Session 1: 1D Ordinary Differential Equations

Building an Interactive Spruce Budworm Simulation

Applied Math Modeling Lab

23rd January 2026

1 Introduction

Welcome to this hands-on session on mathematical modeling 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 [1, Chapter 3.7]:

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

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).

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:

```
def spruce_budworm(t: float, x: float, r: float = 0.5, k: float = 10)
    -> float:
    """Docstring and type hints"""
    # Your implementation here
    dxdt = ???
    return dxdt
```

Why do we need `t` as an input? Although the equation does not explicitly depend on time, `solve_ivp` requires the function to accept time as the first argument.

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`
- Parameters: `x_t` (current population), `r`, `k`
- Use `matplotlib` for plotting
- 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 $y = 0$ to indicate equilibria (null rate of change).
- Label axes and add a title.
- Mark the current population x_t with a vertical dashed line.

Figure 1 shows an example of the expected output.

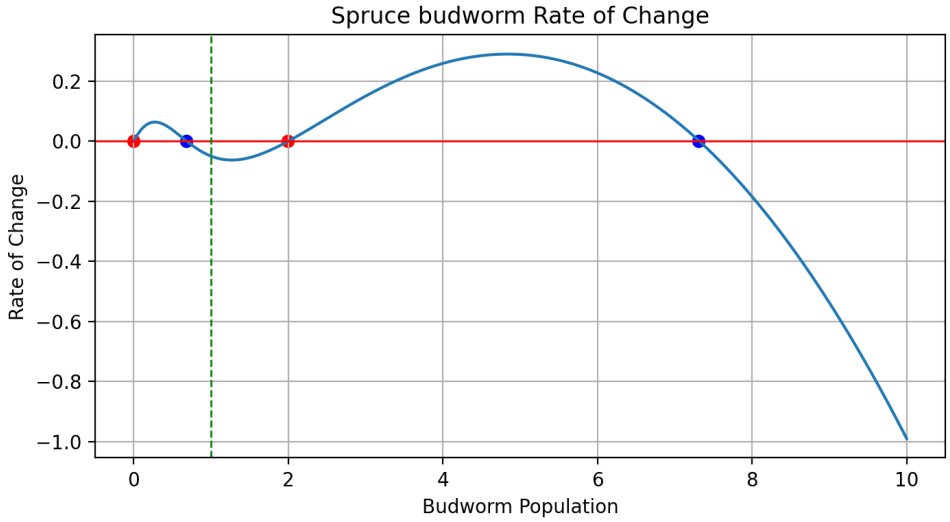

Figure 1: Example phase portrait of the spruce budworm model showing stable (blue) and unstable (red) equilibria. The green dashed line indicates the current population x_t. Where do you think this population will evolve?

3.3 Hints

- You can use `np.linspace` to create an array of x values.
- To find zero crossings: you can look for sign changes combining `np.diff` and `np.sign`.
- 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`
- Inputs: `t` (time array), `x` (population array), `r`, `k`, `t_eval` (duration to evolve).
- Use `scipy.integrate.solve_ivp` with the RK45 method.
- 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 `t` and `x` arrays.

4.3 Hints

```python
def evolve_spruce_budworm(t: np.ndarray, x: np.ndarray, ...):
    """Don't forget the docstring and type hints"""
    # Define time span from last time point
    # This indicates the start and end times for integration
    t_span = (t[-1], t[-1] + t_eval)

    # Create evaluation points, t_eval
    # This indicates where we want the solution evaluated
    # and should be distributed along the time span
    # Hint: use np.linspace

    # Solve the ODE
    solution = solve_ivp(
        fun=spruce_budworm,
        t_span=t_span,
        y0=[x[-1]],
        t_eval=t_eval,
        args=(r, k),
        method="RK45"
    )
    t_new = solution.t
    x_new = solution.y[0]

    # Concatenate results
    # Hint: use np.concatenate

    # Ensure non-negative population
    # Hint: use np.clip

    return t, x
```

4.4 Important Notes

- The `args` parameter in `solve_ivp` passes additional arguments to your ODE function. You can see how we use it in the example above.
- 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`
- Parameters: `t` (time array), `x` (population array).
- 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.

See Figure 2 for an example output.

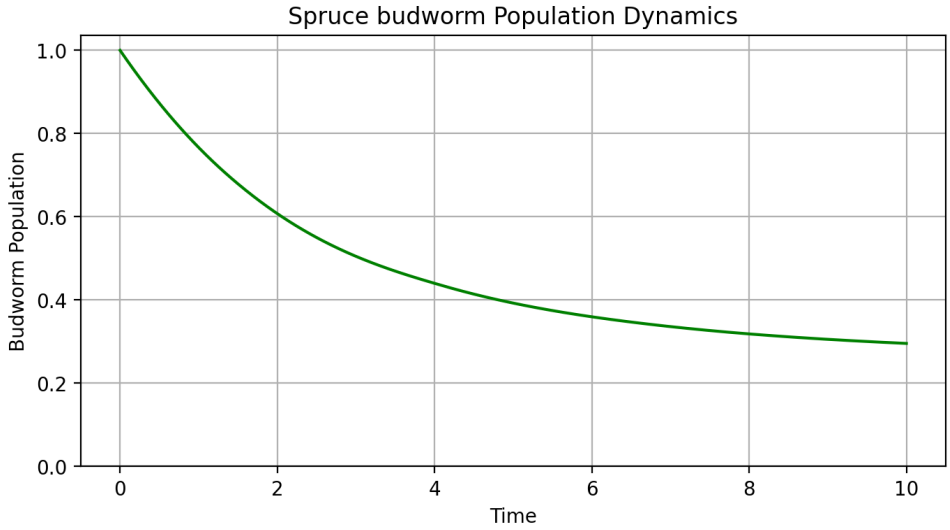

Figure 2: Example time series of the spruce budworm population over time.

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

While all the previous code could be done in Google Colab, Streamlit has to be build on your local machine. Follow the instructions in the `README.md` file in the repository to set up your environment.
Design an script, name it `spruce_budworm_app.py`. You will find a suggested layout below. This script will use the functions you implemented in sections 2 through 5. You can either import them from a separate module (another script you created) or paste the function definitions directly into the script. For best practices, consider creating a module (e.g., `spruce_budworm_model.py`) and importing the functions.
To test your app, run the following command in your terminal:

```
streamlit run spruce_budworm_app.py
```

6.2.1 Sidebar Controls

Create sliders for:

- Growth rate r (range: 0.0 to 1.0, default 0.5).
- Carrying capacity k (range: 0.1 to 10.0, default 10.0).
- Initial population: set automatically to $k/10$ (the app uses $x_0 = k/10$ by default).
- Time slider for evolution (range: 1 to 100, default 10).

6.2.2 Interactive Features

- Display the differential equation with current parameter values.
- Show the phase portrait (rate of change plot), using your function from section 3.
- Show the time series evolution, using your function from section 5.
- Add a button to "Evolve Forward" that continues the simulation, updating the plots.
- Use `st.session_state` to maintain simulation state between button clicks. You will need to store the time and population arrays in the session state, otherwise they will reset on each interaction.

6.2.3 Layout Structure

```
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
)  # Or paste your functions here

st.title("Spruce Budworm Population Dynamics")

# Sidebar parameters (as in the app)
r = st.sidebar.slider("Intrinsic growth rate (r)", 0.0, 1.0, 0.5)
k = st.sidebar.slider("Carrying capacity (k)", 0.1, 10.0, 10.0)

# The app sets the initial population to k/10 by default
x0 = k / 10

# Initialize session state
if ("sbw_x" not in st.session_state):
    st.session_state["sbw_t"] = np.array([0])
    st.session_state["sbw_x"] = np.array([x0])

# Time slider and control buttons
t_eval = st.sidebar.slider("Time", 1, 100, 10)
button = st.sidebar.button("Evolve")

# Retrieve session data
t = st.session_state["sbw_t"]
x = st.session_state["sbw_x"]

# Evolve if requested
if button:
    t, x = evolve_spruce_budworm(t, x, r=r, k=k, t_eval=t_eval)
    st.session_state["sbw_t"] = t
    st.session_state["sbw_x"] = x

# Plot phase portrait and time series
fig1, ax1 = plot_spruce_budworm_rate(x[-1], r=r, k=k)
st.pyplot(fig1)
fig2, ax2 = plot_spruce_budworm(t, x)
st.pyplot(fig2)
```

6.3 Advanced Features (Optional)

- Add a reset button to restart the simulation.
- 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? If you are interested in this concept, see section 7.1.
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?

7.1 Slow-Fast Dynamics

The carrying capacity k can be interpreted as a slowly varying parameter in real ecosystems (e.g., due to seasonal changes or forest management). You can simulate this by gradually changing k over time in your app. This can lead to **hysteresis** effects, where the population does not return to its original state after k is restored. Experiment with this by modifying your Streamlit app to allow k to vary over time.

8 Deliverable

Implement the complete Streamlit application as described above, following sections 2 through 6. Ensure that all functions are correctly defined and integrated into the app. Test the application thoroughly to confirm that it behaves as expected.
Answer at least three of the exploration questions from section 7 and document your findings in a brief report (1-2 pages). You are encouraged to use LaTeX here. Include screenshots of your application demonstrating different behaviors observed during your exploration.
If you want to go the extra mile, here are some additional challenges you can tackle:

- Create a GitHub repository for your project and push your code there. You can include the text of your report in the repository as well.
- Deploy your Streamlit app using Streamlit Cloud or another hosting service. Share the link in your report.
- Implement some of the advanced features mentioned in section 6.
- Instead of using SciPy's built-in ODE solver, implement your own simple Euler or Runge-Kutta integrator and compare results.

9 Mathematical Background

In this section I provide some additional mathematical context for the spruce budworm model. Play with your simulation to see these concepts in action!

9.1 Equilibrium Analysis

Equilibrium points satisfy:

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

This can be rewritten as:

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

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

9.2 Stability Analysis

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

$$\frac{d}{dx}\left(\frac{dx}{dt}\right)\Bigg|_{x=x^*} \tag{4}$$

If this derivative is:

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

9.3 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.

10 Resources

- [1, Chapter 3.7] for theoretical background on the spruce budworm model.
- 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`

11 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 (and additional documentation) is there to help you understand the structure.
- **Experiment:** Try different parameter values and see what happens.
- **Collaborate:** Discuss with your teammates, divide the work if needed. You can also work separately and then compare your implementations.
- **Ask questions:** If you're stuck, ask for help!

Good luck and enjoy your coding!

References

[1] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. Chapman and Hall/CRC, 2024.