Uppsala University

Scientific Computing for Data Analysis

Mini-Project Group Kand.02

Exploring the Reproducibility of Circadian Clock Models in Systems Biology

Authors:
Hamit Efe Çınar, Mehmet Fırat Dündar

April 22, 2024

1 Introduction

In this mini-project, we're diving into a study about biological clocks—specifically, the circadian clock. Think of it as an internal clock in living organisms that helps them adapt to daily changes, like the cycle of day and night. Our guide for this exploration is a research article that looks at how these clocks work at a genetic level.

The key thing we're focusing on is how these clocks stay accurate and stable, even when things around them change. The researchers in the article developed two models, one deterministic (predictable) and another stochastic (random), to study this. They're based on two proteins: one that starts processes (activator) and another that stops them (repressor). The interaction between these proteins is what keeps the clock ticking regularly.

Our project is all about trying to replicate parts of this study to see if we can get similar results. We're not expected to understand every tiny detail of the biology involved, but we should grasp enough to carry out our mini-project. By doing this, we'll learn not just about circadian clocks, but also about the importance of being able to repeat experiments in science.

2 Approach and Method

2.1 Overview

In this project, our goal is to simulate and analyze a model of the circadian oscillator—a biological clock mechanism in organisms. We use computational methods to replicate the dynamics of this system, focusing on the interactions between key proteins involved in the process.

2.2 Models and Computational Methods

We employed two primary models in our simulations:

2.2.1 Ordinary Differential Equations (ODEs)

ODEs are used to describe the continuous, deterministic behavior of the system. In the context of the circadian oscillator, ODEs help us understand how the concentrations of various molecular species (like proteins and mRNA) evolve over time under fixed reaction rates.

2.2.2 Stochastic Simulation Algorithm (SSA)

The Gillespie algorithm, a type of SSA, is used to model the stochastic nature of biochemical reactions at the molecular level. This algorithm is particularly useful for capturing the random fluctuations and noise inherent in biological systems, which are significant especially when dealing with small numbers of molecules.

2.3 Mathematical Formulation

For simplicity, we took account of a smaller system. We consider a state vector $\mathbf{y} = [y_1, y_2] = [F, R]$, representing the concentration of activator (F) and repressor (R) proteins. The propensity functions and state-change vectors are given as follows:

$$\begin{array}{llll} 1. & R \xrightarrow{\alpha} 2R & v_1 = [0,1], & w_1 = \alpha y_2 \\ 2. & R + F \xrightarrow{\beta} 2F & v_2 = [1,-1], & w_2 = \beta y_1 y_2 \\ 3. & F \xrightarrow{\gamma} \emptyset & v_3 = [-1,0], & w_3 = \gamma y_1 \end{array}$$

We define the total propensity as $a = w_1 + w_2 + w_3$, and the probability of each reaction occurring as $p_j = \frac{w_j}{a}$ for $j = 1,2,3$. The state-change matrix is:

$$S = \begin{bmatrix} 0 & 1 \\ 1 & -1 \\ -1 & 0 \end{bmatrix}.$$

The corresponding ODEs for the system are:

$$\frac{dF}{dt} = \beta FR - \gamma F$$
$$\frac{dR}{dt} = \alpha R - \beta FR$$

These equations represent the rate of change of the activator and repressor protein concentrations over time, considering their interactions.

2.4 Justification of Methods

2.4.1 ODEs

This approach is suitable for understanding the general behavior of the system under idealized, noise-free conditions. It helps in forming a baseline understanding of the system dynamics.

2.4.2 SSA/Gillespie Algorithm

This method is crucial for capturing the inherent randomness in biochemical systems, especially important for low molecule numbers where stochastic effects dominate.

2.5 Implementation

In the implementation, we utilized ‘gillespy2‘, a Python package designed for simulating biochemical networks. This allowed us to set up the circadian oscillator model with defined species, reactions, and parameters. The simulation results from both ODE and SSA methods were then visualized using ‘matplotlib‘ to compare and analyze the deterministic and stochastic behaviors of the system.

The code snippet provided (See Appendix A for the whole code.) in the report outlines the setup of the model, indicating the species, parameters, and reactions involved. We used ‘numpy‘ for numerical operations and ‘matplotlib‘ for plotting the results, giving us a clear visual representation of the system’s dynamics over time.

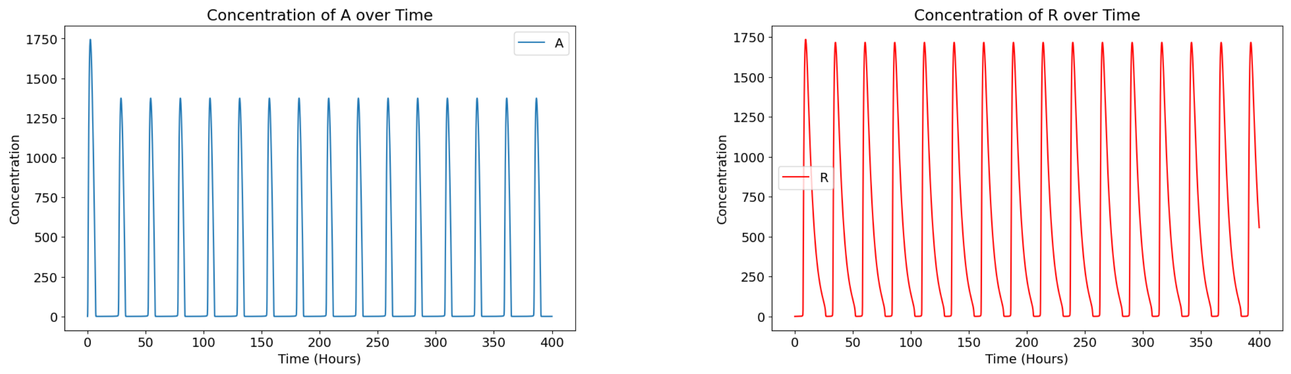
3 Results

3.1 Deterministic Model Reproduction

For the deterministic model reproduction, we used the Ordinary Differential Equations (ODEs) approach to simulate the circadian rhythm over 400 hours. The results were plotted using 'matplotlib.pyplot', focusing on the concentrations of the activator protein A and repressor protein R, as described in the model from the article.

3.1.1 Figure 2a and 2b Reproduction

The ODE model from Eq.[1] of the article was solved, and the results are shown in Figure 1 below. The plots demonstrate how the concentrations of proteins A and R evolve over time, displaying a regular, oscillatory pattern, which is a characteristic feature of the circadian rhythm. These results are consistent with the original figures in the article, demonstrating the reliability of the ODE model in capturing the deterministic nature of the circadian oscillator.



(a) Concentration of A over time (ODE)   (b) Concentration of R over time (ODE)

Figure 1: Concentration over time (ODE)

3.2 Stochastic Model Reproduction

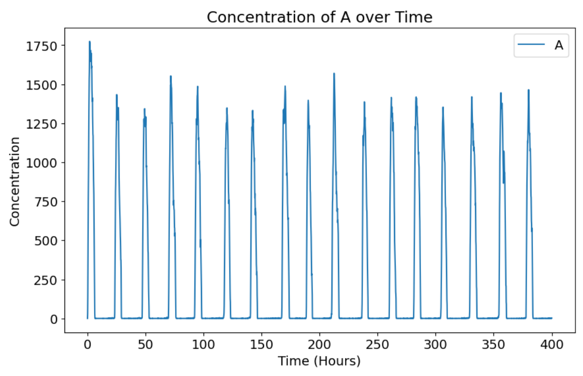
For the stochastic model, the Gillespie algorithm (Stochastic Simulation Algorithm - SSA) was employed to simulate the system over 400 hours. We used the same parameter values as in the deterministic model to maintain consistency.

3.2.1 Figure 2c and 2d Reproduction

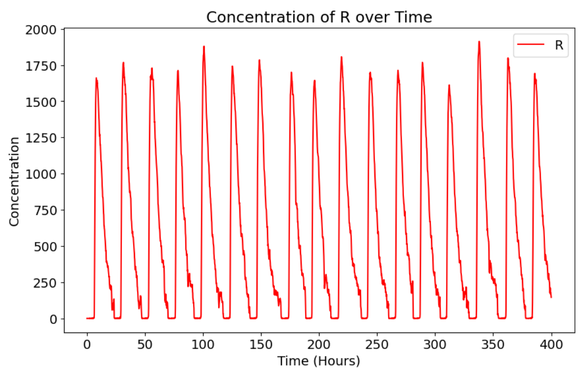
The results from the SSA simulations for proteins A and R are shown in Figure 2 below. Unlike the deterministic model, the stochastic model exhibits variability in the oscillation patterns, reflecting the intrinsic noise present in biological systems. Running the simulation multiple times produced different outcomes each time, highlighting the stochastic nature of the process. These results align with the notion that stochastic models can reveal different behavioral aspects of biological systems compared to deterministic models.

3.2.2 Comparative Analysis

The deterministic and stochastic models yielded similar patterns in terms of the oscillatory nature of the circadian rhythm. However, the stochastic model displayed variability in each simulation run, contrasting with the predictability of the deterministic model. This aligns



(a) Concentration of A over time (SSA)



(b) Concentration of R over time (SSA)

Figure 2: Concentration over time (SSA)

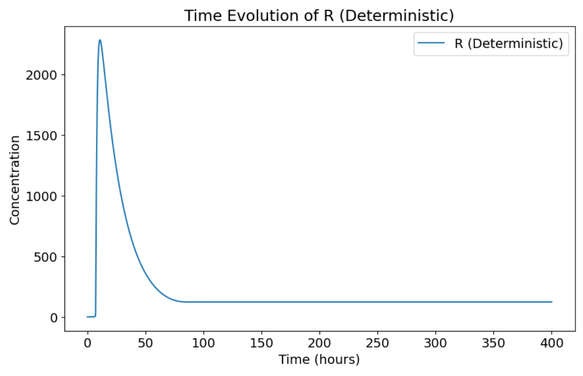
with the findings in the article, where random white noise in the stochastic model introduces variability not observed in the deterministic approach.

3.3 Reproduction and Analysis of Figure 5 in the Article

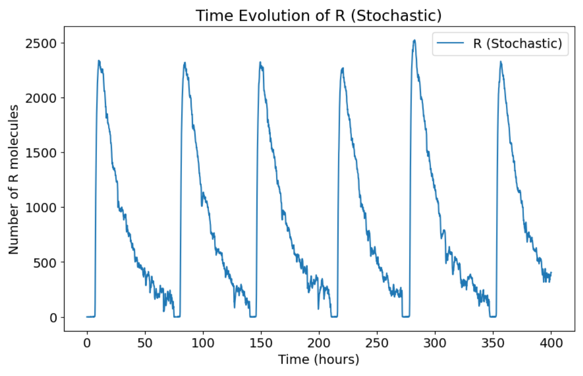
The task here is to reproduce the result shown in Figure 5 of the article, where the presence of noise in the stochastic model led to qualitative differences compared to the deterministic model.

3.3.1 Deterministic vs Stochastic Model in Special Case

In this scenario, the stochastic model revealed a behavior not predicted by the deterministic model. The stochastic simulations showed sustained oscillations under conditions where the deterministic model predicted a stable state. This phenomenon illustrates how noise can not only add variability but also qualitatively change the behavior of the system, a key insight provided by the article. These special cases are successfully reproduces as can be seen in Figure 3 provided.



(a) Time evolution of R for the deterministic model



(b) Time evolution of R for the stochastic model

Figure 3: Time evolution of R with different models

4 Discussion

The choice between deterministic and stochastic models in biological systems, particularly in the context of the circadian clock, is a nuanced one, reflecting the complexities of biological processes at different scales.

4.1 Deterministic Models

Deterministic models, like the one described by the set of reaction rate equations in the article by Vilar et al., are typically valid in well-controlled, macroscopic systems where the number of interacting molecules is large, and random fluctuations are negligible. These models are useful for understanding the general behavior and long-term trends of a system. They provide a clear and predictable picture of how the system should behave under idealized conditions, making them excellent tools for initial hypothesis testing and theory development.

In the context of circadian rhythms, deterministic models help to establish a baseline understanding of the oscillatory mechanisms, as they capture the average behavior of the system over time. For instance, in well-mixed environments where molecular species are present in large numbers, deterministic models can accurately predict the behavior of the system.

4.2 Stochastic Models

On the other hand, at the cellular level, where the number of molecules can be low and the biochemical reactions exhibit intrinsic stochasticity, stochastic models become more relevant. In these models, randomness is an inherent part of the system's behavior. The Gillespie algorithm, as mentioned in the article, is a key method for simulating such stochastic behavior, as it considers the probabilistic nature of chemical reactions.

Stochastic models are essential when studying systems where noise plays a significant role. As the article points out, circadian clocks, despite being subject to fluctuations in molecular concentrations and environmental conditions, maintain a relatively constant period. This robustness against noise can be better understood through stochastic models. In certain conditions, as shown in the article, the stochastic nature can even enhance the oscillator's performance, a phenomenon not captured by deterministic models.

4.3 Comparative Insights from the Circadian Clock Study

The circadian clock study by Vilar et al. provides valuable insights into how these two types of models can yield different understandings of the same biological system. While deterministic models capture the idealized behavior of the circadian oscillator, stochastic models reveal how the system might behave under the more variable and less predictable conditions found in actual biological cells. The presence of noise can lead to qualitative differences in behavior, as demonstrated in the article where certain conditions show sustained oscillations in the stochastic model but not in the deterministic model. [1]

5 Conclusion and Outlook

In conclusion, in this study, we explored the modeling of circadian rhythms through both deterministic and stochastic approaches, drawing on the seminal work by Vilar et al. Our simulations successfully reproduced the key results presented in the article, demonstrating the oscillatory behaviors of activator and repressor proteins in a circadian clock system. The deterministic model provided a clear and predictable pattern of these oscillations, illustrating an idealized view of the system under controlled conditions. In contrast, the stochastic model, realized through the Gillespie algorithm, introduced the element of randomness, reflecting the inherent noise present in cellular-level biochemical reactions.

The comparative analysis between these two modeling approaches highlighted the significance of context in choosing the appropriate modeling framework. While deterministic models excel in macroscopic systems with large molecule numbers, stochastic models are indispensable for understanding systems where noise and random fluctuations play a crucial role.

What we found especially compelling is the way our comparative analysis illuminated the importance of context in model selection. This study serves as a powerful reminder that the choice between deterministic and stochastic models is not just a technical decision but a crucial strategic consideration, deeply connected to the nature of the system being studied.

Future extensions of this study could explore more intricate biological interactions for enhanced model accuracy, couple computational models with experimental data for validation, and conduct sensitivity analyses to identify key influencing factors. Additionally, integrating multi-scale modeling approaches and machine learning techniques could offer deeper insights into biological rhythms. These advancements have potential applications in medical research, particularly in understanding and treating health conditions linked to circadian rhythm disruptions.

References

[1] Vilar et al. “Mechanisms of Noise-Resistance in Genetic Oscillators”. In: *Proceedings of the National Academy of Sciences* (2002).

6 Appendix

A The Whole Code

```
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png','pdf')
import numpy as np
import gillespy2
import matplotlib.pyplot as plt
from gillespy2 import Model, Species, Reaction, Parameter, RateRule, AssignmentRule, FunctionDefinition
from gillespy2 import EventAssignment, EventTrigger, Event, ODESolver


class Circadian_Oscillator(Model):
    def __init__(self, parameter_values=None):
        Model.__init__(self, name="Circadian_Oscillator")
        self.volume = 1

        # Define the parameters of the system
        self.add_parameter(Parameter(name="alpha", expression=50)) # Basal rate of transcription for A
        self.add_parameter(Parameter(name="alpha_prime", expression=500)) # Activated rate of transcription for A
        self.add_parameter(Parameter(name="alpha_R", expression=0.01)) # Basal rate of transcription for R
        self.add_parameter(Parameter(name="alpha_R_prime", expression=50)) # Activated rate of transcription for R
        self.add_parameter(Parameter(name="beta_A", expression=50)) # Translation rate for A
        self.add_parameter(Parameter(name="beta_R", expression=5)) # Translation rate for R
        self.add_parameter(Parameter(name="delta_MA", expression=10)) # Degradation rate of mRNA A
        self.add_parameter(Parameter(name="delta_MR", expression=0.5)) # Degradation rate of mRNA R
        self.add_parameter(Parameter(name="delta_A", expression=1)) # Degradation rate of A
        self.add_parameter(Parameter(name="delta_R", expression=0.2)) # Degradation rate of R
        self.add_parameter(Parameter(name="gamma_A", expression=1)) # Binding rate of A to other components
        self.add_parameter(Parameter(name="gamma_R", expression=1)) # Binding rate of R to other components
        self.add_parameter(Parameter(name="gamma_C", expression=2)) # Unbinding rate of A from other components
        self.add_parameter(Parameter(name="theta_A", expression=50))
        self.add_parameter(Parameter(name="theta_R", expression=100))

        # Add species and define the initial condition (count or concentration) of each species
        self.add_species(Species(name="DA", initial_value=1, mode="discrete")) # Number of activator genes with A bound to its promoter
        self.add_species(Species(name="DA_prime", initial_value=0, mode="discrete")) # Number of activator genes without A bound to its promoter
        self.add_species(Species(name="DR", initial_value=1, mode="discrete")) # Number of repressor genes with R bound to its promoter
        self.add_species(Species(name="DR_prime", initial_value=0, mode="discrete")) # Number of repressor genes without R bound to its promoter
        self.add_species(Species(name="MA", initial_value=0, mode="discrete")) # mRNA of A
        self.add_species(Species(name="MR", initial_value=0, mode="discrete")) # mRNA of R
        self.add_species(Species(name="A", initial_value=0, mode="discrete")) # Activator protein
        self.add_species(Species(name="R", initial_value=0, mode="discrete")) # Repressor protein
        self.add_species(Species(name="C", initial_value=0, mode="discrete")) # Inactivated complex formed by A and R

        # Reactions
        self.add_reaction(Reaction(name="r1", reactants={'A': 1, 'R': 1}, products={'C': 1}, rate=self.listOfParameters["gamma_C"]))
        self.add_reaction(Reaction(name="r2", reactants={'A': 1}, products={}, rate=self.listOfParameters["delta_A"]))
        self.add_reaction(Reaction(name="r3", reactants={'C': 1}, products={'R': 1}, rate=self.listOfParameters["delta_A"]))
        self.add_reaction(Reaction(name="r4", reactants={'R': 1}, products={}, rate=self.listOfParameters["delta_R"]))
        self.add_reaction(Reaction(name="r5", reactants={'DA': 1, 'A': 1}, products={'DA_prime': 1}, rate=self.listOfParameters["gamma_A"]))
        self.add_reaction(Reaction(name="r6", reactants={'DR': 1, 'A': 1}, products={'DR_prime': 1}, rate=self.listOfParameters["gamma_R"]))
        self.add_reaction(Reaction(name="r7", reactants={'DA_prime': 1}, products={'DA': 1, 'A':1}, rate=self.listOfParameters["theta_A"]))
        self.add_reaction(Reaction(name="r8", reactants={'DA': 1}, products={'DA': 1, 'MA':1}, rate=self.listOfParameters["alpha"]))
        self.add_reaction(Reaction(name="r9", reactants={'DA_prime': 1}, products={'DA_prime': 1, 'MA':1}, rate=self.listOfParameters["alpha_prime"]))
        self.add_reaction(Reaction(name="r10", reactants={'MA': 1}, products={}, rate=self.listOfParameters["delta_MA"]))
        self.add_reaction(Reaction(name="r11", reactants={'MA': 1}, products={'A': 1, 'MA': 1}, rate=self.listOfParameters["beta_A"]))
        self.add_reaction(Reaction(name="r12", reactants={'DR_prime': 1}, products={'DR': 1, "A":1}, rate=self.listOfParameters["theta_R"]))
        self.add_reaction(Reaction(name="r13", reactants={'DR': 1}, products={'DR': 1, 'MR':1}, rate=self.listOfParameters["alpha_R"]))
        self.add_reaction(Reaction(name="r14", reactants={'DR_prime': 1}, products={'DR_prime': 1, 'MR':1}, rate=self.listOfParameters["alpha_R_prime"]))
        self.add_reaction(Reaction(name="r15", reactants={'MR': 1}, products={}, rate=self.listOfParameters["delta_MR"]))
        self.add_reaction(Reaction(name="r16", reactants={'MR': 1}, products={'R': 1, 'MR': 1}, rate=self.listOfParameters["beta_R"]))

        # Timespan
        self.timespan(np.linspace(0, 400, 3000))


model = Circadian_Oscillator()
model.timespan(np.linspace(0, 400, 3000)) # Set appropriate timespan
# Run the simulation using the ODE solver
# Choose and set up the ODE solver
results = model.run(algorithm="ODE")

plt.figure(figsize=(10, 6))
plt.plot(results['time'], results['A'], label='A')
plt.xlabel('Time')
plt.ylabel('Concentration')
plt.title('Concentration of A over Time')
plt.legend()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(results['time'], results['R'], label='R', color='red')
plt.xlabel('Time')
plt.ylabel('Concentration')
plt.title('Concentration of R over Time')
plt.legend()
plt.show()

model = Circadian_Oscillator()
results = model.run(algorithm = "SSA")

plt.figure(figsize=(10, 6))
plt.plot(results['time'], results['A'], label='A')
plt.xlabel('Time')
plt.ylabel('Concentration')
plt.title('Concentration of A over Time')
plt.legend()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(results['time'], results['R'], label='R', color='red')
plt.xlabel('Time')
plt.ylabel('Concentration')
plt.title('Concentration of R over Time')
plt.legend()
plt.show()


class Circadian_Oscillator(Model):
    def __init__(self, parameter_values=None):
        Model.__init__(self, name="Circadian_Oscillator")
        self.volume = 1

        # Define the parameters of the system
        self.add_parameter(Parameter(name="alpha", expression=50)) # Basal rate of transcription for A
        self.add_parameter(Parameter(name="alpha_prime", expression=500)) # Activated rate of transcription for A
        self.add_parameter(Parameter(name="alpha_R", expression=0.01)) # Basal rate of transcription for R
        self.add_parameter(Parameter(name="alpha_R_prime", expression=50)) # Activated rate of transcription for R
        self.add_parameter(Parameter(name="beta_A", expression=50)) # Translation rate for A
        self.add_parameter(Parameter(name="beta_R", expression=5)) # Translation rate for R
        self.add_parameter(Parameter(name="delta_MA", expression=10)) # Degradation rate of mRNA A
        self.add_parameter(Parameter(name="delta_MR", expression=0.5)) # Degradation rate of mRNA R
        self.add_parameter(Parameter(name="delta_A", expression=1)) # Degradation rate of A
        self.add_parameter(Parameter(name="delta_R", expression=0.05)) # Degradation rate of R
        self.add_parameter(Parameter(name="gamma_A", expression=1)) # Binding rate of A to other components
        self.add_parameter(Parameter(name="gamma_R", expression=1)) # Binding rate of R to other components
        self.add_parameter(Parameter(name="gamma_C", expression=2)) # Unbinding rate of A from other components
        self.add_parameter(Parameter(name="theta_A", expression=50))
        self.add_parameter(Parameter(name="theta_R", expression=100))

        # Add species and define the initial condition (count or concentration) of each species
        self.add_species(Species(name="DA", initial_value=1, mode="discrete")) # Number of activator genes with A bound to its promoter
        self.add_species(Species(name="DA_prime", initial_value=0, mode="discrete")) # Number of activator genes without A bound to its promoter
        self.add_species(Species(name="DR", initial_value=1, mode="discrete")) # Number of repressor genes with R bound to its promoter
        self.add_species(Species(name="DR_prime", initial_value=0, mode="discrete")) # Number of repressor genes without R bound to its promoter
        self.add_species(Species(name="MA", initial_value=0, mode="discrete")) # mRNA of A
        self.add_species(Species(name="MR", initial_value=0, mode="discrete")) # mRNA of R
        self.add_species(Species(name="A", initial_value=0, mode="discrete")) # Activator protein
        self.add_species(Species(name="R", initial_value=0, mode="discrete")) # Repressor protein
        self.add_species(Species(name="C", initial_value=0, mode="discrete")) # Inactivated complex formed by A and R

        # Reactions
        self.add_reaction(Reaction(name="r1", reactants={'A': 1, 'R': 1}, products={'C': 1}, rate=self.listOfParameters["gamma_C"]))
        self.add_reaction(Reaction(name="r2", reactants={'A': 1}, products={}, rate=self.listOfParameters["delta_A"]))
        self.add_reaction(Reaction(name="r3", reactants={'C': 1}, products={'R': 1}, rate=self.listOfParameters["delta_A"]))
        self.add_reaction(Reaction(name="r4", reactants={'R': 1}, products={}, rate=self.listOfParameters["delta_R"]))
        self.add_reaction(Reaction(name="r5", reactants={'DA': 1, 'A': 1}, products={'DA_prime': 1}, rate=self.listOfParameters["gamma_A"]))
        self.add_reaction(Reaction(name="r6", reactants={'DR': 1, 'A': 1}, products={'DR_prime': 1}, rate=self.listOfParameters["gamma_R"]))
        self.add_reaction(Reaction(name="r7", reactants={'DA_prime': 1}, products={'DA': 1, 'A':1}, rate=self.listOfParameters["theta_A"]))
        self.add_reaction(Reaction(name="r8", reactants={'DA': 1}, products={'DA': 1, 'MA':1}, rate=self.listOfParameters["alpha"]))
        self.add_reaction(Reaction(name="r9", reactants={'DA_prime': 1}, products={'DA_prime': 1, 'MA':1}, rate=self.listOfParameters["alpha_prime"]))
        self.add_reaction(Reaction(name="r10", reactants={'MA': 1}, products={}, rate=self.listOfParameters["delta_MA"]))
        self.add_reaction(Reaction(name="r11", reactants={'MA': 1}, products={'A': 1, 'MA': 1}, rate=self.listOfParameters["beta_A"]))
        self.add_reaction(Reaction(name="r12", reactants={'DR_prime': 1}, products={'DR': 1, "A":1}, rate=self.listOfParameters["theta_R"]))
        self.add_reaction(Reaction(name="r13", reactants={'DR': 1}, products={'DR': 1, 'MR':1}, rate=self.listOfParameters["alpha_R"]))
        self.add_reaction(Reaction(name="r14", reactants={'DR_prime': 1}, products={'DR_prime': 1, 'MR':1}, rate=self.listOfParameters["alpha_R_prime"]))
        self.add_reaction(Reaction(name="r15", reactants={'MR': 1}, products={}, rate=self.listOfParameters["delta_MR"]))
        self.add_reaction(Reaction(name="r16", reactants={'MR': 1}, products={'R': 1, 'MR': 1}, rate=self.listOfParameters["beta_R"]))

        # Timespan
        self.timespan(np.linspace(0, 400, 3000))


 # Create an instance of the deterministic model
 model_deterministic = Circadian_Oscillator()

 # Choose and set up the ODE solver
 solver_deterministic = ODESolver(model=model_deterministic)

 # Run the deterministic simulation
 results_deterministic = solver_deterministic.run()

 # Extract simulation data
 timepoints_det = results_deterministic['time']
 R_values_det = results_deterministic['R']

 # Plot the deterministic simulation results
 plt.figure(figsize=(10, 6))
 plt.plot(timepoints_det, R_values_det, label='R (Deterministic)')
 plt.xlabel('Time (hours)')
 plt.ylabel('Concentration')
 plt.title('Time Evolution of R (Deterministic)')
 plt.legend()
 plt.show()

 # Create an instance of the stochastic model
 model_stochastic = Circadian_Oscillator()

 # Run the stochastic simulation
 results_stochastic = model_stochastic.run(algorithm = "SSA")

 # Extract simulation data
 timepoints_stoch = results_stochastic['time']
 R_values_stoch = results_stochastic['R']

 # Plot the stochastic simulation results
 plt.figure(figsize=(10, 6))
 plt.plot(timepoints_stoch, R_values_stoch, label='R (Stochastic)')
 plt.xlabel('Time (hours)')
 plt.ylabel('Number of R molecules')
 plt.title('Time Evolution of R (Stochastic)')
 plt.legend()
 plt.show()
```