pendulum (Nengo, n.d.).

2.0 An Introduction to Circadian Rhythms

Our example focused on the application of Nengo on the biological system of circadian rhythms. Circadian rhythms are oscillations with a period of 24 hours inherent in numerous biological processes such as cognitive activities, sleep, and periods of alertness (Forteinou, 2018). They allow the ability to regulate the cycles of activity on normative schedules. In biology, circadian rhythms are implemented using the multiple phosphorylation forms of the PER protein and the negative feedback exerted by the protein on the transcription gene. PER produces rhythmic behaviour because PER-mRNA itself varies in a sinusoidal 24-hour manner (Goldbeter, 1995). The negative feedback exerted by the PER protein has a delay before it effects the PER gene creating cellular oscillations (Goldbeter, 1995). The goal of this computational model is to better understand the functioning of a mechanism whose parts, operations, and organization already have been determined.

Goldbeter's model of circadian rhythm oscillations is pictured in Figure 3. This map describes the path of PER in its raw form and the route from the nucleus into the cytosol. The associated variables are rates of degradation, conversion, accumulation and or phosphorylation.

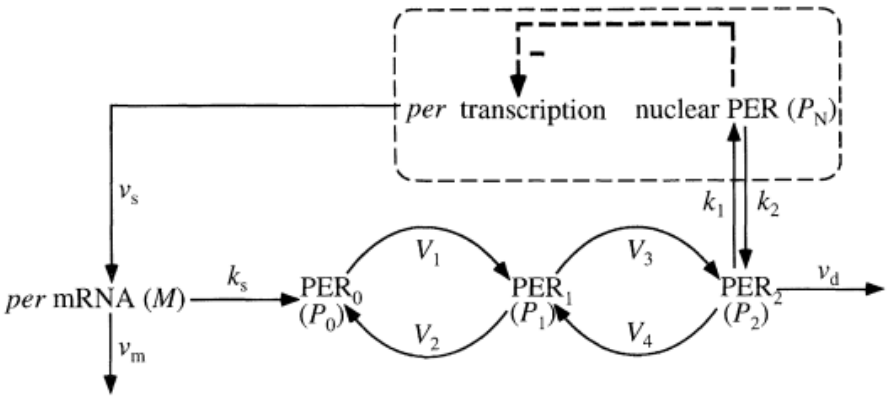

Figure 3: Scheme of the model for circadian oscillations in PER and per mRNA (Goldbeter, 1995).

The individual components are calculated through a system of first order differential equations (Figure 4) that show the rate of accumulation of cytoplasmic per-mRNA over time to produce the oscillations seen in the computational model. Overall, the basic idea of each equation in this group is that the rate of accumulation equals the rate of production minus the rate of degradation.

$$\frac{dM}{dt} = v_s \frac{K_I^n}{K_I^n + P_N^n} - v_m \frac{M}{K_m + M} \quad (1)$$

$$\frac{dP_0}{dt} = k_s M - V_1 \frac{P_0}{K_1 + P_0} + V_2 \frac{P_1}{K_2 + P_1} \quad (2)$$

$$\frac{dP_1}{dt} = V_1 \frac{P_0}{K_1 + P_0} - V_2 \frac{P_1}{K_2 + P_1} - V_3 \frac{P_1}{K_3 + P_1} + V_4 \frac{P_2}{K_4 + P_2} \quad (3)$$

$$\frac{dP_2}{dt} = V_3 \frac{P_1}{K_3 + P_1} - V_4 \frac{P_2}{K_4 + P_2} - k_1 P_2 + k_2 P_N - v_d \frac{P_2}{K_d + P_2} \quad (4)$$

$$\frac{dP_N}{dt} = k_1 P_2 - k_2 P_N \quad (5)$$

Figure 4: Goldbeter's Model of Circadian Rhythms (Goldbeter,1995)

Equation 1 is the most important equation out of the group. In this equation, M is the accumulation of cytoplasmic per-mRNA. V_s is the maximum rate for the accumulation of M. Next, the K_i (1-4) is the threshold of constant inhibition. Following that is the P_n, which is the raw concentration of PER in the nucleus. The "n" variable is the Hill coefficient which describes the minimum number of cooperating molecules required to achieve inhibition. Next, v_m is the maximum rate for the degradation of M. Finally, K_M is the Michaelis constant for the degradation

reaction. Table 1 summarizes the parameter values used by Goldbeter et al to generate their model.

Table 1: Parameter Values used in Goldbeter et al's Model

Parameter Name	Parameter Value	Units
vs	0.76	nM/hr
vm	0.65	nM/hr
vd	0.95	nM/hr
Km	0.5	nM
Kt	1	nM
Kd	0.2	nM
K1, K2, K3, K4	2	nM
ks	0.38	hr^{-1}
k1	1.9	hr^{-1}
k2	1.3	hr^{-1}
V1	3.2	nM/hr
V2	1.58	nM/hr
V3	5	nM/hr
V4	2.5	nM/hr
n	4	(Hill coefficient)

3.0 Modelling Circadian Rhythms using Nengo

Multiple models of circadian rhythms were constructed using the Nengo. The first model only represented Equation 1 in Figure 4. The other parameters were declared as constants and the changes in the PER-mRNA were potted with time. The next model attempted to replicate the complete circadian rhythm model generated by Goldbeter et al, as pictured in Figure 3. Results from both models and discussed along with possible avenues for improvement.

3.1 Simplified Model of Circadian Rhythms

A simplified model of circadian rhythms was created by holding the P0, P1, and P2 concentrations constant. This results in the model presented in Figure 5. In this mode, only the

nuclear-PER concentration is varied using a sinusoidal function that mimic the nuclear-PER concentration variations in biology. As such, this model results in a *dampening oscillator* system where PER MRNA concentration values are a sum of the current PN concentration (multiplied by some factor) and the previous MRNA concentration (multiplied by some factor and some decay rate).

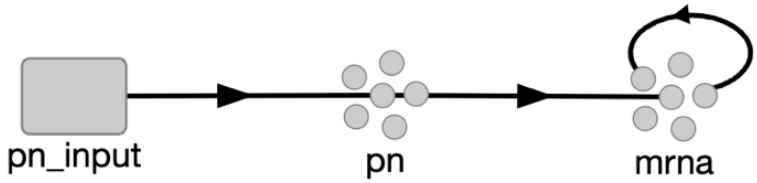

Figure 5: Simplified Model of Circadian Rhythms

The nuclear-PER concentration is varied using a simple sine function. This value is fed into an *ensemble* of artificial neurons, titled *pn* in Figure 5, where some neurons are fired based on their *tuning curves*. (The tuning curves are automatically generated by Nengo). The output of the *pn* ensemble of neurons is transmitted into another artificial nucleus, named *mrna*, that records the PER-mRNA concentration. The *mrna* nucleus functions as an integrator, and continuously updates the PER-mRNA concentration, based on a sum of the signal from the *pn* nucleus, and the current PER-mRNA concentration. The script used to generate the basic model is pictured in Figure 6.

```
def create_model():
    """ Create a mock model of the circadian rhythm.
    Returns the model, and probes of the model.
    """
    model = nengo.Network()
    with model:
        # Create the pn, mrna ensemble of neurons.
        pn = nengo.Ensemble(n_neurons=2000, dimensions=1, radius=2)
        mrna = nengo.Ensemble(n_neurons=2000, dimensions=1, radius=2)

        # Create a sinusodial input for the pn ensemble, and connect the stimulus to the
ensemble.
        pn_input= nengo.Node(lambda t: sin(t) + 1)
        nengo.Connection(pn_input, pn)

        # Connect the `pn` ensemble to the `mrna` ensemble.
        def pn_to_mrna(v):
            return 0.5 * (1 / (1 + v))
        nengo.Connection(pn, mrna, function=pn_to_mrna, synapse=0.5)

        # Connect the `mrna` nucleus to itself. This is basic integrator architecture.
        def mrna_integrator(c):
            return -0.5 * (c / (1 + c))
        nengo.Connection(mrna, mrna, function=mrna_integrator, synapse=0.5)

        # Create two probes to measure the nucleur-PER concentration and PER MRNA concentration.
        pn_probe = nengo.Probe(pn)
        mrna_probe = nengo.Probe(mrna)
        return model, {
            "pn_probe": pn_probe,
            "mrna_probe": mrna_probe
        }
```

Figure 6: Mock model of Circadian Rhythms. The function titled `pn_to_mrna` is approximated as $\frac{0.5}{1+P_n}$, whereas the function titled `mrna_integrator` is approximated as $-0.5c/(1+c)$. This is based on Equation 1 (Figure 4), with all values except P_n and M being held constant.

The results from this experiment display a harmonic variation in the PER-mRNA concentration, as expected. The *left* graph in Figure 7 displays the change in the PER-mRNA with time and the *right* graph plots the change in PER-mRNA concentration as the nuclear-PER concentration changes. The later graph also displays the globular variation in the two parameters,

regardless of the starting position of the two concentrations. In fact, that's the regulatory nature of the circadian rhythm in action!

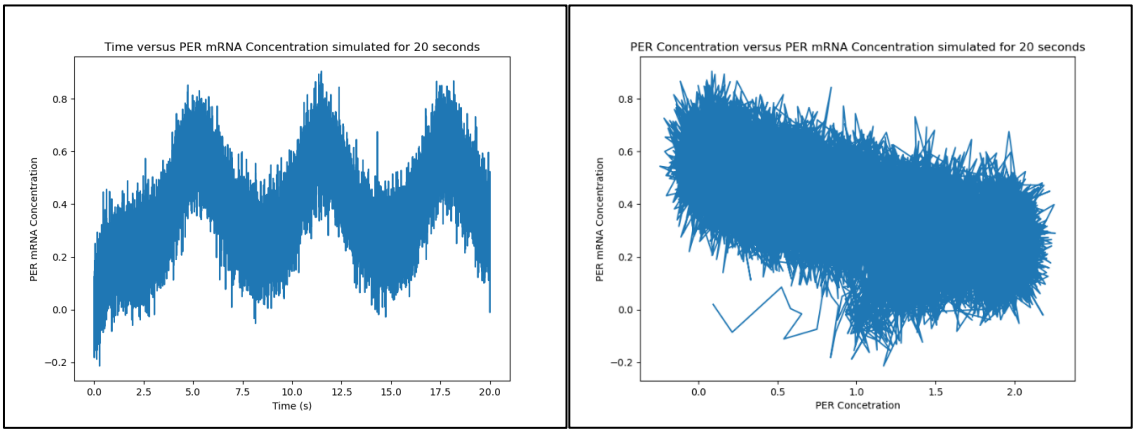

Figure 7: (*left*) Change in PER mRNA concentration with time. (*right*) PER mRNA concentration versus nuclear PER concentration.

3.2 Goldbeter's Model of Circadian Rhythms

Goldbeter's model of circadian rhythm in *drosophilia* was re-created using the Nengo software, as pictured in Figure 8. Although there are numerous similarities between this model and the model presented in Figure 3, there is a key *architecture* difference. Each "node" in the Nengo architecture is split into two components: the actual protein component and the derivative component. The protein component for each form (eg. P0) connects to the derivative of the next (P1) and previous (mRNA) component as determined by the set of differential equations presented in Figure 4.

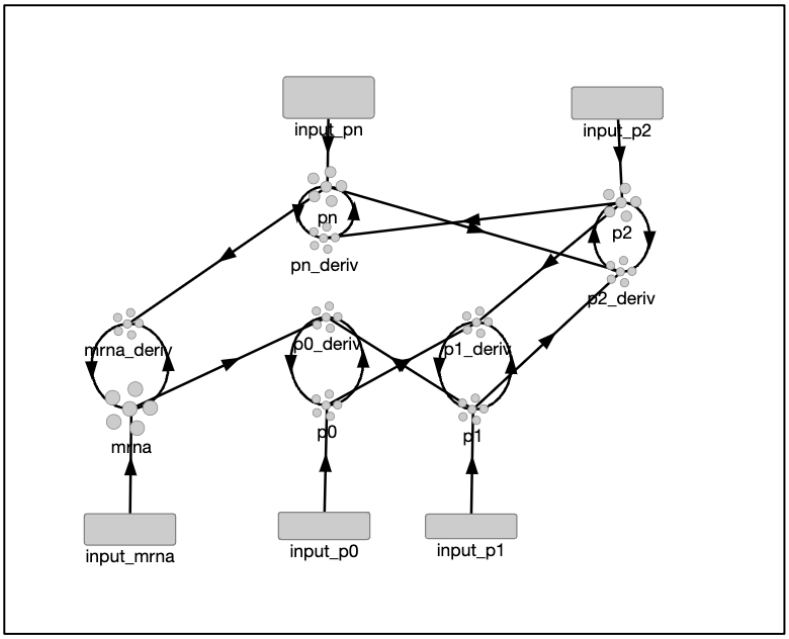

Figure 8: Goldbeter's circadian rhythm model implemented in Nengo

All the constant values for this model matched those listed in Table 1. Goldbeter et al. did not list the initial conditions used for their model, so the values listed in Table 2 are approximations. The initial conditions of the simulation did not seem to change the final oscillatory behavior of the output, although the initial concentrations were always kept within 0nM and 1nM. The full code for this model can be found in Appendix A: The Circadian Model in Nengo.

Table 2: Initial Conditions for the Circadian Rhythm Model

Type of PER Protein	**Concentration (nM)**
PER-0	0.2
PER-1	0.2
PER-2	0.2
nuclear-PER	0.5
PER mRNA	0.5

The results from this model are pictured in Figure 9. A sinusoidal variation in concentration can be seen in nuclear-PER concentration (*right)* but cannot be seen in the PER-mRNA

concentration (*left)*. Further, the PER-mRNA concentration is consistently negative, which is biologically infeasible. A 24-hour oscillation/period cannot be seen in either plot. In order to obtain better results, several model parameters were manipulated without any positive results. This includes the number of neurons in each nucleus, the refractory period of each synapse, the initial conditions, and the refractory period of the derivative nuclei. However, the combined evidence indicates that the model is *unable* to model circadian rhythms in *drosophila* using the current Nengo architecture. Section 4.2: Recommendations elaborates on this point.

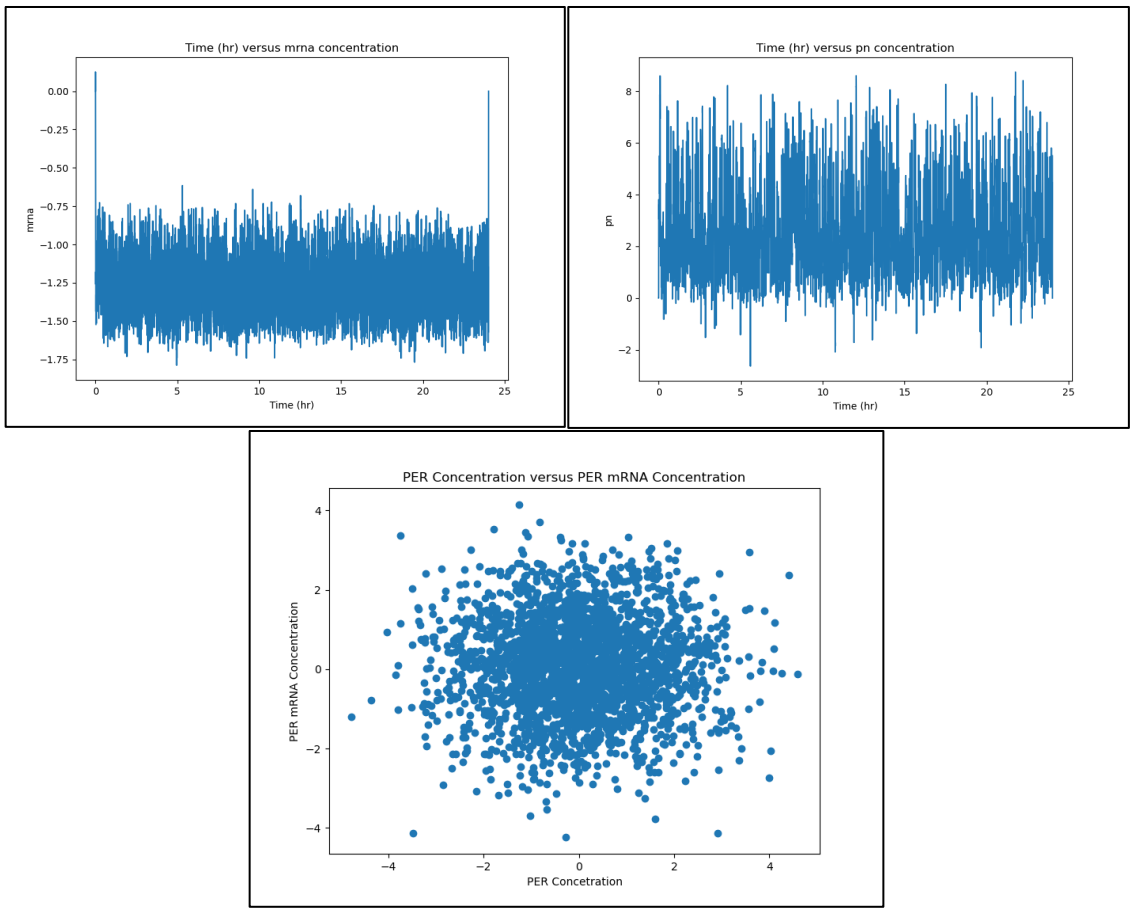

Figure 9: *(left)* mRNA concentration versus time with a 7-point rolling average. *(right)* nuclear-PER concentration with a 7-point rolling average. *(bottom)* mRNA concentration versus PER concentration.

4.0 Project Summary

4.1 Conclusions

Nengo is a software that can build large-scale models using the NEF foundation. Both a simplified and a complex model of circadian rhythms in *drosophila* were re-created using the Nengo software. The simplified model displayed sinusoidal variations in PER-mRNA concentrations over time, although a 24-hour period was not observed. These results are encouraging because they display that the main equation of Goldbeter's circadian rhythm model is correct. The complex model was unable to model the PER-mRNA oscillations over time. Moreover, the concentration remained negative throughout the 24-hour period. To try to fix the model, several parameters were manipulated without any positive results such as the number of neurons in each nucleus, the refractory period of each synapse, the initial conditions, and the refractory period of the derivative nuclei. Therefore, Nengo is unable to model circadian rhythms in *drosophila* using the current architecture and parameters.

4.2 Recommendations

It is unclear why the Nengo model is unable to replicate the analytical results from Goldbeter's model. Due to its vast abilities, the problem is not likely in the capability of the Nengo software to model circadian rhythms, but rather in how the model is implemented. Since the simplified model shows promising results, we recommend that the simplified model be used as a starting foundation, and each neuron ensemble should be added one-by-one. Upon each addition, the model should be run multiple times to ensure that the oscillations do not disappear. Additionally, it is also possible to manipulate the constant parameters listed in Table 1, as Goldbeter et al. chose those values because they allowed the analytical solution to have a 24-hour circadian rhythm cycle. As such, each individual value is arbitrary.

References

Andrew, A. (2016, January 12). *Period*. ADBScience. Retrieved March 31, 2022, from https://adbscience.com/tag/period/

Bechtel, W., & Abrahamsen, A. (2010). Dynamic mechanistic explanation: Computational modeling of circadian rhythms as an exemplar for cognitive science. *Studies in History and Philosophy of Science Part A*, *41*(3), 321–333. https://doi.org/10.1016/j.shpsa.2010.07.003

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., Choo, X., Voelker, A. R., & Eliasmith, C. (2014). Nengo: A python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, *7*. https://doi.org/10.3389/fninf.2013.00048

CNRGLab. (n.d.). *Neural Engineering Framework*. CNRGlab @ UWaterloo | Research. Retrieved April 11, 2022, from http://compneuro.uwaterloo.ca/research/nef.html *Examples*¶. Nengo. (n.d.). Retrieved March 31, 2022, from https://www.nengo.ai/nengo-fpga/examples.html#adaptive-pendulum-control

Foteinou, P. T., Venkataraman, A., Francey, L. J., Anafi, R. C., Hogenesch, J. B., & Doyle, F. J. (2018). Computational and experimental insights into the circadian effects of SIRT1. *Proceedings of the National Academy of Sciences*, *115*(45), 11643–11648. https://doi.org/10.1073/pnas.1803410115

Humphries, M. (2021, February 22). *Why model the brain?* Medium. Retrieved April 11, 2022, from https://medium.com/the-spike/why-model-the-brain-c7a8e160e566#:~:text=Building%20detailed%20models%20of%20a,closely%20your%20model%20imitates%20reality

Kelly, C. (2020, July 6). *What is circadian rhythm? - disorders, symptoms, health effects*. Prescription Hope. Retrieved March 31, 2022, from https://prescriptionhope.com/blog-what-is-circadian-rhythm-disorders-symptoms-health-effects/

Leloup, J.-C., & Goldbeter, A. (1997). Temperature compensation of circadian rhythms: Control of the period in a model for circadian oscillations of the PER protein in drosophila. *Chronobiology International*, *14*(5), 511–520. https://doi.org/10.3109/07420529709001472

Wikimedia Foundation. (2022, March 13). *Inverted pendulum*. Wikipedia. Retrieved March 31, 2022, from https://en.wikipedia.org/wiki/Inverted_pendulum#/media/File:Balancer_with_wine_3.JPG

Appendix A: The Circadian Model in Nengo

This appendix breaks down the model into multiple sections. First, the overall model is generated using a *context manager* provided by Nengo as pictured in Figure A.1.

```
model = nengo.Network()
with model:
   # build the model here.
```

Figure A.1: Nengo Context Manager

Next, the nuclei for each form of the PER protein, as well as the derivative of each protein is generated. For consistency, the same number of neurons are allocated to each nucleus.

```
radius = 10
n_neurons = 500
mrna = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)
mrna_deriv = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)

p0 = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)
p0_deriv = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)

p1 = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)
p1_deriv = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)

p2 = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)
p2_deriv = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)

pn = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)
pn_deriv = nengo.Ensemble(n_neurons=n_neurons, dimensions=1, radius=radius)
```

Figure A.2: Neuron Ensembles / Nuclei for each form of the PER protein

Connections between each ensemble are made using functions to describe how each nucleus affects another. These calculations are based on the equation set listed in Figure 2. Figure A.3 displays the equations as well as the connections.

```
        # Connections to P0_deriv
        def connect_m_to_p0(x):
            return EQUATION_VARS['ks'] * x[0]
        nengo.Connection(mrna, p0_deriv, function=connect_m_to_p0)
        def connect_p0_to_p0_deriv(x):
            return -1 * EQUATION_VARS['V1'] * x[0] / (EQUATION_VARS['K1'] + x[0])
        nengo.Connection(p0, p0_deriv, function=connect_p0_to_p0_deriv)
        def connect_p1_to_p0(x):
            return EQUATION_VARS['V2'] * x[0] / (EQUATION_VARS['K2'] + x[0])
        nengo.Connection(p1, p0_deriv, function=connect_p1_to_p0)


        # Connections to P1_deriv
        def connect_p0_to_p1(x):
            return EQUATION_VARS['V1'] * x[0] / (EQUATION_VARS['K1'] + x[0])
        nengo.Connection(p0, p1_deriv, function=connect_p0_to_p1)
        def connect_p1_to_p1_deriv(x):
            return (
                -1 * EQUATION_VARS['V2'] * x[0] / (EQUATION_VARS['K2'] + x[0])
                -1 * EQUATION_VARS['V3'] * x[0] / (EQUATION_VARS['K3'] + x[0])
            )
        nengo.Connection(p1, p1_deriv, function=connect_p1_to_p1_deriv)
        def connect_p2_to_p1(x):
            return -1 * EQUATION_VARS['V4'] * x[0] / (EQUATION_VARS['K4'] + x[0])
        nengo.Connection(p2, p1_deriv, function=connect_p2_to_p1)


        # Connections to P2_deriv
        def connect_p1_to_p2(x):
            return EQUATION_VARS['V3'] * x[0] / (EQUATION_VARS['K3'] + x[0])
        nengo.Connection(p1, p2_deriv, function=connect_p1_to_p2)
        def connect_p2_to_p2_deriv(x):
            return (
                -1 * EQUATION_VARS['V4'] * x[0] / (EQUATION_VARS['K4'] + x[0])
                -1 * EQUATION_VARS['k1_rate'] * x[0]
                -1 * EQUATION_VARS['vd'] * x[0] / (EQUATION_VARS['KD'] + x[0])
            )
        nengo.Connection(p2, p2_deriv,function=connect_p2_to_p2_deriv)
        def connect_pn_to_p2(x):
            return EQUATION_VARS['k2_rate'] * x[0]
        nengo.Connection(pn, p2_deriv, function=connect_pn_to_p2)

        # Connections to PN_deriv
        def connect_p2_to_pn(x):
            return EQUATION_VARS['k1_rate'] * x[0]
        nengo.Connection(p2, pn_deriv, function=connect_p2_to_pn)
        def connect_pn_to_pn_deriv(x):
            return -1* EQUATION_VARS['k2_rate'] * x[0]
```

Figure A.3 Connections in the Nengo Model

Finally, input nuclei were added with a simple piecewise function to start the oscillations. This replicates the Goldbeter's model. Figure A.4 displays the input nuclei and the connections to the respective parameter.

```
# Make all inputs
        input_p0 = nengo.Node(Piecewise({0: [INITIAL_CONDITIONS['P0']], tao: [0]}))
        nengo.Connection(input_p0, p0, synapse=tao)

        input_p1 = nengo.Node(Piecewise({0: [INITIAL_CONDITIONS['P1']], tao: [0]}))
        nengo.Connection(input_p1, p1, synapse=tao)

        input_p2 = nengo.Node(Piecewise({0: [INITIAL_CONDITIONS['P2']], tao: [0]}))
        nengo.Connection(input_p2, p2, synapse=tao)

        input_pn = nengo.Node(Piecewise({0: [INITIAL_CONDITIONS['Pn']], tao: [0]}))
        nengo.Connection(input_pn, pn, synapse=tao)

        input_mrna = nengo.Node(Piecewise({0: [INITIAL_CONDITIONS['mrna']], tao:
[0]}))
        nengo.Connection(input_mrna, mrna, synapse=tao)
```

Figure A.4: Inputs in the Nengo Model

In order to record the measurements, probes were added to each PER nuclei. Figure A.5 details the generation of the probes. Once the model is created, it is simulated using the native Nengo simulation function, and the results are plotted using the matplotlib library.

```
        # Probes
        probes = {
            "mrna_probe": nengo.Probe(mrna),
            "p0_probe": nengo.Probe(p0),
            "p1_probe": nengo.Probe(p1),
            "p2_probe": nengo.Probe(p2),
            "pn_probe": nengo.Probe(pn)
        }
```

Figure A.5: Probes in the Nengo Model.