Robustness of Biochemical Oscillators

New Mexico
Supercomputing Challenge
Final Report
April 4, 2007

Team 8
Albuquerque Academy

Team Member:
Michael Wang

Teacher:
Jim Mims

Project Mentor:
Yifeng Wang, Ph.D

Table of Contents

Executive Summary

Every biological system has internal biochemical oscillators generated autonomously through a set of coupled chemical reactions. These oscillators are a fundamental part of the functions of an organism, and their robustness to environmental changes is critical to the well being of that organism. These environmental changes may result in irregular oscillations or eventually cause a biological system to collapse. In this project, I chose the circadian rhythms as an example to study the effects of environmental perturbations on the behavior of a biochemical oscillator. Based on Goldbeter's mathematical formulation, I have developed a software tool for studying such effects. With a unique graphic interface, this software allows me to (1) conveniently construct phase diagrams that can be used to delineate the parameter spaces for different behaviors of the circadian oscillatior, (2) determine the recovery time and phase shift of the oscillator in response to a single pulse disturbance, and (3) determine the conditions under which the oscillator becomes chaotic in response to multiple perturbations. From model simulations, I have identified the most sensitive parameters in the circadian model. In the single perturbation cases, I have found that the phase shift generally decreases with increasing disturbance amplitude while the recovery time exhibits a nonlinear manner. This finding is qualitatively in agreement with experimental observations. In the multiple disturbance cases, I have found that the oscillations generally become irregular when the disturbance is approximately greater than 30% of original parameter values, indicating that the circadian oscillator is fairly robust. The software developed here can be applied to many other biochemical oscillators. This project is interdisciplinary across biology, chemistry, and physics.

Introduction

Biochemical oscillators are ubiquitous in all organisms, either unicellular or multicellular, with oscillation periods ranging from less than a second to more than a year (Goldbeter, 1996). These oscillations are produced autonomously through a set of coupled chemical reactions without outside influence. For such oscillations to occur, these chemical reactions must involve either positive or negative feedbacks. A positive feedback means that two factors reinforce each other while a negative feedback means that two factors repress each other. Biochemical oscillations are a fundamental part of the functions of a biological system.

A biological system is constantly subjected to environmental fluctuations. The ability for organisms to withstand or adapt to such changes is crucial for their well being. Such ability depends on the robustness of chemical oscillators involved. I here define the robustness of a chemical oscillator as: (1) the size of the parameter space in phase diagrams where sustainable oscillations persist (the larger the parameter space, the more robust), (2) the ability for the system to recover from a single disturbance, and (3) the ability of the system to retain regular oscillations in the presence of multiple disturbances. Based on a literature search, I have found that the existing research has mainly focused on the first aspect (e.g. Goldbeter and Guilmot, 1996; Tyson et al., 1999; Goldbeter, 1996). Some experimental work has been done on the response of specific oscillators to single pulse disturbance (Hastings and Sweeney, 1958; Taylor et al., 1982). These experiments show that the phase shifts and recovery time both depend on the amplitude of the disturbance. In general, a systematic modeling study of the robustness of a biochemical oscillator with respect to environmental perturbations, especially multiple pulses, is still lacking. One difficulty in studying the dynamics of a biochemical oscillator is the large number of model parameters involved, which makes it very hard to explore model behavior for all possible parameter combinations. This calls for a development of new software tools.

For this project, I chose the circadian rhythms as a representative oscillator. This oscillator is well known for being responsible for our sleep/wake cycles. Based on the mathematical model proposed by Albert Goldbeter (1995), I have developed a code to study the robustness of biochemical oscillators. With a graphic interface, this code allows me to quickly construct phase diagrams that can be used to delineate the parameter spaces for different behaviors of a chemical oscillator, determine the ability for the oscillator to recover from a single pulse disturbance, and determine the conditions for the oscillator to become chaotic in response to multiple perturbations. Based on model simulations, an overall conclusion about the robustness of a specific oscillator can then be drawn.

Mathematical Model

The circadian rhythms, generally described as a biological clock in chronobiology (the study of 24-hour periods), are a ubiquitous oscillator that governs many of our physiological functions (Goldbeter, 1996). It has been studied in many organisms, such as the fruit fly *Drosophila melanogaster* (Goldbeter, 1996). The circadian rhythms have many uses, especially in the field of chronopharmacology. For example, it can be used to determine when a given medication is most effective with minimal unwanted side-effects. The effectiveness of anticancer drug, 5-Fluorouridine (5-FU), which inhibits the synthesis of DNA, follows patterns in the oscillations (e.g. drops and peaks) (Goldbeter, 1996).

The circadian rhythms are characterized by three properties (Goldbeter, 1996). First, the circadian rhythms have a free-running period (spontaneous periods) of about 24 hours. Secondly, they can be easily phase-shifted by pulses of light applied for certain durations of time (e.g. jetlag). Thirdly, temperature compensation distinguishes the circadian rhythms from most other biological rhythms**.** The frequency of other oscillations generally increases as temperature increases. However, in circadian rhythms, the oscillations remain largely unaffected by any temperature deviations. The detailed mechanism for the circadian rhythms still remains unknown. A theoretical model has proved very useful for a better understanding of the underlying mechanism.

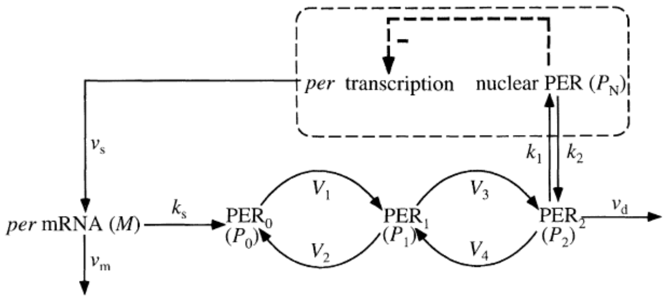


Figure 1. Scheme of the Circadian Oscillations model (Goldbeter, 1995).

Shown in Figure 1 is the model of circadian rhythms proposed by Goldbeter (1995). This model is described by five nonlinear ordinary differential equations (ODEs) (Goldbeter, 1995):

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

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

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

$$\frac{dP_2}{dt} = V_3 \frac{P_1}{K_3 + P_1} - V_4 \frac{P_2}{K_4 + P_4} - k_1 P_2 + k_2 P_N - v_d \frac{P_2}{K_d + P_2} \tag{4}$$

$$\frac{dP_N}{dt} = k_1 P_2 - k_2 P_N \tag{5}$$

where the variables have the following meanings:

M = *per* mRNA (messenger RNA)
P_0 = unphosphorylated PER (period protein)
P_1 = monophosphorylated PER
P_2 = biphosphorylated PER
P_N = fully phosphorylated PER (referred to as simply "PER")

v_s = maximum rate of M accumulation
v_m = maximum rate of M degradation
v_d = maximum rate of P_2 degradation
k_s = rate constant characterizing the synthesis of M
k_1 and k_2 = rate constant characterizing the transportation between P_2 and P_N
V_1 = maximum rate of conversion, $P_0 \rightarrow P_1$
V_2 = maximum rate of reverse conversion, $P_1 \rightarrow P_0$
V_3 = maximum rate of conversion, $P_1 \rightarrow P_2$
V_4 = maximum rate of reverse conversion, $P_2 \rightarrow P_1$
K_1 = Michaelis constant for describing V_1
K_2 = Michaelis constant for describing V_2
K_3 = Michaelis constant for describing V_3
K_4 = Michaelis constant for describing V_4
K_I = Threshold constant
n = degree of cooperativity

This model includes a negative feedback between PER and *per* mRNA, as shown in Figure 1. As PER increases, the production of *per* mRNA decreases, causing unphosphorylated PER, monophosphorylated PER, biphosphorylated PER, and PER also to decrease, but as PER decreases, the production of *per* mRNA increases, causing PER to increase again, and so on, resulting in oscillations.

Numerical Solution

Equations (1-5) are nonlinear ODEs that are impossible to solve analytically. They must be solved numerically, either implicitly or explicitly. For convenience, I chose the explicit finite difference method. The explicit method requires a small time step to ensure the stability and accuracy of the numerical solutions (*see* Appendix A: Stability Analysis). The code developed here allows the user to set any time step. For all the simulations here, the default time step is set to 12 min, which is accurate enough for my model simulations as shown in Appendix A.

As an example, the finite difference of Equation (5) can be written as follows:

$$\frac{dP_N}{dt} = k_1 P_2 - k_2 P_N$$

$$\frac{P_{N,i} - P_{N,i-1}}{\Delta t} = k_1 P_{2,i-1} - k_2 P_{N,i-1}$$

$$P_{N,i} = (k_1 P_{2,i-1} - k_2 P_{N,i-1})\Delta t + P_{N,i-1} \qquad (5a)$$

In the above equations, the derivative $\frac{dP_N}{dt}$ is replaced by the difference quotient, $\frac{P_{N,i} - P_{N,i-1}}{\Delta t}$, and $k_1P_2 – k_1P_N$ by $k_1P_{2,i–1} – k_1P_{N,i–1}$, such that $P_{N,i}$, the concentration of P_N at the current time step *i*, can be easily calculated from the previous time step. Equations 1, 2, 3, and 4 are solved in a similar manner:

$$M_i = (v_s \frac{K_I^n}{K_I^n + P_{N,i-1}^n} - v_m \frac{M_{i-1}}{K_m + M_{i-1}})\Delta t + M_{i-1} \qquad (1a)$$

$$P_{0,i} = (k_s M_{i-1} - V_1 \frac{P_{0,i-1}}{K_1 + P_{0,i-1}} + V_2 \frac{P_{1,i-1}}{K_2 + P_{1,i-1}})\Delta t + P_{0,i-1} \qquad (2a)$$

$$P_{1,i} = (V_1 \frac{P_{0,i-1}}{K_1 + P_{0,i-1}} - V_2 \frac{P_{1,i-1}}{K_2 + P_{1,i-1}} - V_3 \frac{P_{1,i-1}}{K_3 + P_{1,i-1}} + V_4 \frac{P_{2,i-1}}{K_4 + P_{2,i-1}})\Delta t + P_{1,i-1} \qquad (3a)$$

$$P_{2,i} = (V_3 \frac{P_{1,i-1}}{K_3 + P_{1,i-1}} - V_4 \frac{P_{2,i-1}}{K_4 + P_{2,i-1}} - k_1 P_{2,i-1} + k_2 P_{N,i-1} - v_d \frac{P_{2,i-1}}{K_d + P_{2,i-1}})\Delta t + P_{2,i-1} \quad (4a)$$

$$P_{N,i} = (k_1 P_{2,i-1} - k_2 P_{N,i-1})\Delta t + P_{N,i-1} \quad (5a)$$

With the resultant finite difference equations, the concentration variables M, P_0, P_1, P_2, and P_N can now be calculated using iterations from one time step to the next.

Figure 2 depicts sustainable oscillations simulated using the following parameter values: $v_s = 0.76$ μM h^{-1}, $v_m = 0.65$ μM h^{-1}, $v_d = 0.95$ μM h^{-1}, $k_s = 0.38$ h^{-1}, $k_1 = 1.9$ h^{-1}, $k_2 = 1.3$ h^{-1}, $V_1 = 3.2$ μM h^{-1}, $V_2 = 1.58$ μM h^{-1}, $V_3 = 5$ μM h^{-1}, $V_4 = 2.5$ μM h^{-1}, $K_1 = K_2 = K_3 = K_4 = 2$ μM, $K_m = 0.5$ μM, $K_d = 0.2$ μM, $K_I = 1$ μM, and $n = 4$ with initial M, P_0, P_1, P_2, and P_N at 0.6 μM, 0.5 μM, 0.5 μM, 0.6 μM, and 1.1 μM, respectively (Fall et al., 2002). This simulation constitutes a base case for this project. The same result is presented by Fall et al. (2002, p.252). This demonstrates that I solved Equations (1-5) correctly.

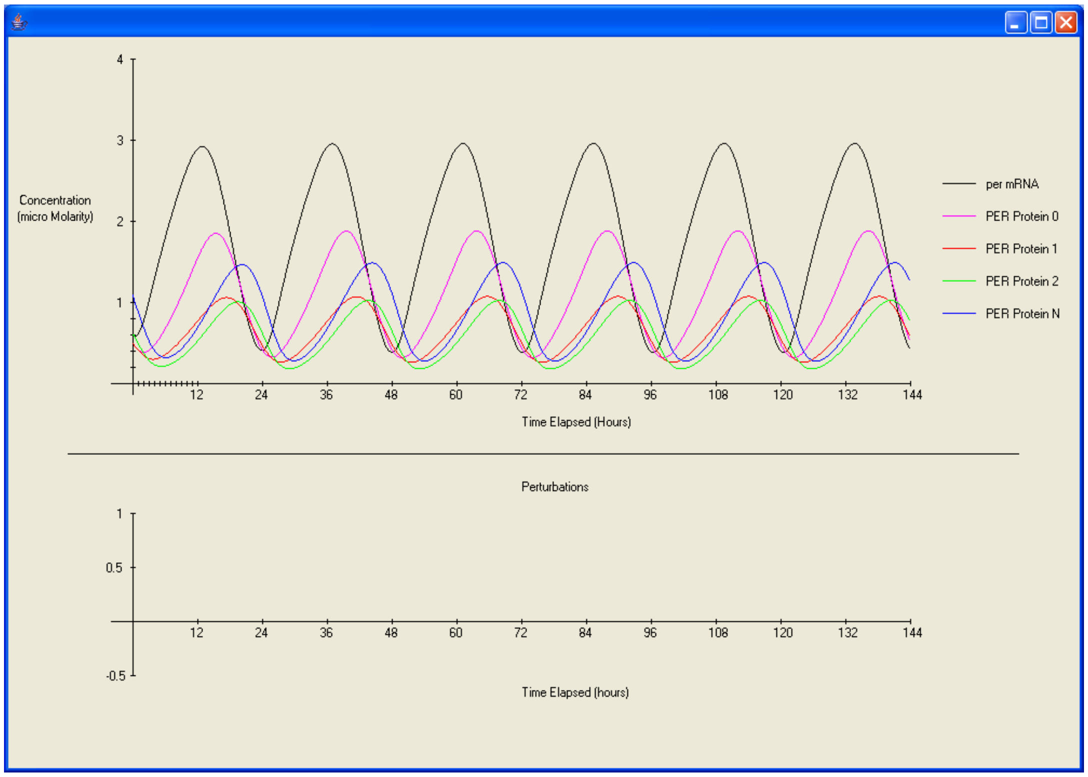


Figure 2. Base case simulation. No perturbations were added (*see* bottom graph).

To study the effects of environmental disturbances, a "noise" is added to a selected model parameter, for example, for Equation (5a),

$$P_{N,i} = [(k_1 + amp_{k_1}\varepsilon_t)P_{2,i-1} - (k_2 + amp_{k_2}\varepsilon_t)P_{N,i-1}]\Delta t + P_{N,i-1}$$

where amp_{k_1} and amp_{k_2} are the disturbance amplitudes of the selected parameters k_1 and k_2, respectively; and ε_t is the random number from 0 to 1 (a concept used in Monte-Carlo simulations). Similar "noises" are added to all the parameters but M, P_0, P_1, P_2, P_N, and Δt in Equations (1a-5a) (e.g. for the parameter V_1, the amplitude is amp_{V_1}).

Architecture of the Software

The main interface Main (referred to as "Welcome" in Appendix B, Figure 1) gives the user a broad overview of the program and allows them to choose whether to start or quit. When start is chosen, the class Input (Figure B-2) is called. There, new values can be assigned to the model parameters. After the changes are made, the user may choose to create a simulation or input environmental disturbances. If the user chooses to create a simulation, he or she may select either Non_Dynamic (Figure B-4) (a static display) or Dynamic (a dynamic display). The dynamic display turns out to be very useful for observing long term oscillations that cannot be seen in a static display. If environmental disturbances are needed, RandomPerturbation (Figure B-3) is called. In this class, disturbance amplitudes for selected parameters are entered. These values are saved for either a multiple pulse or a single pulse simulation. These values are stored in the class Parameters and then accessed by Input to create a simulation with disturbances.

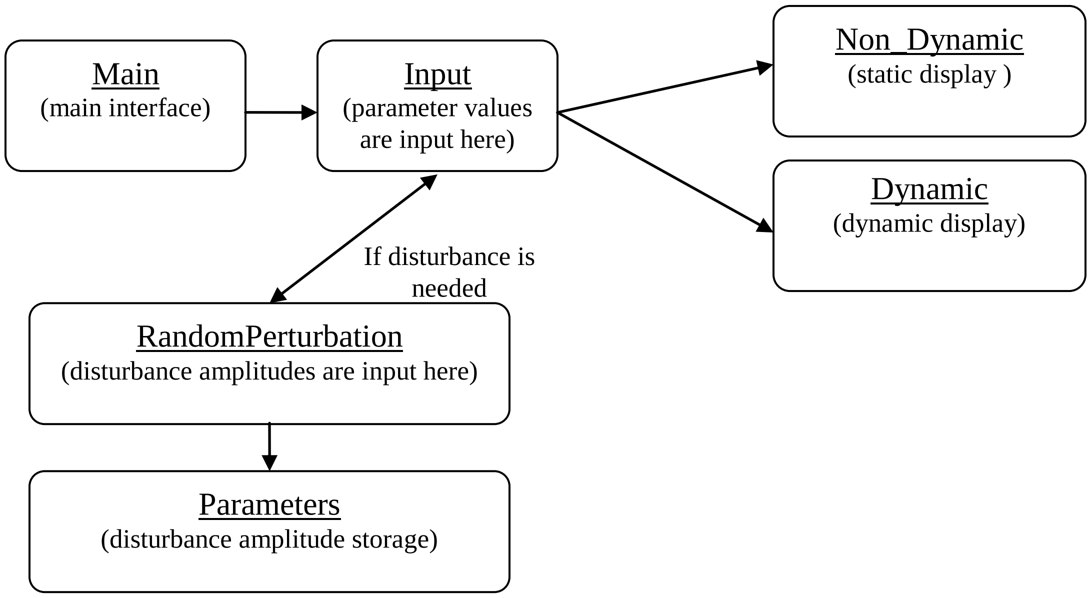


Figure 3. The architecture of my program.

This software is written in Java using the JBuilder integrated development environment. The graphic interface is designed with a Java applet. One advantage of using Java is that the code developed is portable to other platforms

Results and Discussion

Phase Diagrams

The software developed here is able to quickly simulate and display circadian oscillations for a given set of model parameters, which allows me to explore a much broader parameter space than the scientists could possibly do before. I simulated circadian oscillations for a number of combinations of parameter values. Based on the simulations, I have identified the most sensitive parameters and constructed phase diagrams to delineate the parameter spaces for sustainable oscillations. Among the parameters investigated, v_s (the maximum rate of *per* mRNA accumulations), v_m (the maximum rate of *per* mRNA degradation), and K_I (the threshold constant) are identified to be most sensitive. Some of these phase diagram I constructed are shown in Figures 4 and 5.

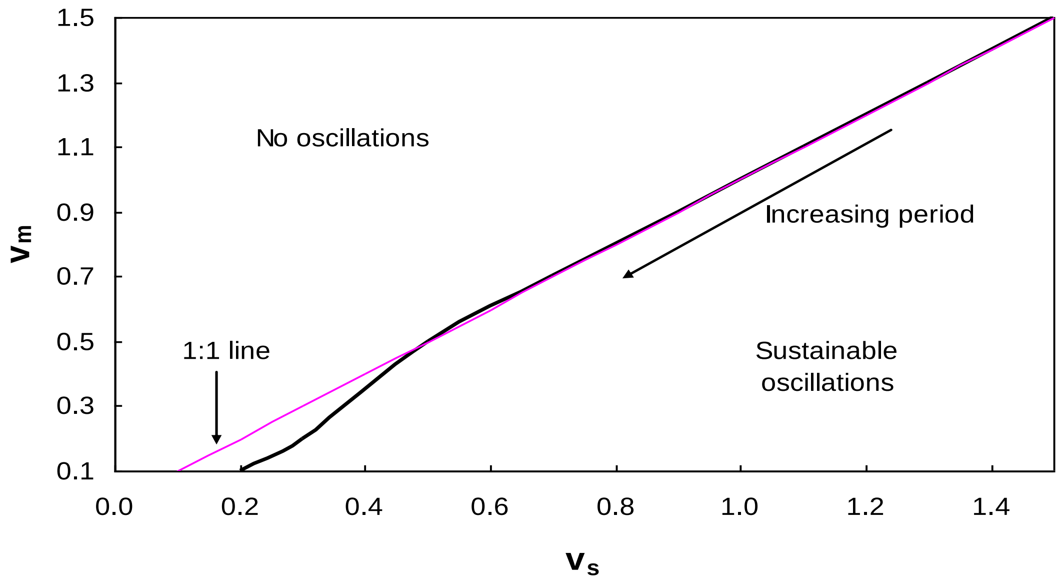


Figure 4. Phase diagram of v_m and v_s. Sustainable oscillations exist only when v_s is greater than v_m. The domain for such oscillations is relatively large.

As shown in Figure 4, the system with parameters v_m and v_s below the line produces sustainable oscillations. Roughly speaking, a sustainable oscillation requires the rate of accumulation v_s to be greater than the rate of degradation v_m; otherwise, the *per* mRNA would degrade faster than it accumulates, resulting in the depletion of proteins and therefore a collapse of the system. From this assumption, a 1:1 line can be drawn, which roughly coincides with the line obtained from simulations. In addition, the period of the oscillations increase with decreasing v_m and v_s.

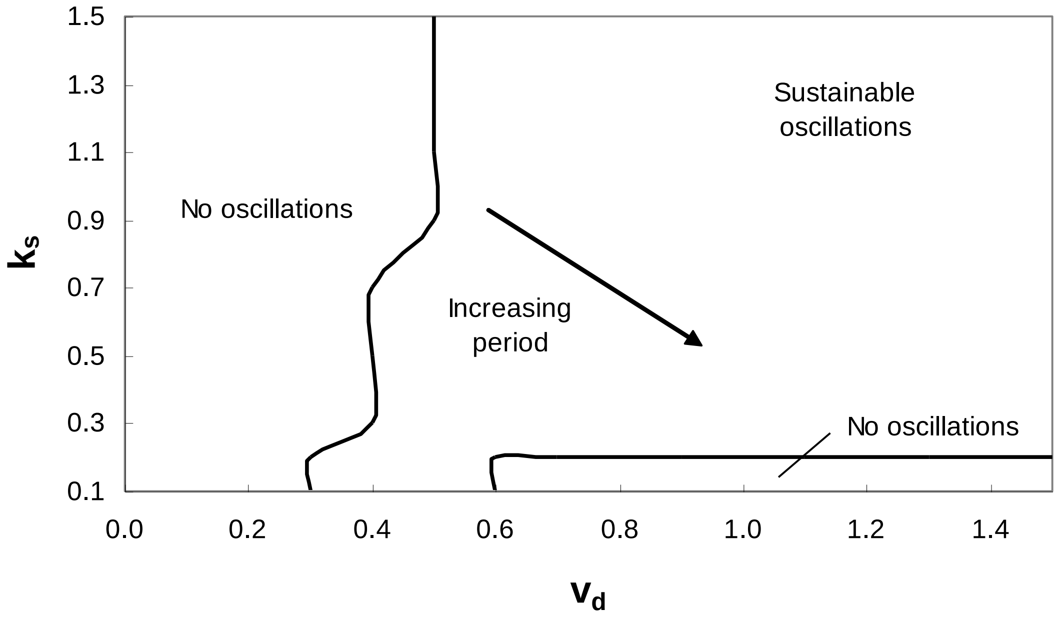


Figure 5. Phase diagram of k_s and v_d. Sustainable oscillations tend to occur for large k_s and v_d.

For some combinations of parameters, the phase diagrams are more complicated. Figure 5 is one of these. In the parameter space of k_s and v_d, there are no oscillations in the two domains (left and bottom). This diagram is much harder to understand intuitively. The figure also shows that as k_s decreases and v_d increases, the period of the oscillations increases.

Single Pulse Disturbance

My program can simulate two kinds of disturbances: single pulse and multiple pulses. Single pulse disturbances have been studied experimentally on organisms such as the algae *Gonyaulax polyedra* (Hastings and Sweeney, 1958). The main issue related to the single pulse disturbance is how long it takes for a system to recover from a disturbance. One example of single pulse disturbance is jet lag, which is closely mimicked in Figure 6.

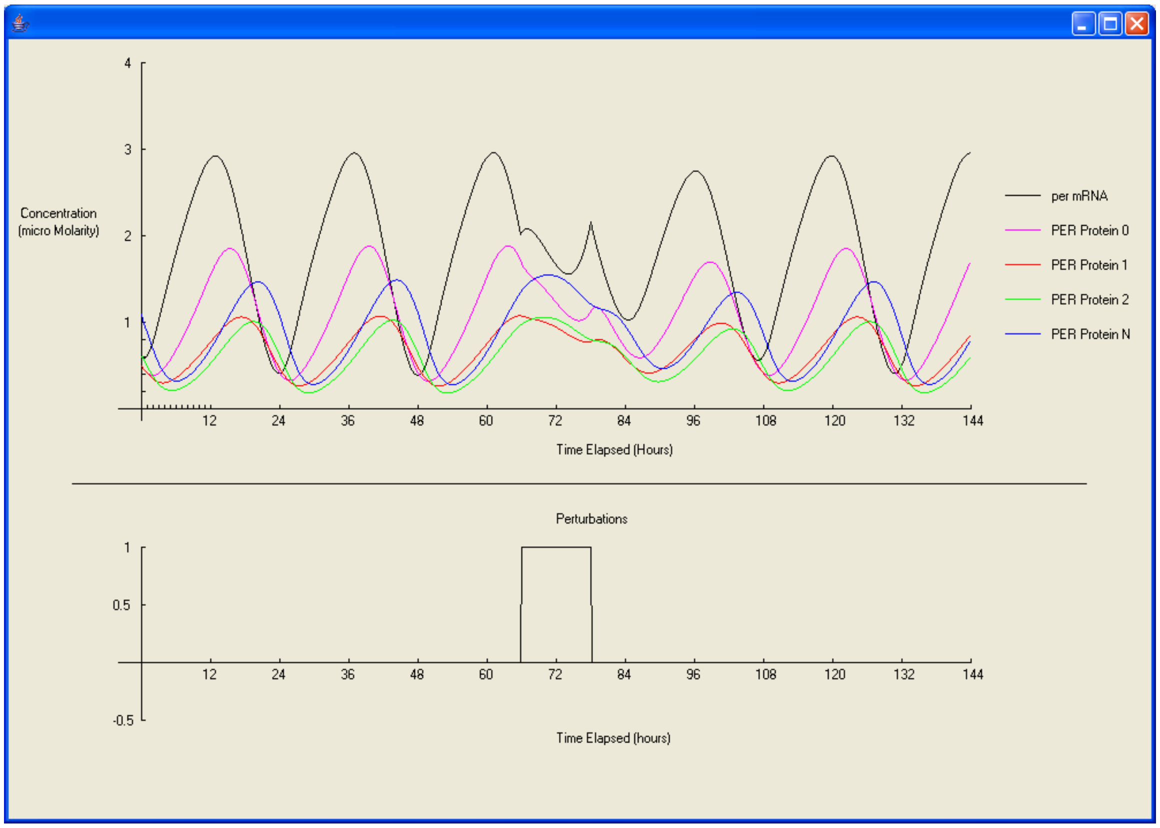


Figure 6: Example of single pulse perturbation in v_s. This closely mimics a half day jet lag because of its complete flip near the end.

Studying the sensitivity of a system to a single disturbance includes the estimation of recovery time and phase shift (amount of hours shift forward or backward). The estimation of recovery time here is only approximate because it is difficult to find the exact point at which the oscillations return back to normal.

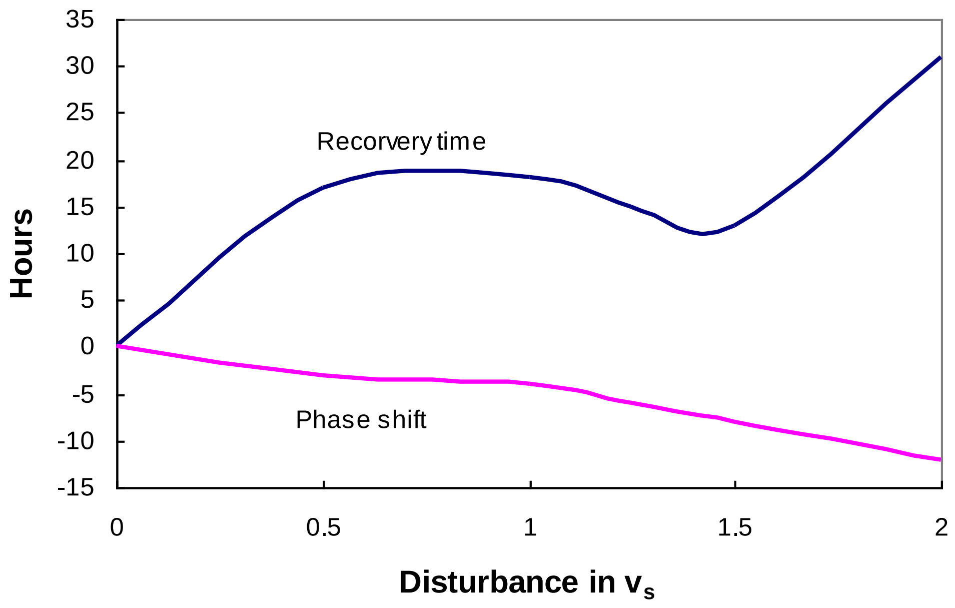


Figure 7. Recovery time and phase shift as a function of the amplitude of disturbance in v_s. The duration of disturbance is 12 hours. As the disturbance amplitude increases, the recovery time behaves nonlinearly while the phase shift is negative (oscillations shift backwards).

In Figure 7, the duration of the disturbance is kept at 12 hours while the amplitude of disturbance is increased. In the interval from 0 to 0.5, the recovery time rises significantly and then levels off over the interval from 0.5 to 1. Surprisingly enough, it actually decreases in the interval of 1 to 1.5. The reason for this nonlinear behavior is unknown. Also shown in Figure 7, the oscillations shift backward and the phase shift increases with the disturbance amplitude. This prediction qualitatively agrees with experimental observations (Hastings and Sweeney, 1958; Taylor et al., 1982).

In most of the single pulse simulations completed, the oscillations shift backwards. However, there is one in particular for v_m where the oscillations shift forward as much as 6 hours over the interval from 0 to 1.5, as shown in Figure 8. At the amplitude 1.5, the phase shift begins to decrease, but from simulations done for amplitudes greater than 2, the phase shift actually levels off and remains about the same.

This is easy to explain because when the single pulse is applied, the value of v_m (disturbance > 2) is greater than v_s, meaning that during this pulse interval the oscillations collapses, but once the disturbance disappears, v_m returns to its original value and the oscillations recover with a phase shift of about the length of the duration.

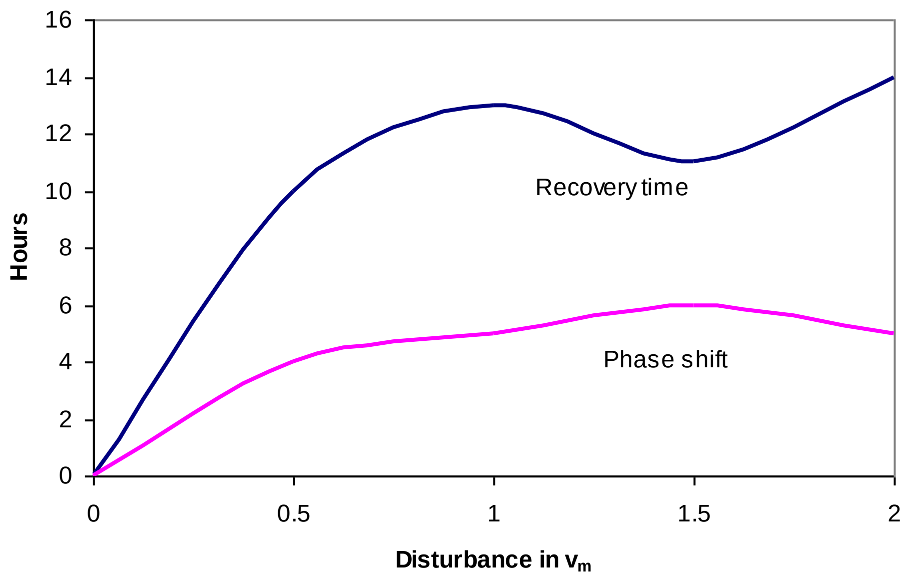


Figure 8. Recovery time and phase shift as a function of the amplitude of disturbance in v_m. The duration of disturbance is 12 hours. Unlike Figure 8, this phase shift is positive (forward shift).

Multiple pulse disturbances

Based on a literature search, I found that little work had been done on multiple pulse disturbances. Simulating multiple pulse perturbations is one of the unique aspects of my project. When multiple disturbances are randomly added, oscillations generally do not collapse because the constant changes in disturbance always spur an increase or decrease in the oscillations. Instead, these oscillations can either remain regular or become irregular. In general, irregular oscillations are harmful to a biological system. Examples of irregular and regular oscillations are shown in Figures 9 and 10.

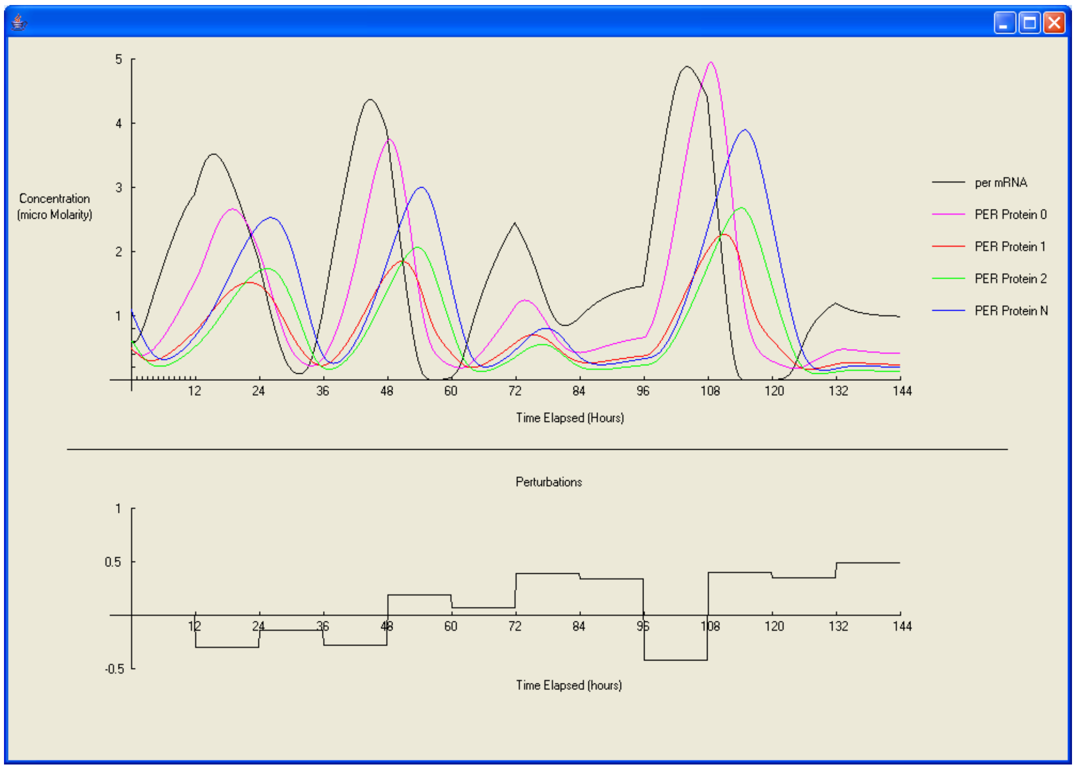


Figure 9: Example of irregular oscillations. The bottom graph displays the random numbers generated to represent random perturbations in v_s.

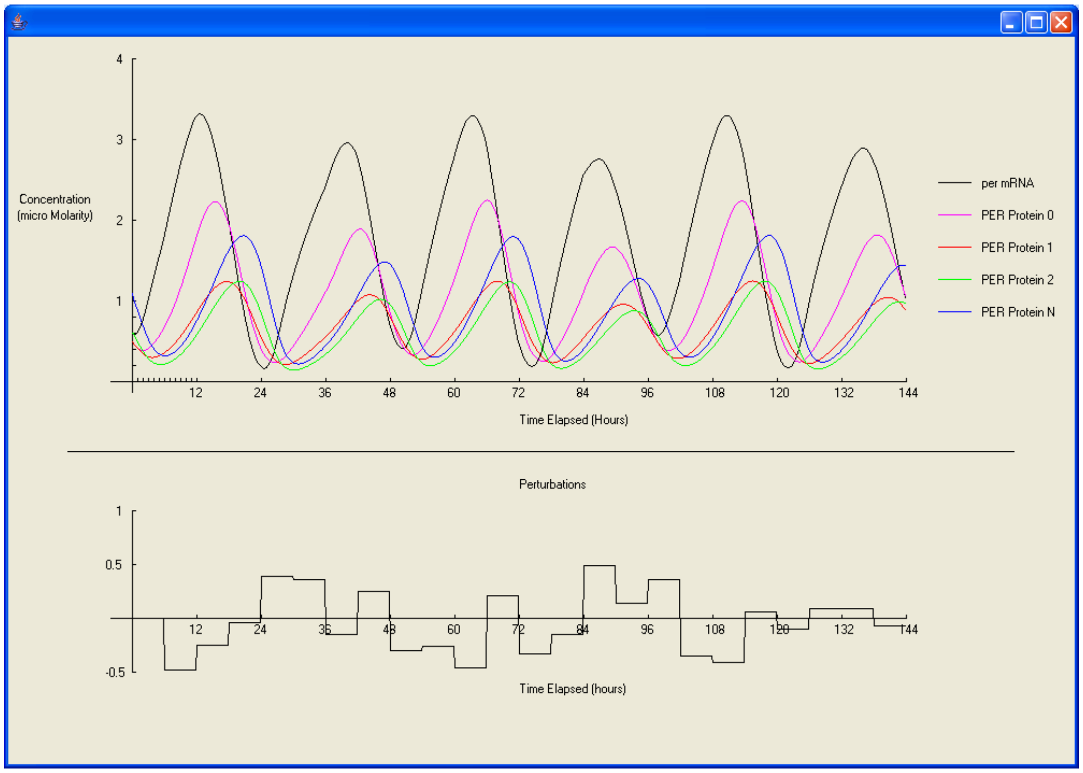


Figure 10: Example of regular oscillations. The bottom graph displays the random numbers generated to represent random perturbations in v_s. For normal oscillations, see Appendix B.

It can be seen in Figure 11 that when the disturbance amplitude and duration of v_s are relatively large, the oscillations are irregular. It seems that a large disturbance and short duration would also produce irregular oscillations; however, this is not the case. As the duration decreases, the system becomes less sensitive to the disturbance amplitude, as shown in Figure 11. In the parameter space I simulated, the oscillations generally become irregular when the disturbance amplitude for v_s exceeds about 30% of the original value of v_s, which is 0.76. Since v_s is one of the most sensitive parameters, this indicates that the circadian oscillator is fairly robust. I also noticed that when there are three or more consecutive perturbations with the same signs (+/-) and large disturbance amplitudes, the oscillations tend to become extremely irregular.

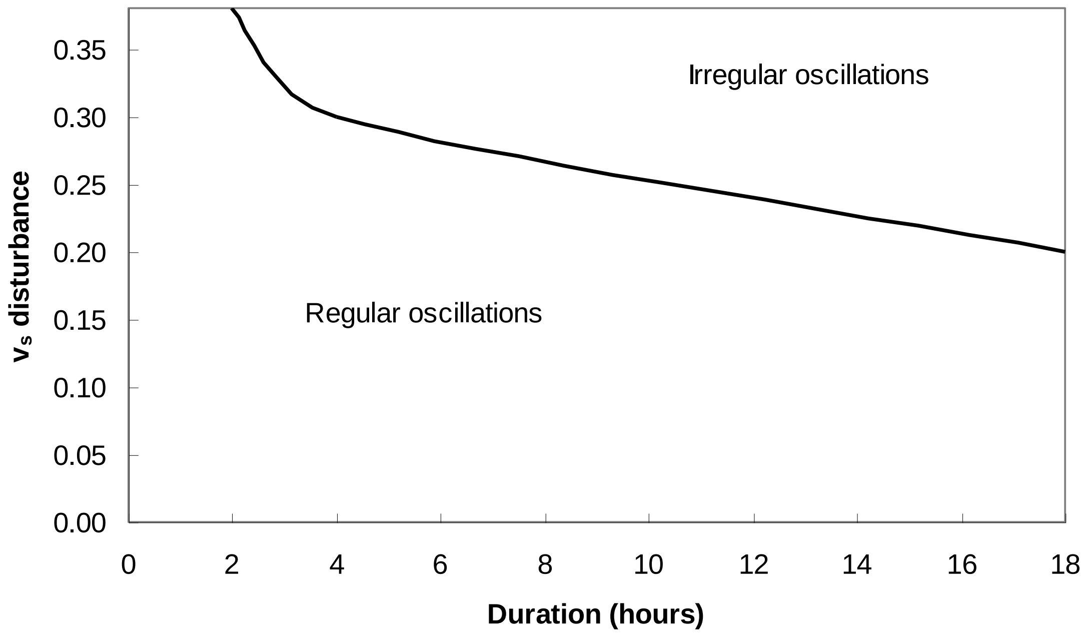


Figure 11: Multiple pulse disturbance phase diagram for v_s. The oscillations become irregular when the disturbance amplitude of v_s exceeds about 30% of its original value. Since v_s is one of the most sensitive parameters, it can be seen that the circadian oscillator is fairly robust for the parameter space simulated.

Conclusions

All living organisms have internal biochemical oscillators. Under ideal conditions, these oscillators maintain regular oscillations. However, environmental disturbances are always present. Because of these disturbances, the oscillations may become irregular or eventually collapse. Therefore, the robustness of an oscillator is crucial to the survival of an organism. For this reason, the study of the effects of environmental disturbance on biochemical oscillators is important. Given the time constraint, I only chose the circadian rhythms for my project. The conclusions I made from my simulations a listed below.

I have successfully developed a program that is able to simulate the oscillations in the presence of environmental disturbances. The simulation results obtained so far match experimental observations reported in the literatures, thus giving validity to my program. This program contains an input interface where the user can change the default values of model parameters and add disturbances to these parameters. The simulation results are displayed graphically, either statically or dynamically. The dynamic display is useful for observing long term oscillations that cannot be displayed statically.

Based on a large number of simulations, I identified the most sensitive parameters in the circadian model. In the single perturbation cases, I found that the phase shift generally decreases with increasing disturbance amplitude while the recovery time exhibits a nonlinear behavior. In the multiple disturbance cases, I found that the oscillations generally become irregular when the disturbance is approximately greater than 30% of original parameter values. Based on all of these observations, I conclude that the circadian oscillator is fairly robust and it can adapt to an environmental disturbance by shifting its oscillation phase. The software developed here can be extended to other biochemical oscillators.

One of my major achievements on this project was the coding part because I had to start from scratch (no preexisting code was used). The dynamic simulations were the hardest part of the coding since I had failed many times before trying to create a simple animation. My second major achievement was running many simulations and creating all of the diagrams that helped me to understand the dynamics of the circadian oscillations. In version 2.0 of my program, I hope to solve the equations implicitly rather than explicitly to improve the numerical stability. In addition, I hope to include other oscillators.

Acknowledgements

First of all, I would like to thank Mr. Mims for his encouragement and help. I took his Java class last summer from which I learned a lot. I would also like to thank my mentor, also my dad, for reviewing my project and for providing me resources.

References

Fall C.P., Marland E.S., Wagner J.M., & Tyson J.J. (eds.). 2002. Computational Cell Biology. Springer-Verlag New York. **20**: 250-54.

Goldbeter, A. 1996. Biochemical Oscillators and Cellular Rhythms: The molecular bases of periodic and chaotic behavior. Cambridge University Press. 459-90.

Goldbeter, A. 1995. A Model for Circadian Oscillations in the Drosophila Period Protein (PER). *Proceedings: Biological Sciences*, **261**: 319-24.

Goldbeter A. & Gonze D. 2000. Entrainment vs. Chaos in a Model for a Circadian Oscillator Driven by Light-Dark Cycles. *Journal of Statistical Physics*, **101**: 649-63.

Goldbeter, A. & Guilmot, J-M. 1996. Thresholds and Oscillations in Enzymatic Cascades. *J. Phys. Chem.*, **100**: 19174-19181.

Hastings, J.W. & B.M. Sweeney. 1958. A Persistent Diurnal Rhythm of Luminescence in *Gonyaulax polyedra*. *Biological Bulletin*, **115**: 440-58.

Taylor, W., Krasnow, R., Dunlap, J.C., Broda, H., & Hastings J.W. 1982. Critical pulses of anisomycin drive the circadian oscillator in *Gonyaulax* towards its singularity. *J. Comp. Physiol.* **148:** 11-25.

Tyson J.J., Hong C.I., Thron C.D., & Novak B. 1999. A Simple Model of Circadian Rhythms Based on Dimerization and Proteolysis of PER and TIM. *Biophysical Journal*, **77**: 2411-2417.

Appendix A: Numerical Stability Analysis

The use of the explicit difference method can potentially produce inaccurate results for large time steps. As shown in Figures A-1 through A-4, these inaccurate results generally appear when the time step is greater than 25 minutes. The numerically stable simulations with 6 and 12 minute time steps (Figures A-1 and A-2) are both comparable with the oscillations shown in Fall et al. (2002, p.252). Cutting the time step from 12 to 6 minutes does not show any significant improvement to the results. Therefore, I decided that my default time step, 12 minutes, is stable enough.

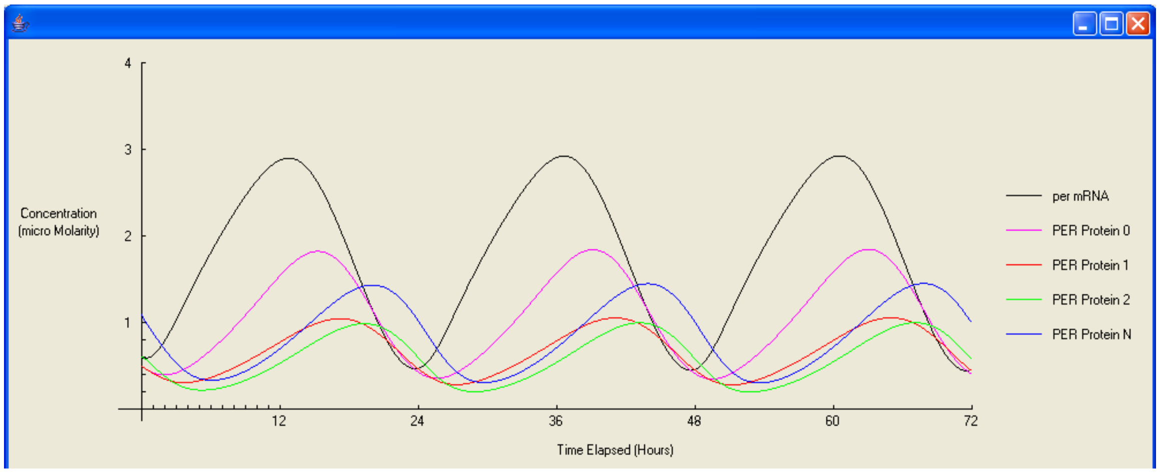


Figure A-1: This six minute time step simulation is extremely stable.

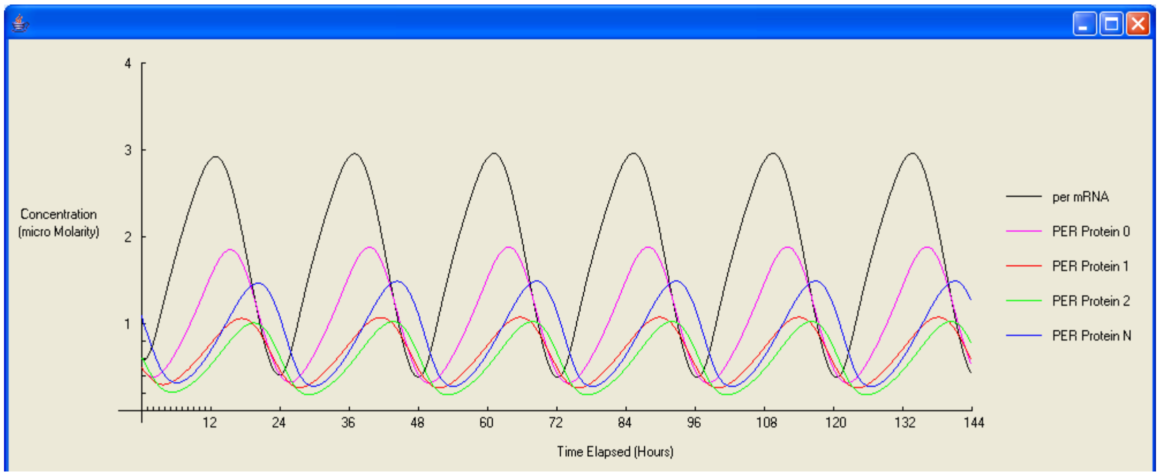


Figure A-2: This 12 minute simulation has about the same accuracy as a six minute simulation.

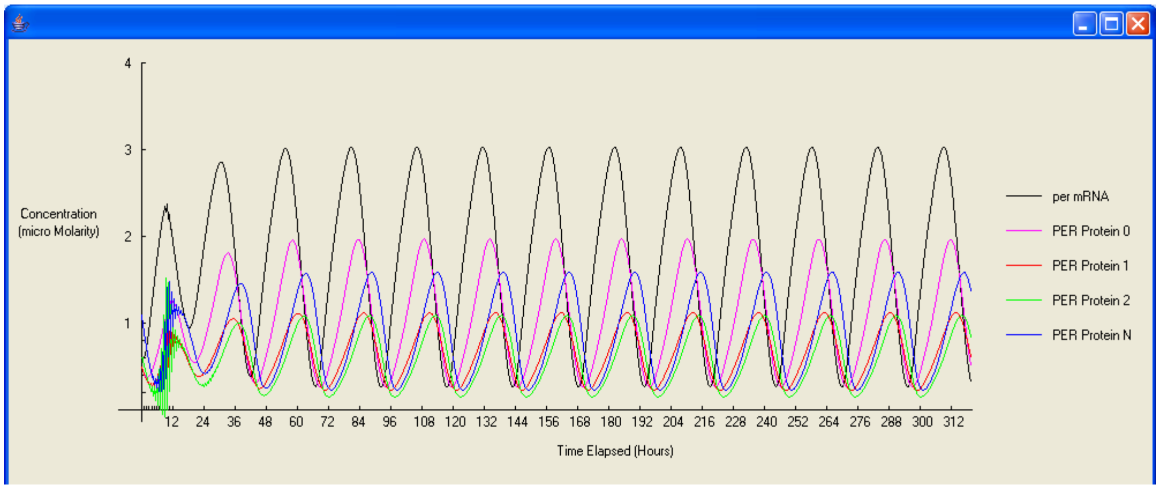


Figure A-3: This 26 minute time-step simulation is fairly unstable because of its sudden divergence in the first 12 hours.

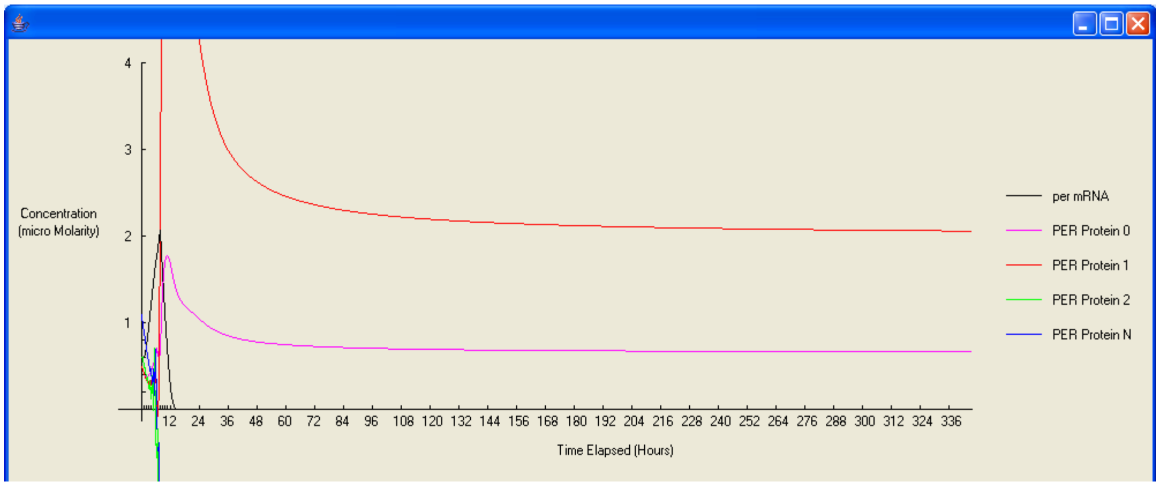
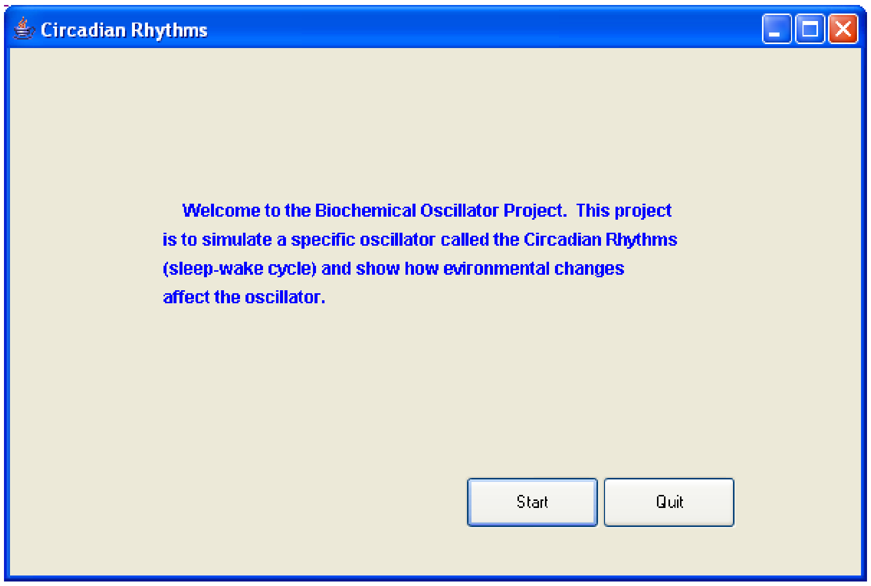


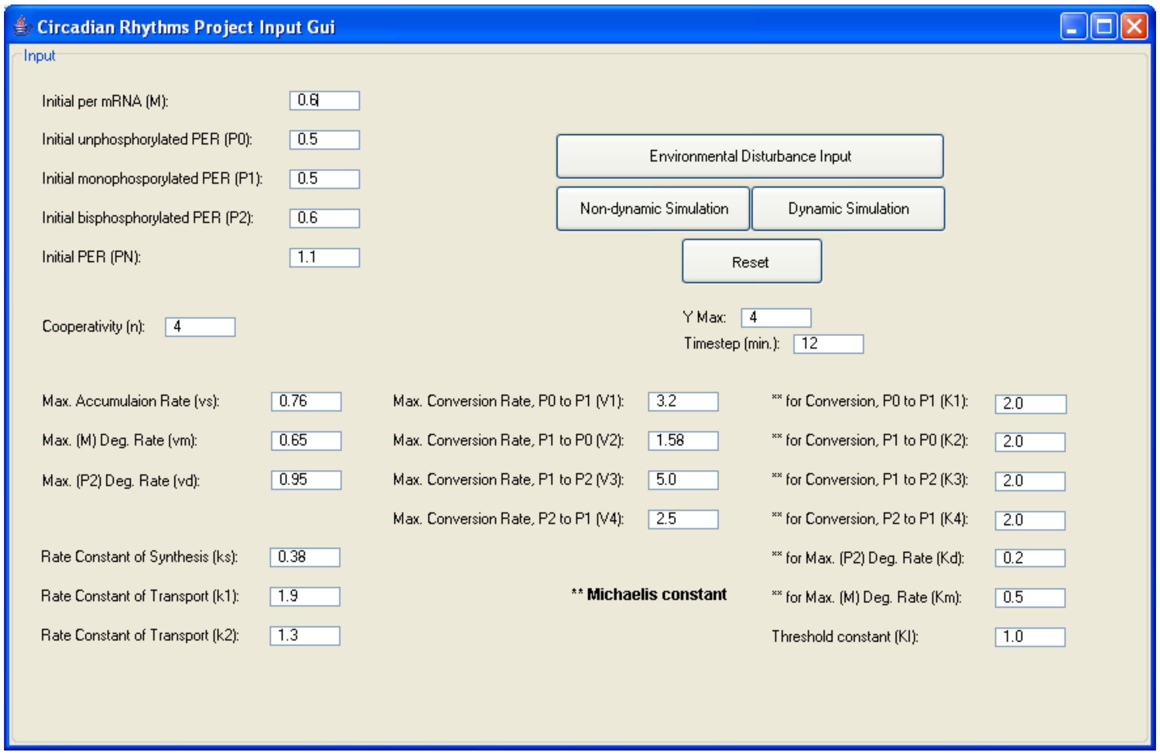
Figure A-4: This 28 minute time-step simulation is extremely unstable and unusable.

Appendix B: Screenshots



Welcome to the Biochemical Oscillator Project. This project is to simulate a specific oscillator called the Circadian Rhythms (sleep-wake cycle) and show how evironmental changes affect the oscillator.

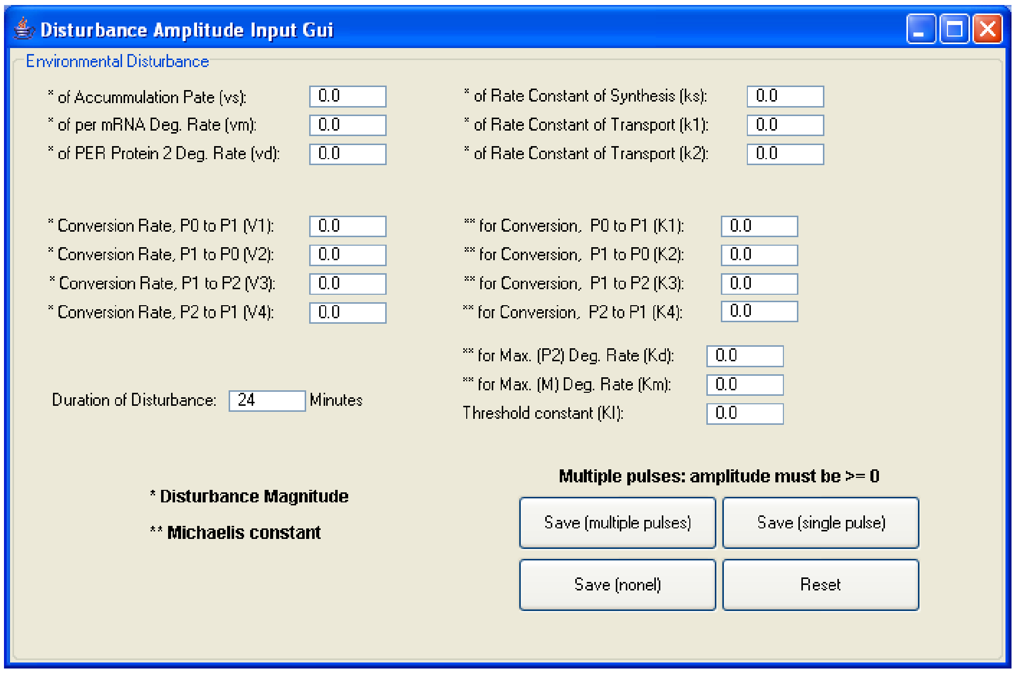
Figure B-1: Welcome screen. The user may choose to start or quit the program.


Circadian Rhythms Project Input Gui
Input
Environmental Disturbance Input
Non-dynamic Simulation
Dynamic Simulation
Reset

Initial per mRNA (M):	0.6
Initial unphosphorylated PER (P0):	0.5
Initial monophosporylated PER (P1):	0.5
Initial bisphosphorylated PER (P2):	0.6
Initial PER (PN):	1.1
Cooperativity (n):	4
Y Max:	4
Timestep (min.):	12
Max. Accumulaion Rate (vs):	0.76
Max. (M) Deg. Rate (vm):	0.65
Max. (P2) Deg. Rate (vd):	0.95
Rate Constant of Synthesis (ks):	0.38
Rate Constant of Transport (k1):	1.9
Rate Constant of Transport (k2):	1.3
Max. Conversion Rate, P0 to P1 (V1):	3.2
Max. Conversion Rate, P1 to P0 (V2):	1.58
Max. Conversion Rate, P1 to P2 (V3):	5.0
Max. Conversion Rate, P2 to P1 (V4):	2.5
** for Conversion, P0 to P1 (K1):	2.0
** for Conversion, P1 to P0 (K2):	2.0
** for Conversion, P1 to P2 (K3):	2.0
** for Conversion, P2 to P1 (K4):	2.0
** for Max. (P2) Deg. Rate (Kd):	0.2
** for Max. (M) Deg. Rate (Km):	0.5
Threshold constant (KI):	1.0

** Michaelis constant

Figure B-2: Input Gui. The values for each model parameter shown above are defaults and can be changed by the user.



Disturbance Amplitude Input Gui

Environmental Disturbance

* of Accummulation Pate (vs):	0.0	* of Rate Constant of Synthesis (ks):	0.0
* of per mRNA Deg. Rate (vm):	0.0	* of Rate Constant of Transport (k1):	0.0
* of PER Protein 2 Deg. Rate (vd):	0.0	* of Rate Constant of Transport (k2):	0.0
* Conversion Rate, P0 to P1 (V1):	0.0	** for Conversion, P0 to P1 (K1):	0.0
* Conversion Rate, P1 to P0 (V2):	0.0	** for Conversion, P1 to P0 (K2):	0.0
* Conversion Rate, P1 to P2 (V3):	0.0	** for Conversion, P1 to P2 (K3):	0.0
* Conversion Rate, P2 to P1 (V4):	0.0	** for Conversion, P2 to P1 (K4):	0.0
		** for Max. (P2) Deg. Rate (Kd):	0.0
		** for Max. (M) Deg. Rate (Km):	0.0
Duration of Disturbance:	24 Minutes	Threshold constant (KI):	0.0

* Disturbance Magnitude

** Michaelis constant

Multiple pulses: amplitude must be >= 0

Figure B-3: Environmental Disturbance Input Gui. Disturbance amplitudes are input here.

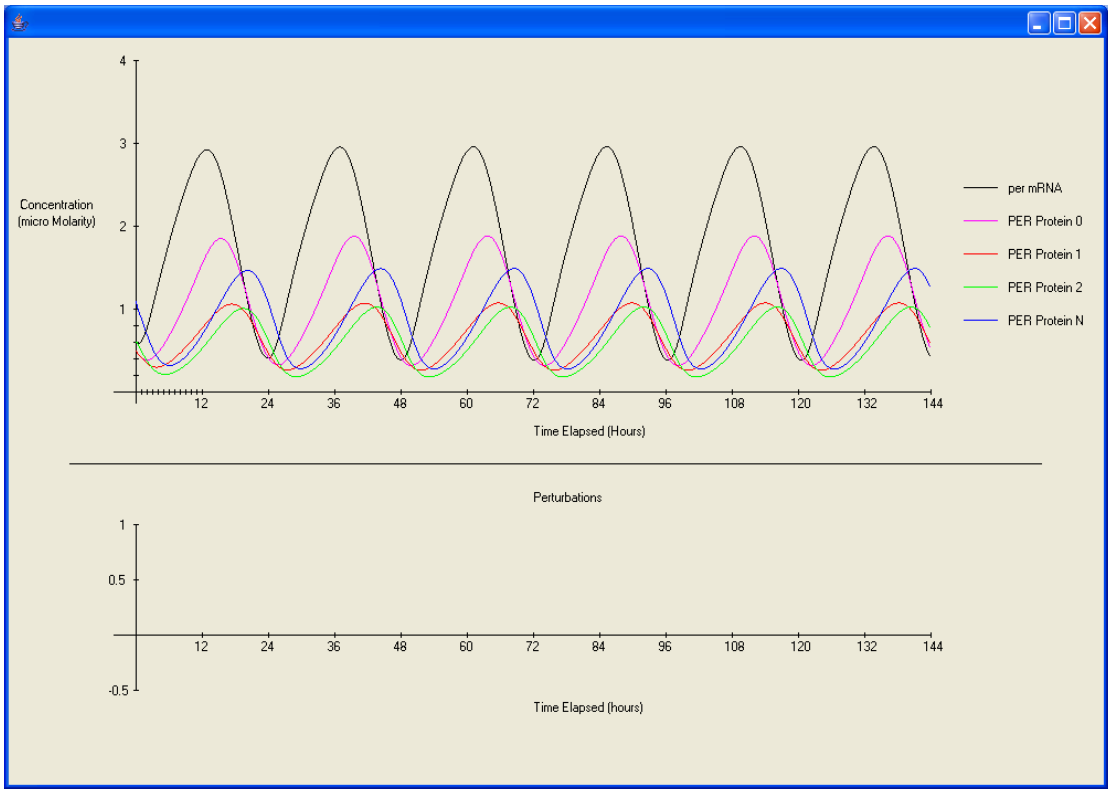


Figure B-4: Model display. The values *M*, P_0, P_1, P_2, and P_N are displayed in the first graph where the x and y axis represents the time elapsed and concentrations, respectively. The second graph displays the disturbances. In the case above, there is no disturbance added (refer to simulations in the text for different cases).

Appendix C: Source Code

1. Welcome.java

```
/**
 * File name: Welcome.java
 *
 * Description: Program start interface
 *
 * Copyright: Copyright (c) 2006
 *
 * Company: Albuquerque Academy
 *
 * Author: Michael Wang
 *
 * Date: Dec 2006
 *
 * @version 1.0
 */

package circadian_rhythms;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Color;
import java.awt.Rectangle;
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class starts the program and describes the project
 */
public class Welcome extends JFrame
{
    JPanel contentPane;
    JLabel jLabel1 = new JLabel();
    JButton bt_start = new JButton();
    JButton bt_quit = new JButton();
    JLabel jLabel2 = new JLabel();
    JLabel jLabel3 = new JLabel();
    JLabel jLabel4 = new JLabel();
    public Welcome()
    {
        try
        {
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            jbInit();
        }
        catch (Exception exception)
        {
            exception.printStackTrace();
        }
    }

    /**
     * Component initialization
     */
    private void jbInit() throws Exception
    {
        contentPane = (JPanel) getContentPane();
        contentPane.setLayout(null);
        setSize(new Dimension(600, 400));
        setTitle("Circadian Rhythms");
        jLabel1.setFont(new java.awt.Font("Dialog", Font.BOLD, 12));
        jLabel1.setForeground(Color.blue);
        jLabel1.setText("Welcome to the Biochemical Oscillator Project.  This project");
        jLabel1.setBounds(new Rectangle(120, 98, 347, 28));
        jLabel2.setFont(new java.awt.Font("Dialog", Font.BOLD, 12));
        jLabel2.setForeground(Color.blue);
        jLabel2.setText("is to simulate a specific oscillator called the Circadian Rhythms ");
        jLabel2.setBounds(new Rectangle(107, 118, 362, 28));
        jLabel3.setFont(new java.awt.Font("Dialog", Font.BOLD, 12));
        jLabel3.setForeground(Color.blue);
        jLabel3.setText("(sleep-wake cycle) and show how evironmental changes");
        jLabel3.setBounds(new Rectangle(107, 138, 362, 28));
        jLabel4.setFont(new java.awt.Font("Dialog", Font.BOLD, 12));
        jLabel4.setForeground(Color.blue);
        jLabel4.setText("affect the oscillator.");
        jLabel4.setBounds(new Rectangle(107, 158, 362, 28));
        bt_start.setBounds(new Rectangle(317, 297, 93, 36));
        bt_start.setText("Start");
        bt_start.addActionListener(new Welcome_bt_start_actionAdapter(this));
        bt_quit.setBounds(new Rectangle(412, 297, 93, 36));
        bt_quit.setText("Quit");
        bt_quit.addActionListener(new Welcome_bt_quit_actionAdapter(this));
        contentPane.add(jLabel1);
        contentPane.add(jLabel2);
        contentPane.add(jLabel3);
        contentPane.add(jLabel4);
        contentPane.add(bt_start);
        contentPane.add(bt_quit);
    }

    /**
     * Start program
     */
    public void bt_start_actionPerformed(ActionEvent e)
    {
        Input input = new Input();
        input.show();
    }

    /**
     * Quit program
     */
    public void bt_quit_actionPerformed(ActionEvent e)
    {
        this.dispose();
    }
}

class Welcome_bt_quit_actionAdapter implements ActionListener
{
    private Welcome adaptee;
    Welcome_bt_quit_actionAdapter(Welcome adaptee)
    {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_quit_actionPerformed(e);
    }
}

class Welcome_bt_start_actionAdapter implements ActionListener
{
    private Welcome adaptee;
    Welcome_bt_start_actionAdapter(Welcome adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_start_actionPerformed(e);
    }
}
```

2. Input.java

```
/**
 * File name: Input.java
 *
 * Description: This interface allows the user to choose dynamic and
 *              non-dynamic simulations to simulate the values given
 *              to each parameter
 *
 * Copyright: Copyright (c) 2006
 *
 * Company: Albuquerque Academy
 *
 * Author: Michael Wang
 *
 * Date: Dec 2006
 *
 * @version 1.0
 */

package circadian_rhythms;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.TitledBorder;
import javax.swing.JLabel;
import java.util.Random;
import java.awt.Dimension;
import java.awt.Font;
import javax.swing.JTextField;
import com.borland.jbcl.layout.XYConstraints;
import com.borland.jbcl.layout.XYLayout;
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JOptionPane;

/**
 * This class creates the interface for inputting values, which are used
 * to calculate the concentrations
 */
```

```
public class Input extends JFrame
{
    JPanel contentPane;
    JLabel jLabel1 = new JLabel();
    JTextField tb_initialM = new JTextField();
    XYLayout xYLayout1 = new XYLayout();
    JLabel jLabel2 = new JLabel();
    JTextField tb_initialP0 = new JTextField();
    JLabel jLabel3 = new JLabel();
    JTextField tb_timeStep = new JTextField();
    JLabel jLabel4 = new JLabel();
    JTextField tb_initialP1 = new JTextField();
    JLabel jLabel5 = new JLabel();
    JTextField tb_initialP2 = new JTextField();
    JLabel jLabel6 = new JLabel();
    JTextField tb_initialPN = new JTextField();
    JTextField tb_vs = new JTextField();
    JLabel jLabel7 = new JLabel();
    JLabel jLabel8 = new JLabel();
    JTextField tb_vm = new JTextField();
    JLabel jLabel9 = new JLabel();
    JTextField tb_vd = new JTextField();
    JLabel jLabel10 = new JLabel();
    JTextField tb_ks = new JTextField();
    JLabel jLabel11 = new JLabel();
    JTextField tb_k1 = new JTextField();
    JLabel jLabel12 = new JLabel();
    JTextField tb_k2 = new JTextField();
    JLabel jLabel13 = new JLabel();
    JTextField tb_V1 = new JTextField();
    JTextField tb_V2 = new JTextField();
    JLabel jLabel14 = new JLabel();
    JTextField tb_V3 = new JTextField();
    JLabel jLabel15 = new JLabel();
    JTextField tb_V4 = new JTextField();
    JLabel jLabel16 = new JLabel();
    JLabel jLabel17 = new JLabel();
    JLabel jLabel18 = new JLabel();
    JTextField tb_K1 = new JTextField();
    JLabel jLabel19 = new JLabel();
    JTextField tb_K2 = new JTextField();
    JLabel jLabel20 = new JLabel();
    JTextField tb_K3 = new JTextField();
    JLabel jLabel21 = new JLabel();
    JTextField tb_K4 = new JTextField();
    JTextField tb_Kd = new JTextField();
    JLabel jLabel23 = new JLabel();
    JTextField tb_Km = new JTextField();
    JLabel jLabel22 = new JLabel();
    JLabel jLabel24 = new JLabel();
    JTextField tb_KI = new JTextField();
    JButton bt_nondynamic = new JButton();
    JButton bt_dynamic = new JButton();
    JLabel jLabel26 = new JLabel();
    JTextField tb_numTicks = new JTextField();
    JButton bt_envdis = new JButton();
    JButton bt_reset = new JButton();
    JLabel jLabel25 = new JLabel();
    JTextField tb_coop = new JTextField();
    Parameters edps = new Parameters();
    public Input()
    {
```

```
    try
    {
        jbInit();
    }
    catch (Exception exception)
    {
        exception.printStackTrace();
    }
}

/**
 * Component Initialization.
 */
private void jbInit() throws Exception
{
    TitledBorder tBorder = new TitledBorder("Input");
    this.setResizable(true);
    this.setTitle("Circadian Rhythms Project Input Gui");
    this.setLocation(136, 147);
    contentPane = (JPanel) getContentPane();
    contentPane.setBorder(tBorder);
    contentPane.setLayout(xYLayout1);
    setSize(new Dimension(880, 570));
    jLabel1.setText("Initial per mRNA (M):");
    tb_initialM.setText("0.6");
    jLabel2.setText("Initial unphosphorylated PER (P0):");
    tb_initialP0.setText("0.5");
    jLabel4.setText("Initial monophosporylated PER (P1):");
    tb_initialP1.setText("0.5");
    jLabel5.setText("Initial bisphosphorylated PER (P2):");
    tb_initialP2.setText("0.6");
    jLabel6.setText("Initial PER (PN):");
    tb_initialPN.setText("1.1");
    jLabel7.setText("Max. Accumulaion Rate (vs):");
    tb_vs.setText("0.76");
    jLabel8.setText("Max. (M) Deg. Rate (vm):");
    tb_vm.setText("0.65");
    jLabel9.setText("Max. (P2) Deg. Rate (vd):");
    tb_vd.setText("0.95");
    jLabel10.setText("Rate Constant of Synthesis (ks):");
    tb_ks.setText("0.38");
    jLabel11.setText("Rate Constant of Transport (k1):");
    tb_k1.setText("1.9");
    jLabel12.setText("Rate Constant of Transport (k2):");
    tb_k2.setText("1.3");
    jLabel13.setText("Max. Conversion Rate, P0 to P1 (V1):");
    tb_V1.setText("3.2");
    jLabel14.setText("Max. Conversion Rate, P1 to P0 (V2):");
    tb_V2.setText("1.58");
    jLabel15.setText("Max. Conversion Rate, P1 to P2 (V3):");
    tb_V3.setText("5.0");
    jLabel16.setText("Max. Conversion Rate, P2 to P1 (V4):");
    tb_V4.setText("2.5");
    jLabel17.setText("** for Conversion, P0 to P1 (K1):");
    tb_K1.setText("2.0");
    jLabel18.setFont(new java.awt.Font("Dialog", Font.BOLD, 12));
    jLabel18.setText("** Michaelis constant");
    jLabel19.setText("** for Conversion, P1 to P0 (K2):");
    tb_K2.setText("2.0");
    jLabel20.setText("** for Conversion, P1 to P2 (K3):");
    tb_K3.setText("2.0");
    jLabel21.setText("** for Conversion, P2 to P1 (K4):");
```

```
tb_K4.setText("2.0");
jLabel23.setText("** for Max. (M) Deg. Rate (Km):");
tb_Km.setText("0.5");
jLabel22.setText("** for Max. (P2) Deg. Rate (Kd):");
tb_Kd.setText("0.2");
jLabel24.setText("Threshold constant (KI):");
tb_KI.setText("1.0");
bt_nondynamic.setText("Non-dynamic Simulation");
bt_nondynamic.addActionListener(new Input_bt_nondynamic_actionAdapter(this));
jLabel26.setText("Y Max:");
tb_numTicks.setToolTipText("");
tb_numTicks.setText("4");
bt_envdis.setText("Environmental Disturbance Input");
bt_envdis.addActionListener(new Input_bt_envdis_actionAdapter(this));
bt_reset.setFont(new java.awt.Font("Dialog", Font.PLAIN, 11));
bt_reset.setText("Reset");
bt_reset.addActionListener(new Input_bt_reset_actionAdapter(this));
jLabel25.setText("Cooperativity (n):");
tb_coop.setText("4");
jLabel3.setText("Timestep (min.):");
tb_timeStep.setToolTipText("");
tb_timeStep.setText("12");
bt_dynamic.setText("Dynamic Simulation");
bt_dynamic.addActionListener(new Input_bt_dynamic_actionAdapter(this));
contentPane.add(jLabel2, new XYConstraints(17, 44, -1, -1));
contentPane.add(jLabel4, new XYConstraints(17, 74, -1, -1));
contentPane.add(jLabel5, new XYConstraints(17, 104, -1, -1));
contentPane.add(jLabel1, new XYConstraints(17, 14, -1, 16));
contentPane.add(jLabel6, new XYConstraints(17, 134, -1, -1));
contentPane.add(jLabel7, new XYConstraints(17, 244, -1, -1));
contentPane.add(tb_vs, new XYConstraints(192, 244, 54, 15));
contentPane.add(tb_vm, new XYConstraints(192, 274, 54, 15));
contentPane.add(jLabel8, new XYConstraints(17, 274, -1, -1));
contentPane.add(jLabel10, new XYConstraints(16, 363, -1, -1));
contentPane.add(jLabel11, new XYConstraints(16, 393, -1, -1));
contentPane.add(jLabel12, new XYConstraints(16, 423, -1, -1));
contentPane.add(tb_k2, new XYConstraints(191, 423, 54, 15));
contentPane.add(tb_k1, new XYConstraints(191, 393, 54, 15));
contentPane.add(tb_ks, new XYConstraints(191, 363, 54, 15));
contentPane.add(tb_vd, new XYConstraints(192, 304, 54, 15));
contentPane.add(jLabel9, new XYConstraints(17, 305, -1, -1));
contentPane.add(jLabel13, new XYConstraints(285, 244, -1, -1));
contentPane.add(jLabel15, new XYConstraints(285, 304, -1, -1));
contentPane.add(tb_V2, new XYConstraints(480, 274, 54, 15));
contentPane.add(jLabel16, new XYConstraints(285, 334, -1, -1));
contentPane.add(jLabel14, new XYConstraints(285, 274, -1, -1));
contentPane.add(tb_V4, new XYConstraints(480, 334, 54, 15));
contentPane.add(tb_V1, new XYConstraints(480, 244, 54, 15));
contentPane.add(tb_V3, new XYConstraints(480, 304, 54, 15));
contentPane.add(jLabel17, new XYConstraints(574, 244, -1, -1));
contentPane.add(jLabel21, new XYConstraints(574, 334, -1, -1));
contentPane.add(jLabel22, new XYConstraints(574, 364, -1, -1));
contentPane.add(jLabel20, new XYConstraints(574, 304, -1, -1));
contentPane.add(jLabel19, new XYConstraints(574, 274, -1, -1));
contentPane.add(jLabel23, new XYConstraints(574, 394, -1, -1));
contentPane.add(jLabel24, new XYConstraints(574, 424, -1, -1));
contentPane.add(jLabel25, new XYConstraints(17, 187, -1, -1));
contentPane.add(tb_coop, new XYConstraints(111, 187, 54, 15));
contentPane.add(jLabel18, new XYConstraints(421, 390, -1, -1));
contentPane.add(tb_K1, new XYConstraints(745, 246, 55, 15));
contentPane.add(tb_K2, new XYConstraints(745, 275, 54, 15));
contentPane.add(tb_K3, new XYConstraints(745, 305, 54, 15));
```

```
    contentPane.add(tb_K4, new XYConstraints(745, 335, 54, 15));
    contentPane.add(tb_Kd, new XYConstraints(745, 364, 54, 14));
    contentPane.add(tb_Km, new XYConstraints(745, 394, 54, 15));
    contentPane.add(tb_KI, new XYConstraints(745, 424, 54, 15));
    contentPane.add(tb_initialM, new XYConstraints(206, 14, 54, 15));
    contentPane.add(tb_initialP0, new XYConstraints(206, 44, 54, 15));
    contentPane.add(tb_initialP1, new XYConstraints(206, 74, 54, 15));
    contentPane.add(tb_initialP2, new XYConstraints(206, 104, 54, 15));
    contentPane.add(tb_initialPN, new XYConstraints(206, 134, 54, 15));
    contentPane.add(jLabel3, new XYConstraints(507, 200, -1, -1));
    contentPane.add(tb_timeStep, new XYConstraints(591, 200, 54, 15));
    contentPane.add(jLabel26, new XYConstraints(507, 180, -1, -1));
    contentPane.add(tb_numTicks, new XYConstraints(551, 180, 54, 15));
    contentPane.add(bt_dynamic, new XYConstraints(558, 86, 150, 36));
    contentPane.add(bt_nondynamic, new XYConstraints(409, 86, 150, 36));
    contentPane.add(bt_envdis, new XYConstraints(409, 46, 298, 36));
    contentPane.add(bt_reset, new XYConstraints(505, 126, 109, 36));
}

int numTicks;      // Number of y-axis ticks
double timeStep;   // Time between the current and previous concentrations
int n;             // Degree of cooperativity
double Mi;         // Initial per mRNA (M)
double P0i;        // Initial unphosphorylated period protein (PER) (P0)
double P1i;        // Initial monophosphorylated period protein (PER) (P1)
double P2i;        // Initial biphosphorylated period protein (PER) (P2)
double PNi;        // Initial period protein (PER) (PN)
double vs;         // Maximum rate of M accumulation
double vm;         // Maximum rate of M degradation
double vd;         // Maximum rate of P2 degradation
double ks;         // Rate constant characterizing the synthesis of PER
double k1;         // Rate constant for transportation into nucleus
double k2;         // Rate constant for transportation out of the nucleus into the cytosol
double V1;         // Rate of conversion (P0 --> P1)
double V2;         // Rate of conversion (P1 --> P0)
double V3;         // Rate of conversion (P1 --> P2)
double V4;         // Rate of conversion (P2 --> P1)
double K1;         // Michaelis constant for describing P0 --> P1
double K2;         // Michaelis constant for describing P1 --> P0
double K3;         // Michaelis constant for describing P1 --> P2
double K4;         // Michaelis constant for describing P2 --> P1
double Kd;         // Michaelis constant for describing P2 degradation
double Km;         // Michaelis constant for describing M degradation
double KI;         // Threshold constant

/**
 * Button for non-dynamic simulation
 */
public void bt_nondynamic_actionPerformed(ActionEvent e)
{
    calc(true);
}

/**
 * Button for dynamic simulation
 */
public void bt_dynamic_actionPerformed(ActionEvent e)
{
    calc(false);
}

/**
```

```
 * RandomPerturbation is called when need to imput the amplitude of
 * disturbance of a calculation parameter.
 */
public void bt_envdis_actionPerformed(ActionEvent e)
{
    RandomPerturbation ED = new RandomPerturbation(edps);
    ED.show();
}

/**
 * Set calculation parameters back to original when "reset" is clicked.
 */
public void bt_reset_actionPerformed(ActionEvent e)
{
    tb_initialM.setText("0.6");
    tb_initialP0.setText("0.5");
    tb_initialP1.setText("0.5");
    tb_initialP2.setText("0.6");
    tb_initialPN.setText("1.1");
    tb_vs.setText("0.76");
    tb_vm.setText("0.65");
    tb_vd.setText("0.95");
    tb_ks.setText("0.38");
    tb_k1.setText("1.9");
    tb_k2.setText("1.3");
    tb_V1.setText("3.2");
    tb_V2.setText("1.58");
    tb_V3.setText("5.0");
    tb_V4.setText("2.5");
    tb_K1.setText("2.0");
    tb_K2.setText("2.0");
    tb_K3.setText("2.0");
    tb_K4.setText("2.0");
    tb_Kd.setText("0.2");
    tb_Km.setText("0.5");
    tb_KI.setText("1.0");
    tb_numTicks.setText("4");
    tb_timeStep.setText("12");
    tb_coop.setText("4");
}

/**
 * Simulation variables M (per mRNA), P0 (unphosphorylated PER Protein),
 * P1 (monophosphorylated PER Protein), P2 (biphosphorylated PER Protein),
 * and PN (PER Protein) are calculated using an explicit time scheme.
 */
public void calc(boolean isNonDynamic)
{
    updateVars();
    checkMaxMin();

    Random r = new Random();
    int numLoops = 720;
    float et = 0f;

    double M[] = new double[numLoops];
      M[0] = Mi;
    double P0[] = new double[numLoops];
      P0[0] = P0i;
    double P1[] = new double[numLoops];
      P1[0] = P1i;
    double P2[] = new double[numLoops];
```

```
      P2[0] = P2i;
    double PN[] = new double[numLoops];
      PN[0] = PNi;

    double dt = timeStep / 60.0; //Time step converted from min. to hrs.
    double t, t1, t2, t3, t4;
    float stoE[] = new float[numLoops]; //array storing all values of et
    for (int i = 1; i < numLoops; i++) {
        //If 1, add a new disturbance every disturbance interval
        //If not 1, only simulate one of these intervals
        if (edps.multi_single == 1)
        {
            if (i % (edps.durationT / timeStep) == 0)
                et = r.nextFloat() - 0.5f;

            stoE[i] = et;
        }
        else if (edps.multi_single == -1)
        {
            if ((i >= numLoops / 2 - edps.durationT / (2 * timeStep)) && (i <= numLoops / 2 + edps.durationT / (2 *
timeStep)))
                et = 1;
            else
                et = 0;

            stoE[i] = et;
        }

        //t, t1, t2, t3, t4 replace repeated expressions in the five kinetic equations.
        t = (V1 + edps.EDV1 * et) * P0[i - 1] / ((K1 + edps.EDK1 * et) + P0[i - 1]);
        t1 = (V2 + edps.EDV2 * et) * P1[i - 1] / ((K2 + edps.EDK2 * et) + P1[i - 1]);
        t2 = (V3 + edps.EDV3 * et) * P1[i - 1] / ((K3 + edps.EDK3 * et) + P1[i - 1]);
        t3 = (V4 + edps.EDV4 * et) * P2[i - 1] / ((K4 + edps.EDK4 * et) + P2[i - 1]);
        t4 = (k1 + edps.EDk1 * et) * P2[i - 1] - (k2 + edps.EDk2 * et) * PN[i - 1];

        M[i] = ((vs + edps.EDvs * et) * Math.pow(KI + edps.EDKI * et, n) /
                (Math.pow(KI + edps.EDKI * et, n) + Math.pow(PN[i - 1], n))
                - (vm + edps.EDvm * et) * M[i - 1] / ((Km + edps.EDKm * et) + M[i - 1]))
               * dt + M[i - 1];
        P0[i] = ((ks + edps.EDks * et) * M[i - 1] - t + t1) * dt + P0[i - 1];
        P1[i] = (t - t1 - t2 + t3) * dt + P1[i - 1];
        P2[i] = (t2 - t3 - t4 - (vd + edps.EDvd * et) * P2[i - 1] /
                 ((Kd + edps.EDKd * et) + P2[i - 1])) * dt + P2[i - 1];
        PN[i] = t4 * dt + PN[i - 1];
    }

    //Initialize graphs and display oscillations.
    JFrame disp = new JFrame();
    //Non-dynamic
    if (isNonDynamic == true)
    {
        Non_Dynamic simVars = new Non_Dynamic();
         simVars.simulation(M, P0, P1, P2, PN, stoE, numLoops, numTicks,
                         timeStep);
        disp.setSize(1000, 710);
        disp.setLocation(76, 77);
        disp.getContentPane().add(simVars);
        disp.show();
    }
    //Dynamic
    else
    {
        Dynamic dymVars = new Dynamic();
         dymVars.dynamic(M, P0, P1, P2, PN, vs, vm, vd, ks, k1, k2, V1, V2,
                    V3, V4, K1, K2, K3, K4, Kd, Km, KI, n, numTicks, timeStep, stoE, edps);
        disp.setSize(1000, 700);
        disp.setLocation(76, 77);
        disp.getContentPane().add(dymVars);
        disp.show();
    }
}

/**
 * Model parameters are extracted and converted (String --> number)
 */
public void updateVars()
{
    Mi = Double.parseDouble(tb_initialM.getText());
    P0i = Double.parseDouble(tb_initialP0.getText());
    P1i = Double.parseDouble(tb_initialP1.getText());
    P2i = Double.parseDouble(tb_initialP2.getText());
    PNi = Double.parseDouble(tb_initialPN.getText());
    vs = Double.parseDouble(tb_vs.getText());
    vm = Double.parseDouble(tb_vm.getText());
    vd = Double.parseDouble(tb_vd.getText());
    ks = Double.parseDouble(tb_ks.getText());
    k1 = Double.parseDouble(tb_k1.getText());
    k2 = Double.parseDouble(tb_k2.getText());
    V1 = Double.parseDouble(tb_V1.getText());
    V2 = Double.parseDouble(tb_V2.getText());
    V3 = Double.parseDouble(tb_V3.getText());
    V4 = Double.parseDouble(tb_V4.getText());
    K1 = Double.parseDouble(tb_K1.getText());
    K2 = Double.parseDouble(tb_K2.getText());
    K3 = Double.parseDouble(tb_K3.getText());
    K4 = Double.parseDouble(tb_K4.getText());
    Kd = Double.parseDouble(tb_Kd.getText());
    Km = Double.parseDouble(tb_Km.getText());
    KI = Double.parseDouble(tb_KI.getText());
    numTicks = Integer.parseInt(tb_numTicks.getText());
    timeStep = Double.parseDouble(tb_timeStep.getText());
    n = Integer.parseInt(tb_coop.getText());
}

/**
 * Check whether a disturbance is greater than the max. or less than the min.
 */
public void checkMaxMin()
{
    /**
     * Check whether the disturbance amplitude of each basal parameter
     * exceeds its limit or is less than 0 for multiple pulses.
     * (Limit = 2 * (calculation parameter))
     */
    if (edps.multi_single == 1)
    {
        String strWarning =
                "Any disturbance amplitudes exceeding their max. or less than 0 will be set to that max.";
        String strMax = "\n(max. = 2 * (calculation parameter))";
        JOptionPane.showMessageDialog(null, strWarning + strMax, "Warning", 2);

        if (edps.EDvs > vs * 2 || edps.EDvs < 0)
            edps.EDvs = vs * 2;
        if (edps.EDvm > vm * 2 || edps.EDvm < 0)
            edps.EDvm = vm * 2;
        if (edps.EDvd > vd * 2 || edps.EDvd < 0)
            edps.EDvd = vd * 2;
        if (edps.EDks > ks * 2 || edps.EDks < 0)
            edps.EDks = ks * 2;
        if (edps.EDk1 > k1 * 2 || edps.EDk1 < 0)
            edps.EDk1 = k1 * 2;
        if (edps.EDk2 > k2 * 2 || edps.EDk2 < 0)
            edps.EDk2 = k2 * 2;
        if (edps.EDV1 > V1 * 2 || edps.EDV1 < 0)
            edps.EDV1 = V1 * 2;
        if (edps.EDV2 > V2 * 2 || edps.EDV2 < 0)
            edps.EDV2 = V2 * 2;
        if (edps.EDV3 > V3 * 2 || edps.EDV3 < 0)
            edps.EDV3 = V3 * 2;
        if (edps.EDV4 > V4 * 2 || edps.EDV4 < 0)
            edps.EDV4 = V4 * 2;
        if (edps.EDK1 > K1 * 2 || edps.EDK1 < 0)
            edps.EDK1 = K1 * 2;
        if (edps.EDK2 > K2 * 2 || edps.EDK2 < 0)
            edps.EDK2 = K2 * 2;
        if (edps.EDK3 > K3 * 2 || edps.EDK3 < 0)
            edps.EDK3 = K3 * 2;
        if (edps.EDK4 > K4 * 2 || edps.EDK4 < 0)
            edps.EDK4 = K4 * 2;
        if (edps.EDKd > Kd * 2 || edps.EDKd < 0)
            edps.EDKd = Kd * 2;
        if (edps.EDKm > Km * 2 || edps.EDKm < 0)
            edps.EDKm = Km * 2;
        if (edps.EDKI > KI * 2 || edps.EDKI < 0)
            edps.EDKI = KI * 2;
    }

    /**
     * Check whether the disturbance amplitude of each basal parameter
     * is less than its min for a single pulse.
     * (min = -(calculation parameter) * 2)
     */
    if (edps.multi_single == -1)
    {
        String strWarning1 =
                "Any disturbance amplitudes less than their min. will be set to that min.";
        String strMin = "\n(min. = -(calculation parameter))";
        JOptionPane.showMessageDialog(null, strWarning1 + strMin, "Warning", 2);

        if (edps.EDvs < -vs)
            edps.EDvs = -vs;
        if (edps.EDvm < -vm)
            edps.EDvm = -vm;
        if (edps.EDvd < -vd)
            edps.EDvd = -vd;
        if (edps.EDks < -ks)
            edps.EDks = -ks;
        if (edps.EDk1 < -k1)
            edps.EDk1 = -k1;
        if (edps.EDk2 < -k2)
            edps.EDk2 = -k2;
        if (edps.EDV1 < -V1)
            edps.EDV1 = -V1;
        if (edps.EDV2 < -V2)
            edps.EDV2 = -V2;
        if (edps.EDV3 < -V3)
```

```
                edps.EDV3 = -V3;
            if (edps.EDV4 < -V4)
                edps.EDV4 = -V4;
            if (edps.EDK1 < -K1)
                edps.EDK1 = -K1;
            if (edps.EDK2 < -K2)
                edps.EDK2 = -K2;
            if (edps.EDK3 < -K3)
                edps.EDK3 = -K3;
            if (edps.EDK4 < -K4)
                edps.EDK4 = -K4;
            if (edps.EDKd < -Kd)
                edps.EDKd = -Kd;
            if (edps.EDKm < -Km)
                edps.EDKm = -Km;
            if (edps.EDKI < -KI)
                edps.EDKI = -KI;
        }
    }
}

class Input_bt_dynamic_actionAdapter implements ActionListener
{
    private Input adaptee;
    Input_bt_dynamic_actionAdapter(Input adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_dynamic_actionPerformed(e);
    }
}

class Input_bt_reset_actionAdapter implements ActionListener
{
    private Input adaptee;
    Input_bt_reset_actionAdapter(Input adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_reset_actionPerformed(e);
    }
}

class Input_bt_envdis_actionAdapter implements ActionListener
{
    private Input adaptee;
    Input_bt_envdis_actionAdapter(Input adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_envdis_actionPerformed(e);
    }
}
```

```
class Input_bt_nondynamic_actionAdapter implements ActionListener
{
    private Input adaptee;
    Input_bt_nondynamic_actionAdapter(Input adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_nondynamic_actionPerformed(e);
    }
}
```

3. RandomPerturbation.java

```
/**
 * File name: RandomPerturbation.java
 *
 * Description: This interface allows the user to input the amplitudes
 *              of disturbances, and save them into the class Parameters.
 *
 * Copyright: Copyright (c) 2006
 *
 * Company: Albuquerque Academy
 *
 * Author: Michael Wang
 *
 * Date: Dec 2006
 *
 * @version 1.0
 */

package circadian_rhythms;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.TitledBorder;
import javax.swing.JLabel;
import java.awt.Dimension;
import java.awt.Font;
import com.borland.jbcl.layout.XYLayout;
import com.borland.jbcl.layout.XYConstraints;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This provides an interface for the user to input disturbance amplitudes
 */
public class RandomPerturbation extends JFrame
{
    JPanel contentPane;
    XYLayout xYLayout1 = new XYLayout();
    JLabel jLabel1 = new JLabel();
    JTextField tb_EDvs = new JTextField();
    JLabel jLabel2 = new JLabel();
    JTextField tb_EDvm = new JTextField();
    JLabel jLabel3 = new JLabel();
```

```
JTextField tb_EDvd = new JTextField();
JLabel jLabel4 = new JLabel();
JTextField tb_EDks = new JTextField();
JLabel jLabel5 = new JLabel();
JLabel jLabel6 = new JLabel();
JTextField tb_EDk1 = new JTextField();
JTextField tb_EDk2 = new JTextField();
JLabel jLabel7 = new JLabel();
JTextField tb_EDV1 = new JTextField();
JLabel jLabel8 = new JLabel();
JTextField tb_EDV2 = new JTextField();
JLabel jLabel9 = new JLabel();
JTextField tb_EDV3 = new JTextField();
JLabel jLabel10 = new JLabel();
JTextField tb_EDV4 = new JTextField();
JButton bt_multi = new JButton();
JLabel jLabel11 = new JLabel();
JLabel jLabel12 = new JLabel();
JTextField tb_t = new JTextField();
JLabel jLabel13 = new JLabel();
JButton bt_reset = new JButton();
JLabel jLabel14 = new JLabel();
JLabel jLabel15 = new JLabel();
JTextField tb_EDK1 = new JTextField();
JLabel jLabel16 = new JLabel();
JTextField tb_EDK2 = new JTextField();
JLabel jLabel17 = new JLabel();
JTextField tb_EDK3 = new JTextField();
JLabel jLabel18 = new JLabel();
JTextField tb_EDK4 = new JTextField();
JLabel jLabel19 = new JLabel();
JTextField tb_EDKd = new JTextField();
JLabel jLabel20 = new JLabel();
JTextField tb_EDKm = new JTextField();
JLabel jLabel21 = new JLabel();
JTextField tb_EDKI = new JTextField();
JLabel jLabel22 = new JLabel();
JButton bt_single = new JButton();
JButton bt_none = new JButton();
Parameters edps2;
public RandomPerturbation(Parameters edps1)
{
    try
    {
        edps2 = edps1;
        jbInit();
    }
    catch (Exception exception)
    {
        exception.printStackTrace();
    }
}

/**
 * Component Initialization
 */
private void jbInit() throws Exception
{
    contentPane = (JPanel) getContentPane();
    TitledBorder tBorder = new TitledBorder("Environmental Disturbance");
    this.setTitle("Disturbance Amplitude Input Gui");
    this.setLocation(226, 202);
```

```
contentPane.setBorder(tBorder);
contentPane.setLayout(xYLayout1);
setSize(new Dimension(700, 460));
jLabel1.setText("* of Accummulation Pate (vs):");
jLabel2.setText("* of per mRNA Deg. Rate (vm):");
jLabel3.setText("* of PER Protein 2 Deg. Rate (vd):");
jLabel4.setText("* of Rate Constant of Synthesis (ks):");
jLabel5.setText("* of Rate Constant of Transport (k1):");
jLabel6.setText("* of Rate Constant of Transport (k2):");
jLabel7.setText("* Conversion Rate, P0 to P1 (V1):");
jLabel8.setText("* Conversion Rate, P1 to P0 (V2):");
jLabel9.setText("* Conversion Rate, P1 to P2 (V3):");
jLabel10.setText("* Conversion Rate, P2 to P1 (V4):");
bt_multi.setText("Save (multiple pulses)");
bt_multi.addActionListener(new RandomPerturbation_bt_multi_actionAdapter(this));
jLabel11.setFont(new java.awt.Font("Dialog", Font.BOLD, 12));
jLabel11.setText("* Disturbance Magnitude");
jLabel12.setText("Duration of Disturbance:");
jLabel13.setText("Minutes");
jLabel14.setFont(new java.awt.Font("Dialog", Font.BOLD, 12));
jLabel14.setText("** Michaelis constant");
jLabel15.setText("** for Conversion,  P0 to P1 (K1):");
jLabel16.setText("** for Conversion,  P1 to P0 (K2):");
jLabel17.setText("** for Conversion,  P1 to P2 (K3):");
jLabel18.setText("** for Conversion,  P2 to P1 (K4):");
jLabel19.setText("** for Max. (P2) Deg. Rate (Kd):");
jLabel20.setText("** for Max. (M) Deg. Rate (Km):");
jLabel21.setText("Threshold constant (KI):");
jLabel22.setFont(new java.awt.Font("Dialog", Font.BOLD, 12));
jLabel22.setText("Multiple pulses: amplitude must be >= 0");
bt_reset.setText("Reset");
bt_reset.addActionListener(new RandomPerturbation_bt_reset_actionAdapter(this));
bt_single.setText("Save (single pulse)");
bt_single.addActionListener(new RandomPerturbation_bt_single_actionAdapter(this));
bt_none.setText("Save (none)");
bt_none.addActionListener(new RandomPerturbation_bt_none_actionAdapter(this));
//Set values to ones previously inputed
tb_EDvm.setText(edps2.EDvm + "");
tb_EDvs.setText(edps2.EDvs + "");
tb_EDvd.setText(edps2.EDvd + "");
tb_EDks.setText(edps2.EDks + "");
tb_EDk1.setText(edps2.EDk1 + "");
tb_EDk2.setText(edps2.EDk2 + "");
tb_EDV1.setText(edps2.EDV1 + "");
tb_EDV2.setText(edps2.EDV2 + "");
tb_EDV3.setText(edps2.EDV3 + "");
tb_EDV4.setText(edps2.EDV4 + "");
tb_EDK1.setText(edps2.EDK1 + "");
tb_EDK2.setText(edps2.EDK2 + "");
tb_EDK3.setText(edps2.EDK3 + "");
tb_EDK4.setText(edps2.EDK4 + "");
tb_EDKd.setText(edps2.EDKd + "");
tb_EDKm.setText(edps2.EDKm + "");
tb_EDKI.setText(edps2.EDKI + "");
tb_t.setText(edps2.durationT + "");
contentPane.add(jLabel2, new XYConstraints(18, 24, -1, -1));
contentPane.add(jLabel3, new XYConstraints(18, 44, -1, -1));
contentPane.add(jLabel1, new XYConstraints(18, 4, 159, 16));
contentPane.add(jLabel7, new XYConstraints(18, 94, -1, -1));
contentPane.add(jLabel8, new XYConstraints(18, 114, -1, -1));
contentPane.add(jLabel9, new XYConstraints(19, 134, -1, -1));
contentPane.add(jLabel10, new XYConstraints(18, 154, -1, -1));
```

```
    contentPane.add(tb_EDvs, new XYConstraints(200, 4, 54, 15));
    contentPane.add(tb_EDvm, new XYConstraints(200, 24, 54, 15));
    contentPane.add(tb_EDvd, new XYConstraints(200, 44, 54, 15));
    contentPane.add(tb_EDV1, new XYConstraints(200, 94, 54, 15));
    contentPane.add(tb_EDV2, new XYConstraints(200, 114, 54, 15));
    contentPane.add(tb_EDV3, new XYConstraints(200, 134, 54, 15));
    contentPane.add(tb_EDV4, new XYConstraints(200, 154, 54, 15));
    contentPane.add(jLabel12, new XYConstraints(21, 215, -1, -1));
    contentPane.add(tb_t, new XYConstraints(144, 215, 54, 15));
    contentPane.add(jLabel13, new XYConstraints(200, 215, -1, -1));
    contentPane.add(jLabel4, new XYConstraints(307, 4, -1, -1));
    contentPane.add(tb_EDks, new XYConstraints(504, 4, 54, 15));
    contentPane.add(tb_EDk1, new XYConstraints(504, 24, 54, 15));
    contentPane.add(tb_EDk2, new XYConstraints(504, 44, 54, 15));
    contentPane.add(jLabel6, new XYConstraints(307, 44, -1, -1));
    contentPane.add(jLabel5, new XYConstraints(307, 24, -1, -1));
    contentPane.add(jLabel15, new XYConstraints(307, 94, -1, -1));
    contentPane.add(tb_EDK1, new XYConstraints(486, 94, 54, 15));
    contentPane.add(tb_EDK2, new XYConstraints(486, 114, 54, 15));
    contentPane.add(jLabel16, new XYConstraints(307, 114, -1, -1));
    contentPane.add(jLabel17, new XYConstraints(307, 134, -1, -1));
    contentPane.add(tb_EDK3, new XYConstraints(486, 134, 54, 15));
    contentPane.add(tb_EDK4, new XYConstraints(486, 153, 54, 15));
    contentPane.add(jLabel18, new XYConstraints(306, 154, -1, -1));
    contentPane.add(jLabel19, new XYConstraints(306, 184, -1, -1));
    contentPane.add(tb_EDKd, new XYConstraints(476, 184, 54, 15));
    contentPane.add(tb_EDKm, new XYConstraints(476, 204, 54, 15));
    contentPane.add(jLabel20, new XYConstraints(306, 204, -1, -1));
    contentPane.add(jLabel21, new XYConstraints(306, 224, -1, -1));
    contentPane.add(tb_EDKI, new XYConstraints(476, 224, 54, 15));
    contentPane.add(jLabel11, new XYConstraints(89, 280, -1, -1));
    contentPane.add(jLabel14, new XYConstraints(89, 305, -1, -1));
    contentPane.add(jLabel22, new XYConstraints(374, 266, -1, -1));
    contentPane.add(bt_none, new XYConstraints(345, 331, 139, 38));
    contentPane.add(bt_reset, new XYConstraints(486, 331, 139, 38));
    contentPane.add(bt_single, new XYConstraints(486, 288, 139, 38));
    contentPane.add(bt_multi, new XYConstraints(345, 288, 139, 38));
}

/**
 * Call updateVars(), then set multi_single to 1 and call checkChange().
 */
public void bt_multi_actionPerformed(ActionEvent e)
{
    updateVars();
    edps2.multi_single = 1;
    checkChange();
}

/**
 * Call updateVars(), then set multi_single to -1 and call checkChange().
 */
public void bt_single_actionPerformed(ActionEvent e)
{
    updateVars();
    edps2.multi_single = -1;
    checkChange();
}

/**
 * Reset amplitudes to 0 and save (no pulse).
 */
```

```
public void bt_none_actionPerformed(ActionEvent e)
{
    edps2.EDvs = 0.0;
    edps2.EDvm = 0.0;
    edps2.EDvd = 0.0;
    edps2.EDks = 0.0;
    edps2.EDk1 = 0.0;
    edps2.EDk2 = 0.0;
    edps2.EDV1 = 0.0;
    edps2.EDV2 = 0.0;
    edps2.EDV3 = 0.0;
    edps2.EDV4 = 0.0;
    edps2.EDK1 = 0.0;
    edps2.EDK2 = 0.0;
    edps2.EDK3 = 0.0;
    edps2.EDK4 = 0.0;
    edps2.EDKd = 0.0;
    edps2.EDKm = 0.0;
    edps2.EDKI = 0.0;
    edps2.durationT = 24;
    edps2.multi_single = 0;

    this.dispose();
}

/**
 * Set values back to original when "reset" is clicked.
 */
public void bt_reset_actionPerformed(ActionEvent e)
{
    tb_EDvs.setText("0.0");
    tb_EDvm.setText("0.0");
    tb_EDvd.setText("0.0");
    tb_EDks.setText("0.0");
    tb_EDk1.setText("0.0");
    tb_EDk2.setText("0.0");
    tb_EDV1.setText("0.0");
    tb_EDV2.setText("0.0");
    tb_EDV3.setText("0.0");
    tb_EDV4.setText("0.0");
    tb_EDK1.setText("0.0");
    tb_EDK2.setText("0.0");
    tb_EDK3.setText("0.0");
    tb_EDK4.setText("0.0");
    tb_EDKd.setText("0.0");
    tb_EDKm.setText("0.0");
    tb_EDKI.setText("0.0");
    tb_t.setText("24");
}

/**
 * Method for storing newly inputed values
 */
private void updateVars()
{
    edps2.EDvs = Double.parseDouble(tb_EDvs.getText());
    edps2.EDvm = Double.parseDouble(tb_EDvm.getText());
    edps2.EDvd = Double.parseDouble(tb_EDvd.getText());
    edps2.EDks = Double.parseDouble(tb_EDks.getText());
    edps2.EDk1 = Double.parseDouble(tb_EDk1.getText());
    edps2.EDk2 = Double.parseDouble(tb_EDk2.getText());
    edps2.EDV1 = Double.parseDouble(tb_EDV1.getText());
```

```
        edps2.EDV2 = Double.parseDouble(tb_EDV2.getText());
        edps2.EDV3 = Double.parseDouble(tb_EDV3.getText());
        edps2.EDV4 = Double.parseDouble(tb_EDV4.getText());
        edps2.EDK1 = Double.parseDouble(tb_EDK1.getText());
        edps2.EDK2 = Double.parseDouble(tb_EDK2.getText());
        edps2.EDK3 = Double.parseDouble(tb_EDK3.getText());
        edps2.EDK4 = Double.parseDouble(tb_EDK4.getText());
        edps2.EDKd = Double.parseDouble(tb_EDKd.getText());
        edps2.EDKm = Double.parseDouble(tb_EDKm.getText());
        edps2.EDKI = Double.parseDouble(tb_EDKI.getText());
        edps2.durationT = Integer.parseInt(tb_t.getText());
    }

    /**
     * If all values remain unchanged, set multi_single = 0 (no pulse)
     */
    private void checkChange()
    {
        if(edps2.EDvs == 0.0 && edps2.EDvm == 0.0 && edps2.EDvd == 0.0 && edps2.EDks == 0.0
          && edps2.EDk1 == 0.0 && edps2.EDk2 == 0.0 && edps2.EDV1 == 0.0 && edps2.EDV2 == 0.0
          && edps2.EDV3 == 0.0 && edps2.EDV4 == 0.0 && edps2.EDK1 == 0.0 && edps2.EDK2 == 0.0
          && edps2.EDK3 == 0.0 && edps2.EDK4 == 0.0 && edps2.EDKd == 0.0 && edps2.EDKm == 0.0
          && edps2.EDKI == 0.0 && (edps2.durationT >= 24 || edps2.durationT < 24))
        {
            edps2.multi_single = 0;
        }

        this.dispose();
    }
}

class RandomPerturbation_bt_none_actionAdapter implements ActionListener
{
    private RandomPerturbation adaptee;
    RandomPerturbation_bt_none_actionAdapter(RandomPerturbation adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_none_actionPerformed(e);
    }
}

class RandomPerturbation_bt_single_actionAdapter implements ActionListener
{
    private RandomPerturbation adaptee;
    RandomPerturbation_bt_single_actionAdapter(RandomPerturbation adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_single_actionPerformed(e);
    }
}

class RandomPerturbation_bt_reset_actionAdapter implements ActionListener
{
    private RandomPerturbation adaptee;
    RandomPerturbation_bt_reset_actionAdapter(RandomPerturbation adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_reset_actionPerformed(e);
    }
}

class RandomPerturbation_bt_multi_actionAdapter implements ActionListener
{
    private RandomPerturbation adaptee;
    RandomPerturbation_bt_multi_actionAdapter(RandomPerturbation adaptee)
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.bt_multi_actionPerformed(e);
    }
}
```

4. Non_Dynamic.java

```
/**
 * File name: Non_Dynamic.java
 *
 * Description: This applet includes two graphs.  One displays
 *              a non-dynamic simulation from 0 to 144 hours
 *              while the other displays the disturbance,
 *              if any, over this interval.
 *
 * Copyright: Copyright (c) 2006
 *
 * Company: Albuquerque Academy
 *
 * Author: Michael Wang
 *
 * Date: Dec 2006
 *
 * @version 1.0
 */

package circadian_rhythms;

import java.awt.Graphics;
import java.awt.Color;
import java.applet.Applet;

/**
 * This class displays the simulation variables, M, P0, P1, P2, and PN,
 * and the "noise", et.
 */
public class Non_Dynamic extends Applet
{
    double M[], P0[], P1[], P2[], PN[], timeStep;
    float e[];
    int numLoops, numTicks;
```

```
public void simulation(double M2[], double P02[], double P12[], double P22[], double PN2[], float e1[],
                int numLoops2, int numTicks2, double timeStep2)
{
   numLoops = numLoops2;
   numTicks = numTicks2;
   timeStep = timeStep2;

   //Set the size of the arrays storing the simulation values
   M = new double[numLoops];
   P0 = new double[numLoops];
   P1 = new double[numLoops];
   P2 = new double[numLoops];
   PN = new double[numLoops];
   e = new float[numLoops];

   //Initialize each arrays
   for(int i = 0; i < numLoops2; i++)
   {
      M[i] = M2[i];
      P0[i] = P02[i];
      P1[i] = P12[i];
      P2[i] = P22[i];
      PN[i] = PN2[i];
      e[i] = e1[i];
   }
}

public void paint(Graphics g)
{
   this.setSize(1000, 650);
   int xScl = (int)(12 * 60.0 / timeStep);       //# of pixels per 12 hrs.

   g.translate(115, 320);                        //move center from (0,0) to (115,320)

   //x and y axis of the first graph
   g.drawLine(-20, 0, 720, 0);
   g.drawLine(0, 10, 0, -300);
   g.drawString("Time Elapsed (Hours)", 360, 40);
   g.drawString("Concentration", -105, -165);
   g.drawString("(micro Molarity)", -107, -150);

   //Major ticks of the y-axis (first graph).  Vertical scaling factor = 300 / numTicks
   for(int yAxis = 1; yAxis < numTicks + 1; yAxis++)
   {
      g.drawLine(0, -yAxis * 300 / numTicks, 3, -yAxis * 300 / numTicks);
      g.drawString(yAxis + "", -15, -yAxis * 300 / numTicks + 5);
   }

   //Major ticks of the x-axis (first graph)
   for(int xAxis = 1; xAxis < 720 / xScl + 1; xAxis++)
   {
      g.drawLine(xAxis * xScl, 0, xAxis * xScl, -3);
      g.drawString(xAxis * 12 + "", xAxis * xScl - 7, 15);
   }

   //Minor ticks of the y-axis (first graph)
   for(int yAxis = 1; yAxis < 5; yAxis++)
   {
      g.drawLine(0, -yAxis * 300 / numTicks / 5, 3, -yAxis * 300 / numTicks / 5);
   }

   //minor ticks of the x-axis (first graph)
```

```
for(int xAxis = 1; xAxis < 12; xAxis++)
{
   g.drawLine(xAxis * xScl / 12, 0, xAxis * xScl / 12, -3);
}


g.drawLine(-60, 65, 820, 65);       //line separating the two graphs

//x and y axis of the second graph
g.drawLine(-20, 220, 720, 220);
g.drawLine(0, 120, 0, 270);
g.drawString("Perturbations", 360, 100);
g.drawString("Time Elapsed (hours)", 360, 290);

//y-axis ticks (second graph)
g.drawLine(0, 270, 3, 270);
 g.drawString("-0.5", -25, 275);
g.drawLine(0, 170, 3, 170);
 g.drawString("0.5", -25, 175);
g.drawLine(0, 120, 3, 120);
 g.drawString("1", -15, 125);

//x-axis ticks (second graph)
for(int i = 1; i < 720 / xScl + 1; i++)
{
   g.drawLine(i * xScl, 220, i * xScl, 217);
   g.drawString(i * 12 + "", i * xScl - 7, 235);
}


//Graph points (x,M[x]), (x,P0[x]), (x,P1[x]), (x,P2[x]), (x,PN[x]), and (x,e[x])
for(int x = 0; x < numLoops - 1; x++)
{
   //Draw line from the point at x to the point at x + 1.  Vertical scaling factor: 300 / numticks
   g.setColor(Color.black);
    g.drawLine(x, -(int)(M[x] * 300 / numTicks), x + 1, -(int)(M[x + 1] * 300 / numTicks));
    g.drawLine(x, -(int)(e[x] * 100) + 220, x + 1, -(int)(e[x + 1] * 100) + 220);
   g.setColor(Color.magenta);
    g.drawLine(x, -(int)(P0[x] * 300 / numTicks), x + 1, -(int)(P0[x + 1] * 300 / numTicks));
   g.setColor(Color.red);
    g.drawLine(x, -(int)(P1[x] * 300 / numTicks), x + 1, -(int)(P1[x + 1] * 300 / numTicks));
   g.setColor(Color.green);
    g.drawLine(x, -(int)(P2[x] * 300 / numTicks), x + 1, -(int)(P2[x + 1] * 300 / numTicks));
   g.setColor(Color.blue);
    g.drawLine(x, -(int)(PN[x] * 300 / numTicks), x + 1, -(int)(PN[x + 1] * 300 / numTicks));
}

//Line names for simulations variables
g.setColor(Color.black);
g.drawString("per mRNA", 790, -180);
g.drawString("PER Protein 0", 790, -150);
g.drawString("PER Protein 1", 790, -120);
g.drawString("PER Protein 2", 790, -90);
g.drawString("PER Protein N", 790, -60);

//Color code each line
g.setColor(Color.black);
g.drawLine(750, -185, 780, -185);
g.setColor(Color.magenta);
g.drawLine(750, -155, 780, -155);
g.setColor(Color.red);
g.drawLine(750, -125, 780, -125);
```

```
        g.setColor(Color.green);
        g.drawLine(750, -95, 780, -95);
        g.setColor(Color.blue);
        g.drawLine(750, -65, 780, -65);
    }
}
```

5. Dynamic.java

```
/**
 * File name: Dynamic.java
 *
 * Description: This applet is similar to Non_Dynamic.java except that it runs
 *              continuously (animation)
 *
 * Copyright: Copyright (c) 2006
 *
 * Company: Albuquerque Academy
 *
 * Author: Michael Wang
 *
 * Date: Dec 2006
 *
 * @version 1.0
 */

package circadian_rhythms;

import java.awt.*;
import java.util.Random;
import java.applet.*;

/**
 * This class displays the simulation variables, M, P0, P1, P2, and PN,
 * and the "noise", et.
 */
public class Dynamic extends Applet
{
    public Dynamic()
    {
        this.setSize(1000, 650);       //set the dimensions of the applet
    }

    int numLoops = 720;
    int numTicks, n, xScl;
    double M[], P0[], P1[], P2[], PN[], vs, vm, vd, ks, k1, k2, V1, V2, V3, V4;
    double K1, K2, K3, K4, Kd, Km, KI, timeStep, t, t1, t2, t3, t4, dt;
    float e[];
    Parameters edps;
    public void dynamic(double Mi[], double P0i[], double P1i[], double P2i[], double PNi[], double vs2,
                    double vm2, double vd2, double ks2, double k12, double k22, double V12, double V22,
                    double V32, double V42, double K12,  double K22, double K32, double K42, double Kd2,
                    double Km2, double KI2, int n2,  int numTicks2, double timeStep2, float e2[], Parameters edps1)
    {
        //set the size of each array
        M = new double[numLoops];
        P0 = new double[numLoops];
        P1 = new double[numLoops];
        P2 = new double[numLoops];
        PN = new double[numLoops];
        e = new float[numLoops];
    //initialize arrays
    for(int i = 0; i < numLoops; i++)
    {
        M[i] = Mi[i];
        P0[i] = P0i[i];
        P1[i] = P1i[i];
        P2[i] = P2i[i];
        PN[i] = PNi[i];
        e[i] = e2[i];
    }

    //initialize variables
    numTicks = numTicks2;
    timeStep = timeStep2;
    vs = vs2;
    vm = vm2;
    vd = vd2;
    ks = ks2;
    k1 = k12;
    k2 = k22;
    V1 = V12;
    V2 = V22;
    V3 = V32;
    V4 = V42;
    K1 = K12;
    K2 = K22;
    K3 = K32;
    K4 = K42;
    Kd = Kd2;
    Km = Km2;
    KI = KI2;
    n = n2;
    edps = edps1;
}

int shift = 0;              //used to shift graph to keep certain components in the same place
int XValShift = 0;          //used to shift x-axis values when needed
int j = numLoops;
Random r = new Random();
float et = 0;               //disturbance

public void paint(Graphics g)
{
    xScl = (int) (12 * 60.0 / timeStep); //# of pixels per 12 hrs.

    g.translate(115 - shift, 320);                  //move (0,0) to (115,320)
    g.drawLine( -20 + shift, 0, 720 + shift, 0);     //x-axis line
    g.drawLine(shift, 10, shift, -300);              //y-axis line
    g.drawString("Time Elapsed (Hours)", 350 + shift, 40);
    g.drawString("Concentration", -105 + shift, -165);
    g.drawString("(micro Molarity)", -107 + shift, -150);
    g.drawLine(-60 + shift, 65, 820 + shift, 65); //separation line for the two graphs

    //Major ticks of the y-axis
    for (int yAxis = 1; yAxis < numTicks + 1; yAxis++)
    {
        g.drawLine(shift, -yAxis * 300 / numTicks, shift + 3, -yAxis * 300 / numTicks);
        g.drawString(yAxis + "", -15 + shift, -yAxis * 300 / numTicks + 5);
    }

    //Major ticks of the x-axis.  When the total shift = an interval, add a new number.
```

```
if (shift % xScl == 0 && shift > 0)
{
   for (int xAxis = 1; xAxis < 720 / xScl + 1; xAxis++) {
      g.drawLine(xAxis * xScl, 0, xAxis * xScl, -3);
      g.drawString((xAxis + shift) * 12 + "", (xAxis + shift) * xScl - 7, 15);
   }
   XValShift++;
}
else
{
   for (int xAxis = 1; xAxis < 720 / xScl + 1; xAxis++)
   {
      g.drawLine((xAxis + XValShift) * xScl, 0, (xAxis + XValShift) * xScl, -3);
      g.drawString((xAxis + XValShift) * 12 + "", (xAxis + XValShift) * xScl - 7, 15);
   }
}

//Axis of second graph
g.drawLine(-20 + shift, 220, 720  + shift, 220);//x-axis of new graph
g.drawLine(shift, 120, shift, 270);//y-axis of new graph
g.drawString("Perturbations", 360 + shift, 100);
g.drawString("Time Elapsed (hours)", 360 + shift, 290);

//Draw the ticks of the second graph
g.drawLine(shift, 270, 3 + shift, 270);
 g.drawString("-0.5", -25 + shift, 275);
g.drawLine(shift, 170, 3 + shift, 170);
 g.drawString("0.5", -25 + shift, 175);
g.drawLine(shift, 120, 3 + shift, 120);
 g.drawString("1", -15 + shift, 125);
for(int i = 1; i < 720 / xScl + 1; i++)
{
   g.drawLine((i + XValShift) * xScl, 220, (i + XValShift) * xScl, 217);
   g.drawString((i + XValShift) * 12 + "", (i + XValShift) * xScl - 7, 235);
}


//Line names of the first graph
g.setColor(Color.black);
g.drawString("per mRNA", 790 + shift, -180);
g.drawString("PER Protein 0", 790 + shift, -150);
g.drawString("PER Protein 1", 790 + shift, -120);
g.drawString("PER Protein 2", 790 + shift, -90);
g.drawString("PER Protein N", 790 + shift, -60);

//Color code each line
g.setColor(Color.black);
g.drawLine(750 + shift, -185, 780 + shift, -185);
g.setColor(Color.magenta);
g.drawLine(750 + shift, -155, 780 + shift, -155);
g.setColor(Color.red);
g.drawLine(750 + shift, -125, 780 + shift, -125);
g.setColor(Color.green);
g.drawLine(750 + shift, -95, 780 + shift, -95);
g.setColor(Color.blue);
g.drawLine(750 + shift, -65, 780 + shift, -65);

repaint();

//Graph points (x,M[x]), (x,P0[x]), (x,P1[x]), (x,P2[x]), (x,PN[x]), and (x,e[x])
for(int x = 0; x < numLoops - 1; x++)
{
```

```
          //Draw line from the point at x to the point at x - 1.  Scaling factor: 300 / numticks
          g.setColor(Color.black);
           g.drawLine(719 - x + shift, -(int)(M[719 - x] * 300 / numTicks), 718 - x + shift, -(int)(M[718 - x] * 300 /
numTicks));
           g.drawLine(719 - x + shift, -(int)(e[719 - x] * 100) + 220, 718 - x + shift, -(int)(e[718 - x] * 100) + 220);
          g.setColor(Color.magenta);
           g.drawLine(719 - x + shift, -(int)(P0[719 - x] * 300 / numTicks), 718 - x + shift, -(int)(P0[718 - x] * 300 /
numTicks));
          g.setColor(Color.red);
           g.drawLine(719 - x + shift, -(int)(P1[719 - x] * 300 / numTicks), 718 - x + shift, -(int)(P1[718 - x] * 300 /
numTicks));
          g.setColor(Color.green);
           g.drawLine(719 - x + shift, -(int)(P2[719 - x] * 300 / numTicks), 718 - x + shift, -(int)(P2[718 - x] * 300 /
numTicks));
          g.setColor(Color.blue);
           g.drawLine(719 - x + shift, -(int)(PN[719 - x] * 300 / numTicks), 718 - x + shift, -(int)(PN[718 - x] * 300 /
numTicks));
        }

       for(int lag = 0; lag < 4000000; lag++) {}//delay


//---------------------------Calculate next value-----------------------------//
       //If 1, add a new disturbance every disturbance interval
       if (edps.multi_single == 1)
       {
          j++;
          if (j % (int)(edps.durationT / timeStep) == 0)
             et = r.nextFloat() - 0.5f;
          e[numLoops - 1] = et;
       }

       dt = timeStep / 60.0;

       t = (V1 + edps.EDV1 * et) * P0[numLoops - 2] / ((K1 + edps.EDK1 * et) + P0[numLoops - 2]);
       t1 = (V2 + edps.EDV2 * et) * P1[numLoops - 2] / ((K2 + edps.EDK2 * et) + P1[numLoops - 2]);
       t2 = (V3 + edps.EDV3 * et) * P1[numLoops - 2] / ((K3 + edps.EDK3 * et) + P1[numLoops - 2]);
       t3 = (V4 + edps.EDV4 * et) * P2[numLoops - 2] / ((K4 + edps.EDK4 * et) + P2[numLoops - 2]);
       t4 = (k1 + edps.EDk1 * et) * P2[numLoops - 2] - (k2 + edps.EDk2 * et) * PN[numLoops - 2];

       M[numLoops - 1] = ((vs + edps.EDvs * et) * Math.pow(KI + edps.EDKI * et, n) /
                  (Math.pow(KI + edps.EDKI * et, n) + Math.pow(PN[numLoops - 2], n))
                  - (vm + edps.EDvm * et) * M[numLoops - 2] / ((Km + edps.EDKm * et) + M[numLoops - 2]))
                 * dt + M[numLoops - 2];
       P0[numLoops - 1] = ((ks + edps.EDks * et) * M[numLoops - 2] - t + t1) * dt + P0[numLoops - 2];
       P1[numLoops - 1] = (t - t1 - t2 + t3) * dt + P1[numLoops - 2];
       P2[numLoops - 1] = (t2 - t3 - t4 - (vd + edps.EDvd * et) * P2[numLoops - 2]
                   / ((Kd + edps.EDKd * et) + P2[numLoops - 2])) * dt + P2[numLoops - 2];
       PN[numLoops - 1] = t4 * dt + PN[numLoops - 2];
//------------------------------End calculations------------------------------//


       //Replace each value with the next value
       for(int i = 0; i < numLoops - 1; i++)
       {
          M[i] = M[i + 1];
          P0[i] = P0[i + 1];
          P1[i] = P1[i + 1];
          P2[i] = P2[i + 1];
          PN[i] = PN[i + 1];
          e[i] = e[i + 1];
       }
```

```
        shift++;//Increase shift by one
    }
}
```

6. Parameters.java

```
/**
 * File name: Parameters.java
 *
 * Description: This class stores the disturbance amplitudes for each parameter
 *
 * Copyright: Copyright (c) 2006
 *
 * Company: Albuquerque Academy
 *
 * Author: Michael Wang
 *
 * Date: Dec 2006
 *
 * @version 1.0
 */

package circadian_rhythms;

public class Parameters
{
    public double EDvs, EDvm, EDvd, EDks, EDk1, EDk2, EDV1, EDV2, EDV3, EDV4;
    public double EDK1, EDK2, EDK3, EDK4, EDKd, EDKm, EDKI;
    public int durationT, multi_single;
    public Parameters()
    {
        //Initial values
        EDvs = 0.0;
        EDvm = 0.0;
        EDvd = 0.0;
        EDks = 0.0;
        EDk1 = 0.0;
        EDk2 = 0.0;
        EDV1 = 0.0;
        EDV2 = 0.0;
        EDV3 = 0.0;
        EDV4 = 0.0;
        EDK1 = 0.0;
        EDK2 = 0.0;
        EDK3 = 0.0;
        EDK4 = 0.0;
        EDKd = 0.0;
        EDKm = 0.0;
        EDKI = 0.0;
        durationT = 24;
        multi_single = 0;
    }
}
```