Session 1: 1D Ordinary Differential Equations

Building an Interactive Spruce Budworm Simulation

Applied Mathematics Lab

January 21, 2026

1 Introduction

Welcome to this hands-on session on modeling population dynamics using ordinary differential equations (ODEs). Today, we will explore the **spruce budworm model**, a classic example from ecological modeling that demonstrates how simple nonlinear systems can exhibit complex behaviors including multiple equilibria and catastrophic transitions.

1.1 Learning Objectives

By the end of this session, you will be able to:

- Understand the mathematical formulation of the spruce budworm model
- Implement the model as a Python function
- Solve the ODE numerically using `scipy.integrate.solve_ivp`
- Visualize the phase portrait and identify equilibrium points
- Build an interactive Streamlit application to explore the model

1.2 The Spruce Budworm Model

The spruce budworm is an insect that periodically devastates spruce forests. The population dynamics can be modeled by the following ODE:

$$\frac{dx}{dt} = rx\left(1 - \frac{x}{k}\right) - \frac{x^2}{1 + x^2} \tag{1}$$

where:

- $x(t)$ is the budworm population (adimensional)
- r is the intrinsic growth rate (typically $r \approx 0.5$)
- k is the carrying capacity of the forest (typically $k \approx 10$)

The first term represents logistic growth, while the second term models predation by birds (which follows a saturating functional response).

Reference: Strogatz, S. H. (2018). *Nonlinear Dynamics and Chaos*, Chapter 3.7.

2 Implementing the ODE Function

2.1 Task

Create a Python function that implements the spruce budworm differential equation. The function should follow the signature required by `scipy.integrate.solve_ivp`.

2.2 Requirements

- Function name: `spruce_budworm`
- Parameters: `t` (time), `x` (population), `r` (growth rate), `k` (carrying capacity)
- Return: The rate of change $\frac{dx}{dt}$
- Include appropriate docstring documentation

2.3 Hint

The function signature should be:

```python
def spruce_budworm(t: float, x: float, r: float = 0.5, k: float = 10)
    -> float:
    """Model for the spruce budworm population dynamics."""
    # Your implementation here
    dxdt = ???
    return dxdt
```

2.4 Mathematical Expression

Remember to implement:

$$\frac{dx}{dt} = rx\left(1 - \frac{x}{k}\right) - \frac{x^2}{1+x^2} \tag{2}$$

3 Phase Portrait Visualization

3.1 Task

Create a function that plots the rate of change $\frac{dx}{dt}$ as a function of the population x. This phase portrait will help us visualize the equilibrium points and their stability.

3.2 Requirements

- Function name: `plot_spruce_budworm_rate`
- Plot $\frac{dx}{dt}$ vs x for $x \in [0, k]$
- Identify and mark equilibrium points (where $\frac{dx}{dt} = 0$)
- Color-code equilibrium points:
  - Blue circles for stable equilibria (where $\frac{dx}{dt}$ crosses zero from above)

  - Red circles for unstable equilibria (where $\frac{dx}{dt}$ crosses zero from below)

- Add a horizontal line at $y = 0$
- Mark the current population x_t with a vertical dashed line

3.3 Hints

- Use `np.linspace` to create an array of x values
- To find zero crossings: look for sign changes using `np.diff(np.sign(dxdt))`
- Stability: a fixed point is stable if $\frac{dx}{dt}$ decreases as you pass through it

3.4 Key Concepts

- **Equilibrium point**: A value x^* where $\frac{dx}{dt} = 0$
- **Stable equilibrium**: Small perturbations decay back to x^*
- **Unstable equilibrium**: Small perturbations grow away from x^*

4 Numerical Integration

4.1 Task

Create a function that evolves the system forward in time using numerical integration. This function should solve the ODE and append the results to existing time and population arrays.

4.2 Requirements

- Function name: `evolve_spruce_budworm`
- Use `scipy.integrate.solve_ivp` with the RK45 method
- Parameters:
  - `t`: existing time array
  - `x`: existing population array
  - `r`, `k`: model parameters
  - `t_eval`: duration to evolve forward
- Start from the last values in the input arrays
- Concatenate new results to the input arrays
- Ensure population never goes negative (use `np.clip`)
- Return updated time and population arrays

4.3 Code Structure

```python
def evolve_spruce_budworm(
    t: np.ndarray, x: np.ndarray,
    r: float = 0.5, k: float = 10,
    t_eval: float = 50
) -> tuple[np.ndarray, np.ndarray]:
    # Define time span from last time point
    t_span = (t[-1], t[-1] + t_eval)

    # Create evaluation points
    t_eval_points = np.linspace(???)

    # Solve the ODE
    solution = solve_ivp(???)

    # Concatenate results
    t = np.concatenate((t, solution.t))
    x = np.concatenate((x, solution.y[0]))

    # Ensure non-negative population
    x = np.clip(x, a_min=0, a_max=None)

    return t, x
```

4.4 Important Notes

- The `args` parameter in `solve_ivp` passes additional arguments to your ODE function
- Initial condition should be `[x[-1]]` (last population value)
- Use `method="RK45"` for adaptive step-size Runge-Kutta integration

5 Time Series Visualization

5.1 Task

Create a function to plot the population dynamics over time. This visualization shows how the population evolves from the initial condition.

5.2 Requirements

- Function name: `plot_spruce_budworm`
- Plot time on the x-axis and population on the y-axis
- Use green color for the trajectory
- Ensure y-axis starts at 0 (populations cannot be negative)
- Include grid, labels, and title
- Return the figure and axes objects

5.3 Questions to Consider

- What happens to the population in the long run?
- Does the final state depend on the initial condition?
- How do different parameter values affect the dynamics?

6 Building the Streamlit Application

6.1 Task

Now that you have all the components, create an interactive Streamlit application that allows users to explore the spruce budworm model. This will enable real-time parameter adjustment and visualization.

6.2 Requirements

6.2.1 Sidebar Controls

Create sliders for:

- Growth rate r (range: 0.1 to 1.0)
- Carrying capacity k (range: 5 to 20)
- Initial population x_0 (range: 0.1 to k)
- Evolution time step (default: 50)

6.2.2 Interactive Features

- Display the differential equation with current parameter values
- Show the phase portrait (rate of change plot)
- Show the time series evolution
- Add a button to "Evolve Forward" that continues the simulation
- Use `st.session_state` to maintain simulation state between button clicks

6.2.3 Layout Structure

```python
import streamlit as st
import numpy as np
import matplotlib.pyplot as plt
from your_module import (
    spruce_budworm,
    plot_spruce_budworm_rate,
    evolve_spruce_budworm,
    plot_spruce_budworm
)

```

```python
st.title("Spruce Budworm Population Dynamics")

# Sidebar parameters
r = st.sidebar.slider("Growth rate (r)", ...)
k = st.sidebar.slider("Carrying capacity (k)", ...)
x0 = st.sidebar.slider("Initial population", ...)

# Initialize session state
if 'initialized' not in st.session_state:
    st.session_state.t = np.array([0])
    st.session_state.x = np.array([x0])
    st.session_state.initialized = True

# Display equation
st.latex(r"\frac{dx}{dt} = rx\left(1 - \frac{x}{k}\right) - \frac{x
    ^2}{1 + x^2}")

# Plot phase portrait
fig1, ax1 = plot_spruce_budworm_rate(...)
st.pyplot(fig1)

# Plot time series
fig2, ax2 = plot_spruce_budworm(...)
st.pyplot(fig2)

# Evolution button
if st.button("Evolve Forward"):
    st.session_state.t, st.session_state.x = evolve_spruce_budworm(...)
    st.rerun()
```

6.3 Advanced Features (Optional)

- Add a reset button to restart the simulation
- Display current population value and equilibrium states
- Add explanatory text about bifurcations and catastrophes
- Show multiple trajectories with different initial conditions
- Add animation of the population dynamics

7 Exploration Questions

Once your simulation is working, explore the following questions:

1. **Multiple Equilibria:** For $r = 0.5$ and $k = 10$, how many equilibrium points exist? Which are stable?
2. **Bistability:** Start with two different initial conditions (e.g., $x_0 = 1$ and $x_0 = 8$). Do they converge to the same equilibrium?
3. **Hysteresis:** Slowly increase the carrying capacity k from 5 to 15. Then slowly decrease it back to 5. Does the population return to the same state?

4. **Outbreak Dynamics:** What happens if you start with a small population ($x_0 < 2$) and the carrying capacity is large ($k > 10$)?
5. **Critical Slowing Down:** When the population is near an unstable equilibrium, how long does it take to move away? Compare this to the rate of change far from equilibrium.
6. **Parameter Space:** Create a diagram showing the number of equilibria as a function of r and k. Where do bifurcations occur?

8 Mathematical Background

8.1 Equilibrium Analysis

Equilibrium points satisfy:

$$rx^* \left(1 - \frac{x^*}{k}\right) - \frac{(x^*)^2}{1 + (x^*)^2} = 0 \tag{3}$$

This can be rewritten as:

$$rx^* \left(1 - \frac{x^*}{k}\right) = \frac{(x^*)^2}{1 + (x^*)^2} \tag{4}$$

The left side represents birth rate (logistic growth), and the right side represents predation rate. Equilibria occur where these balance.

8.2 Stability Analysis

The stability of an equilibrium x^* is determined by the sign of the derivative:

$$\frac{d}{dx}\left(\frac{dx}{dt}\right)\bigg|_{x=x^*} \tag{5}$$

If this derivative is:

- Negative: the equilibrium is stable (attracting)
- Positive: the equilibrium is unstable (repelling)
- Zero: higher-order analysis is needed

8.3 Ecological Interpretation

- **Low equilibrium:** Few budworms, controlled by predation
- **High equilibrium:** Outbreak state, budworms overwhelm predators
- **Middle equilibrium:** Usually unstable, separates the two basins of attraction
- **Hysteresis:** The system can "jump" between states depending on history

This behavior explains why spruce budworm populations can suddenly explode from low levels to outbreak proportions, and why simply reducing the outbreak may not return the forest to a healthy state.

9 Resources

- Strogatz, S. H. (2018). *Nonlinear Dynamics and Chaos.* CRC Press, Chapter 3.7
- SciPy documentation: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html`
- Streamlit documentation: `https://docs.streamlit.io`
- Reference implementation: `https://github.com/daniprec/BAM-Applied-Math-Lab/tree/main/sessions/s01_odes_1d`

10 Tips for Success

- **Start simple:** Get section 2 working first, then build up.
- **Test incrementally:** Verify each function works before moving to the next.
- **Use the reference:** The provided code is there to help you understand the structure.
- **Experiment:** Try different parameter values and see what happens.
- **Collaborate:** Discuss with your classmates, but write your own code.
- **Ask questions:** If you're stuck, ask for help!

Good luck and enjoy your coding!