Worksheet 6

Lotka-Volterra-Equations and Fractals

Networks and Complex Systems

F. Klimm und B.F. Maier

Schülerakademie 5.2 (Roßleben 2016)

Exercise 6.1 Numerical Solutions of Differential Equations

Often it is not possible to solve *Ordniary Differential Equations* (ODE) analytically. In such cases we have to use numerical solvers to find approximate solutions. One of the most simplistic procedures is the so-called *Euler method*, which we outline below.

Given the ODE

$$\dot{y} = f(t, y). \tag{1}$$

with the initial value $y(t_0) = y_0$, we can compute the following value y_{k+1} from the current value (t_k, y_k) by using

$$\dot{y}_{k+1} = y_k + h \cdot f(t_k, y_k). \tag{2}$$

This is a *linear* approximation and h gives the step size. We find the whole numerical solution by iteratively applying this formula. Usually, smaller step widths result in a better numerical solutions.

Subexercise 6.1.1 Exponential Growth

In the beginning we want to analyse a one-dimensional problem which might be familiar, the *exponential growth*.

Find the analytical solution for the following ODE

$$\dot{x}(t) = \lambda x(t) \tag{3}$$

$$x(t_0) = x_0. \tag{4}$$

Subexercise 6.1.2 One-Dimensional Euler Method

Solve the ODE of Equation (4) numerically with the Euler method. You can choose $h = 1$, $x_0 = 3$, and $\lambda = 1.2$.

Illustrate the temporal behaviour $x(t)$ and compare it with the analytical solution. How does the system behave if the step size h is chosen too large?

Optional: Vary $\lambda \in \{-2, -0.1, 0.1, 2\}$. Illustrate the different curves $x_\lambda(t)$ and describe the behaviour.

Subexercise 6.1.3 Numerical Stability Analysis

We want to investigate the stability of the fixed points x^*. At first we use the analytical approach. Subsequently we validate our findings numerically. To do so we perturb the ODE a small step ε from a fixed point and check with the Euler method whether the system returns to the fixed point or moves further away. Analyse the behaviour of all fixed points for $\lambda = \{-1, 0, 1\}$.

Subexercise 6.1.4 Optional: Error Estimation for the Euler Method

For the simple ODE of Equation (4) we know the analytical solution. Therefore, we can calculate the deviation or *numerical error* of the Euler method in dependence of the step size h.

To achieve this calculate and illustrate $\mathrm{Error}(h) = |x_{\mathrm{analytic}}(t) - x_{\mathrm{numerical}}(t,h)|$. Chose for example $t = 100$ and vary the step size h. How does the numerical error $\mathrm{Error}(h)$ depend on the step size h?

Subexercise 6.1.5 Numerical Solution of the Lotka-Volterra Equations

We can use the Euler method for high dimensional ODE's, as well. Here, we want to analyse the two-dimensional *Lotka-Volterra equations*, which we discussed already earlier and describe the relationship between animal species.

This predator prey system is given by

$$\dot{x}(t) = x(3 - x - 2y) \tag{5}$$
$$\dot{y}(t) = y(2 - x - y)\,. \tag{6}$$

The Euler method works similar to the one-dimensional case, as

$$x_{k+1} = x_k + \dot{x}(t)_k \tag{7}$$
$$y_{k+1} = y_k + \dot{y}(t)_k\,. \tag{8}$$

Now use the Euler method to solve the Lotka-Volterra equations numerically. Use different initial conditions and investigate which of the analytically expected fixed points are observed. To achieve this illustrate $x(t)$ and $y(t)$. Are any of the expected fixed points not observed and if so why is this the case?

Subexercise 6.1.6 Optional: Oscillations in the Lotka-Volterra System

Another variant of the Lotka-Volterra equations is given by

$$\dot{x}(t) = x(\alpha - \beta y) \tag{9}$$
$$\dot{y}(t) = y(\gamma - \delta y)\,. \tag{10}$$

Illustrate different *trajectories* $(x(t), y(t))$ for many choices of the parameters α, β, γ und δ and initial conditions. For example, try $\alpha = 2/3$, $\beta = 4/3$, $\gamma = 1$ and $\delta = 1$ and $x_0 = y_0 = 0.9$.

Subexercise 6.1.7 Optional: *Basins of Attraction* of the Lotka-Volterra System

Research the definition of *Basins of Attraction* and detect them numerically for a certain choice of parameters for the Lotka-Volterra system.

Exercise 6.2 Fractals and Self-Similarity

In the morning we discussed the *Koch curve* from two perspectives, as a Lindenmayer system (L-system) and as a fractal. Now we want to draw this fractal by iteratively using the L-system steps:

$$\text{variables} = \{F\} \tag{11a}$$
$$\text{constants} = \{+, -\} \tag{11b}$$
$$\text{productionrules} = \{F \to F - F + +F - F\} \tag{11c}$$
$$\text{axiom} = F\,. \tag{11d}$$

The F indicates that the *turtle* (which is a virtual drawing pen) moves **F**orward, $-$ that it rotates by 60° leftwards, and $+$ that it rotates 60° rightwards.

Subexercise 6.2.1 Iterative Application of the L-system

We want to write a Python function that applies the L-system rules iteratively n times such that we receive a set of instructions that tells the turtle to draw a Koch curve. The *axiom* is the initial condition.

Subexercise 6.2.2 Teenage Mutant L-System Turtle

The following program is an introdution to drawing with *turtle*. [1]

Change the commands in the first par of the program and discuss how it changes the behaviour. What does the function at the end of the code achieve?

```python
import turtle
import numpy as np

# setze die Geschwindigkeit des Zeichners auf Maximum
turtle.speed(0)

# Hebe den Stift des Zeichners (sodass bei Bewegung
# nicht gezeichnet wird
turtle.penup()

# Bewege den Zeichner zur neuen Startposition
turtle.setpos(-250,250)

# Setze den Stift nieder, sodass nun gezeichnet wird
turtle.down()

# Bewege den Zeichner 30 pixel vorwaerts (zeichne Linie)
turtle.forward(30)

# Drehe den Zeichner 60 Grad nach links
turtle.left(60)

# Gehe 30 Pixel nach vorne (zeichne dabei), drehe 60 Grad nach rechts
turtle.forward(30)
turtle.right(60)

# Bestimme, dass das Zeichenfenster erst bei einem Klick geschlossen wird
turtle.exitonclick()

# Was macht diese Funktion?
def draw_mysterious_thingy(n,base_r=100):

    forward_length = 2 * base_r * np.sin(np.pi/n)
    degree = 360./n

    turtle.penup()
    turtle.setpos(0,0)
    turtle.pendown()

    for i in range(n):
        turtle.forward(forward_length)
        turtle.left(degree)
```

Subexercise 6.2.3 Drawing a Koch curve

Write a function that gives the result of an n-fold iteration of the L-System (11) to a *turtle* and then draws it. Note that the forward movement F has to be scaled by the number n of iterations, specifically with $1/3^n$.

How do we have to change the axiom such that not the Koch curve but the Koch snowflake is produced?

[1] A detailed documentation of the *turtle*-commands is available under `https://docs.python.org/2/library/turtle.html`

Subexercise 6.2.4 General L-Systems

Generalise the function from Exercise 6.2.1, such that arbitrary L-systems can be produced and iterated n-fold.

Exercise 6.3 Creating Fractals

We discuss the L-system

$$\text{variables} = \{A, B\} \tag{12a}$$
$$\text{constants} = \{+, -\} \tag{12b}$$
$$\text{productionrules} = \{A \to +B - A - B+,\ B \to -A + B + A-\} \tag{12c}$$
$$\text{axiom} = A\,. \tag{12d}$$

Here, A and B indicate that the turtle is moving forward. The commands $-$ and $+$ represent left and right rotations by 60°.

Which fractal is created?

Subexercise 6.3.1 Optional: L-Systems with Memory

L-systems with memory denote systems where the turtle saves the current position in two-dimensional space when the command [occurs. It then follows the commands inside the bracket until the command]. Then it 'jumps' to the saved position from [and follows further commands.

We discuss the L-system

$$\text{variables} = \{F\} \tag{13a}$$
$$\text{constants} = \{+, -, [,]\} \tag{13b}$$
$$\text{productionrules} = \{F \to F[+F]F[-F]F\} \tag{13c}$$
$$\text{axiom} = F\,. \tag{13d}$$

F is again a forward movement and $+, -$ indicate rotations by $180°/7$. What does this L-system resemble?