Dynamical systems in ecology and transcription*

Joshua Weinstein *University of Chicago*

Introduction to dynamical systems in biology

Dynamical systems are mathematical models that describe how a system changes over time. In biology, they are crucial for understanding complex processes like gene regulation, population dynamics, and cellular metabolism. A key concept in dynamical systems is the steady state, where the system's variables remain constant over time. Mathematically, we can express this as the derivative

$$\frac{dx}{dt} = 0$$

where x represents the system's variables and t is time. d/dt is simply the time-derivative of x, meaning the rate at which x, whatever it is, changes over time.

Broadly, although steady-state itself is not a particularly interesting area of study in biological dynamics, it is useful to analyze a biological system "in the neighborhood of" steady-states. Take for instance the simple derivative that describes logistic growth of a bacterial culture:

$$\frac{dx}{dt} = rx\left(1 - \frac{x}{K}\right)$$

which describes growth of some species at rate r in a system with finite carrying capacity K. By just looking at this equation, we can see that there are two values of x for which $dx/dt = 0$.

First, is when $x = 0$, such that there is no member of the species to replicate/grow at all. The second is where $x = K$, at which point the carrying capacity has been reached exactly. Neither of these describes quantitative dynamics. However when x *approaches* 0

$$\frac{dx}{dt} \approx rx$$

and when x *approaches* K

$$\frac{dx}{dt} \approx rK(1 - x/K)$$

meaning that the time derivative of x appears to have a linear relationship to x, meaning that we can approximate its behavior by exponential growth/decay. Because of this, linearity — and linear algebra — plays a crucial role in analyzing these systems. We will see this concretely in the next section.

The Lotka-Volterra model

The Lotka-Volterra model, also known as the predator-prey equations, is a pair of first-order non-linear differential equations frequently used to describe the dynamics of biological systems in

*This document is included as part of the Dynamic Systems workshop packet for the BSD qBio Bootcamp, University of Chicago, 2024. **Current version**: August 16, 2024; **Corresponding author**: Joshua Weinstein, jaweinst@uchicago.edu.

which two species interact, one as a predator and the other as prey. While the model is simple, assuming constant rates of predation and reproduction, it allows us to understand how species interact, and how these interactions can lead to complex dynamics. These dynamics have implications well beyond ecology.

In its simplest form, the Lotka-Volterra model is described by the following system of equations:

$$\frac{dx}{dt} = ax - bxy$$
$$\frac{dy}{dt} = -cy + dxy$$

Here, $x(t)$ represents the prey population and $y(t)$ represents the predator population at time t. The parameters a, b, c, and d are positive real numbers that describe the interaction of the two species:

- a: the growth rate of the prey in the absence of predators
- b: the rate at which predators eat prey
- c: the death rate of predators in the absence of prey
- d: the rate at which predators increase by consuming prey

Note that some of the terms in the equations are "first-order", meaning that they contribute a linear term to the time-derivative of the predator or prey, whereas some are "second-order", meaning that the *contribution* of one population explicitly depends on the population of the other population.

The two first-order terms, ax and $-cy$, represent the natural growth of the prey population in the absence of predators and the natural death rate of predators in the absence of prey, respectively. For the two second-order terms, $-bxy$ and dxy, represent the rate at which prey are eaten by predators and the growth of the predator population due to consuming prey, respectively.

The behavior of this system can vary dramatically depending on the values of the parameters and initial conditions. One of the most interesting features of the Lotka-Volterra model is its ability to produce oscillatory behavior under certain conditions. Specifically, when all parameters are positive and non-zero, the system will generally exhibit periodic solutions. These solutions trace closed orbits in the phase plane, representing cyclical fluctuations in both predator and prey populations.

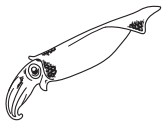
The oscillatory nature of the system can be understood intuitively: as the prey population increases, it provides more food for predators, leading to an increase in the predator population. The increased predator population then consumes more prey, causing the prey population to decrease. With less prey available, the predator population begins to decline, which then allows the prey population to recover, and the cycle continues.

However, not all parameterizations lead to oscillatory behavior. If we set $a = 0$ and $c = 0$, for instance, we get a system where both populations will exponentially decay to zero, regardless of the initial conditions. This represents a scenario where prey have no natural growth and predators have no natural death rate, leading to mutual extinction.

On the other hand, if we set $b = 0$ and $d = 0$, we decouple the equations, resulting in exponential growth for the prey population and exponential decay for the predator population. This could represent a situation where predators are unable to catch prey, leading to unchecked prey growth

and predator extinction.

In the following we'll walk through a simulated example of how these dynamics can work.



D. pealeii

Some setup

If you have not already done so, install these packages:

```
install.packages("deSolve")
install.packages("ggplot2")
install.packages("cowplot")
install.packages("plotly")
```

Let's now load the R packages we will need for this analysis:

```
library(deSolve)
library(ggplot2)
library(cowplot)
library(plotly)
```

Note if you are running this in a Jupyter notebook or in Google Colab, I recommend setting this option using the "repr" R package:

```
install.packages("repr")
library(repr)
options(jupyter.rich_display = FALSE)
```

Also for Jupyter notebook or Google Colab, you will need pandoc for the 3-d plot:

```
install.packages("pandoc")
library(pandoc)
pandoc_install()
pandoc_activate()
```

A small simulation in R

We can now define a function that will return the time-derivatives of the predator. This function defines the Lotka-Volterra equations. It takes the current state (prey and predator populations) and parameters as inputs, and returns the rate of change for both populations.

```
lotka_volterra <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dxdt <- a*x - b*x*y
    dydt <- -c*y + d*x*y
    return(list(c(dxdt, dydt)))
  })
}
```

Here we set the model parameters a, b, c, d, create a time array from 0 to 100 with 1,000 points, and set the maximum values for plotting. We also set the initial populations for prey and predator, combine them into an initial state vector, and define a small "epsilon" value to avoid division by zero later.

```
a <- 1; b <- 0.5; c <- 0.75; d <- 0.25
parameters <- c(a = a, b = b, c = c, d = d)
t <- seq(0, 100, length.out = 1000)
max_x <- 8; max_y <- 6
state <- c(x = 3, y = 4)
epsilon <- 1e-10
```

Now we use deSolve's ode interface to solve the Lotka-Volterra equations over the specified time range, starting from the initial conditions. The solution is then unpacked into separate arrays for prey (x) and predator (y) populations.

```
solution <- ode(y = state, times = t, func = lotka_volterra,
                parms = parameters)
solution <- as.data.frame(solution)
x <- solution[, "x"]
y <- solution[, "y"]
head(solution)
tail(solution)
```

These next lines of code create a grid for the vector field, calculate the direction of change at each point and normalize the vectors.

```
create_vector_field <- function(X, Y) {
  dxdt <- a*X - b*X*Y
  dydt <- -c*Y + d*X*Y
  list(dxdt = dxdt, dydt = dydt)
}
X <- seq(0, max_x, length.out = 20)
Y <- seq(0, max_y, length.out = 20)
grid <- expand.grid(X = X, Y = Y)
vector_field <- create_vector_field(grid$X, grid$Y)
arrow_scale <- 0.2  # Adjust this value to change arrow length
norm <- sqrt(vector_field$dxdt^2 + vector_field$dydt^2) + epsilon
vector_field$u <- arrow_scale * vector_field$dxdt / norm
vector_field$v <- arrow_scale * vector_field$dydt / norm
```

Finally, we create the plot, showing both the vector field and the solution trajectory.

```
p <- ggplot() +
  geom_segment(data = grid, aes(x = X, y = Y,
                                xend = X + vector_field$u,
                                yend = Y + vector_field$v),
               arrow = arrow(length = unit(0.2, "cm")),
               color = "gray", alpha = 0.3) +
  geom_path(data = solution, aes(x = x, y = y), color = "blue") +
  labs(x = "Prey", y = "Predator", title = "Lotka-Volterra Dynamics") +
  coord_cartesian(xlim = c(0, max_x),
                  ylim = c(0, max_y)) +
  theme_cowplot() +
  theme(panel.grid.major = element_line(color = "gray80", linetype = "dashed"),
        panel.grid.minor = element_line(color = "gray90", linetype = "dotted"))
p
```

What happens when we visualize this in a more "routine" time-course? We can check with the script:

```
p_time <- ggplot(solution) +
  geom_line(aes(x = time, y = x, color = "Prey"), linewidth = 1) +
  geom_line(aes(x = time, y = y, color = "Predator"), linewidth = 1) +
  scale_color_manual(values = c("Prey" = "blue", "Predator" = "red")) +
  labs(x = "Time", y = "Population",
       title = "Lotka-Volterra Dynamics: Population vs Time",
       color = "Species") +
  theme_cowplot() +
  theme(legend.position = "right",
        panel.grid = element_line(linetype = "dashed", color = "gray70"))
p_time
```

This gives us the oscillatory dynamics.

Now what happens we change the parameters? Let's say, for example that we increase the predator appetite, so that we simply re-set $a = 0.1$. Then, let's see what we get what in the new plot.

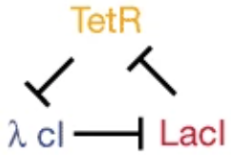
Exercise: Plot the time-course for the new parameterization. Describe in detail how and why the shape of the orbit changes due to this specific parameter change.

Exercise: The new orbit is still an orbit. Would this be true in a real population of predator/prey? Why?

D. melanogaster

Simplified repressilator model

The repressilator is an early example of a synthetic genetic regulatory network that exhibits oscillatory behavior, first described by Elowitz and Leibler in 2000. It consists of three genes arranged in a cycle, where each gene's protein product represses the expression of the next gene in the loop. In the original implementation in E. coli, the system was composed of the lacI gene (from the lac operon), encoding the Lac repressor protein, the tetR gene (from the Tn10 transposon), encoding the Tet repressor protein, and the cI gene (from bacteriophage λ), encoding the λ repressor protein. Each gene was placed under the control of a promoter that could be repressed by the previous gene's protein product. This could be visualized as follows (from Elowitz and Leibler, 2000):



The simplified repressilator model focuses on the dynamics of three proteins in a cyclic repression network. Each protein represses the production of the next protein in the cycle.

The model assumes that mRNA dynamics are fast compared to protein dynamics, allowing us to describe the system using only protein concentrations. The deterministic version of this model can be described by the following system of time-derivatives like the systems we looked at earlier:

$$\begin{aligned}
\frac{dx_1}{dt} &= \frac{\alpha}{1+x_3^n} + \alpha_0 - \beta x_1 \\
\frac{dx_2}{dt} &= \frac{\alpha}{1+x_1^n} + \alpha_0 - \beta x_2 \\
\frac{dx_3}{dt} &= \frac{\alpha}{1+x_2^n} + \alpha_0 - \beta x_3,
\end{aligned}$$

where

- x_i is the concentration of protein i
- α is the maximum production rate
- α_0 is the basal production rate
- n is the Hill coefficient, describing the cooperativity of repression
- β is the degradation rate.

Stochastic simulation

To capture the fact the stochastic counting-error in gene expression, we will use a simplified form of something called the Gillespie algorithm. This algorithm treats each reaction as a discrete event that occurs with a certain probability in a simulated form of continuous time.

```r
simplified_stochastic_repressilator <- function(initial_state, params, tmax) {
  alpha  <- params[1]
  alpha0 <- params[2]
  n      <- params[3]
  beta   <- params[4]

  t <- 0
  state <- as.numeric(initial_state)
  times <- t
  states <- matrix(state, nrow = 1)

  while (t < tmax) {
    x1 <- state[1]
    x2 <- state[2]
    x3 <- state[3]

    # Calculate propensities.
    production   <- alpha / (1 + c(x3, x1, x2)^n) + alpha0
    degradation  <- beta * state
    propensities <- c(production, degradation)

    a0 <- sum(propensities)

    # Time to next reaction.
    tau <- rexp(1, a0)

    # Choose next reaction.
    reaction <- sample(1:6, 1, prob = propensities/a0)

    # Update state.
    state_change <- rep(0,3)
    if (reaction <= 3) {
      state_change[reaction] <- 1  # protein production
    } else {
      state_change[reaction - 3] <- -1  # protein degradation
    }

    state  <- state + state_change
    t      <- t + tau
    times  <- c(times, t)
    states <- rbind(states, state)
  }

  return(list(times = times, states = states))
}
```

Simulation and visualization

We'll set parameters as follows:

```
alpha  <- 100   # Maximum production rate
alpha0 <- 0.01  # Basal production rate
n      <- 4     # Hill coefficient
beta   <- 1     # Degradation rate
params <- c(alpha, alpha0, n, beta)
```

Now let's perform a simulation:

```
initial_state <- c(50, 50, 50)
tmax   <- 100
result <- simplified_stochastic_repressilator(initial_state, params, tmax)
times  <- result$times
states <- result$states
```

Interactive 3D trajectory plot:

```
plot_3d <- plot_ly(x = states[,1], y = states[,2], z = states[,3],
                   type = "scatter3d", mode = "lines", height = 400,
                   width = 400,
                   line = list(width = 6, color = ~states[,1],
                               colorscale = "Viridis")) %>%
 layout(scene = list(xaxis = list(title = "Protein 1"),
                     yaxis = list(title = "Protein 2"),
                     zaxis = list(title = "Protein 3")),
                     title = "3D Trajectory")
```

To view this interactive 2D plot in your default Web browser, simply run this:

```
plot_3d
```

Or if you are running this in a Jupyter notebook or in a Google Colab environment, run this:

```
embed_notebook(plot_3d)
```

Function to calculate the derivative for the quiver plots:

```
repressilator_derivative <- function (X, Y, protein_idx) {
  Z <- mean(states[, (protein_idx %% 3) + 1])  # Use mean of the third protein.
  dX <- alpha / (1 + (Z^n)) + alpha0 - beta * X
  dY <- alpha / (1 + (X^n)) + alpha0 - beta * Y
  return(list(dX = dX, dY = dY))
}
```

2D quiver plot of Protein 1 vs. Protein 2:

```
X <- seq(0, max(states[,1]), length.out = 20)
Y <- seq(0, max(states[,2]), length.out = 20)
grid <- expand.grid(X = X, Y = Y)
derivatives <- repressilator_derivative(grid$X, grid$Y, 0)

# Reduce arrow length.
arrow_scale <- 0.05  # Adjust this value to change arrow length

# Create the quiver plot.
options(repr.plot.width = 6,repr.plot.height = 6,repr.plot.res = 150)
plot1 <- ggplot() +
  geom_segment(data = grid, aes(x = X, y = Y,
                                xend = X + arrow_scale * derivatives$dX,
                                yend = Y + arrow_scale * derivatives$dY),
               arrow = arrow(length = unit(0.1, "cm")),
               color = "gray", alpha = 0.5) +
  geom_path(data = as.data.frame(states), aes(x = V1, y = V2)) +
  labs(title = "Phase Portrait: Protein 1 vs 2",
       x = "Protein 1", y = "Protein 2") +
  theme_cowplot()
plot1
```

2D quiver plot of Protein 2 vs. Protein 3:

```
X <- seq(0, max(states[,2]), length.out = 20)
Y <- seq(0, max(states[,3]), length.out = 20)
grid <- expand.grid(X = X, Y = Y)
derivatives <- repressilator_derivative(grid$X, grid$Y, 1)
plot2 <- ggplot() +
  geom_segment(data = grid, aes(x = X, y = Y,
                                xend = X + arrow_scale * derivatives$dX,
                                yend = Y + arrow_scale * derivatives$dY),
               arrow = arrow(length = unit(0.1, "cm")),
               color = "gray", alpha = 0.5) +
  geom_path(data = as.data.frame(states), aes(x = V2, y = V3)) +
  labs(title = "Phase Portrait: Protein 2 vs 3",
       x = "Protein 2", y = "Protein 3") +
  theme_cowplot()
plot2
```

Exercise: Try running the simulation with $n = 1$, such that there is no cooperativity. Is there still oscillatory behavior? Why is cooperativity necessary/unnecessary?

Exercise: Let's assume that the trajectories in the plot show the distribution of cells in gene expression space. Frequently, PCA — which highlights the dimensions in gene expression space that preserve the greatest total variance — is used to reduce the dimensionality of such a scenario. How might one expect for PCA to fail in this case?