Programming Euler’s Method

Mathematical biologists often use numerical simulation to figure out how dynamical systems will behave. The simplest simulation method is Euler’s method, described in the text. Like all numerical integration algorithms, Euler’s method involves repeatedly performing a series of steps. This lab will guide you through programming an implementation of Euler’s method and using it to simulate the logistic equation, \( N' = 0.2N(1 - \frac{N}{100}) \).

**Exercise 1.** Work through a few steps of Euler’s method by hand, noticing each step. Make notes on what you do. Use your notes to type an outline of a program for Euler’s method into SageMath.

**Exercise 2.** Write code that will use Euler’s method to simulate the differential equation \( N' = 0.2N(1 - \frac{N}{100}) \) with the initial condition \( N(0) = 10 \) and a step size of \( h = 0.1 \). You will probably want to follow the steps below.

1. Write code that performs one step of Euler’s method, without iteration.
2. Iterate this procedure to find the population value at \( t = 100 \). (It should be approximately 99.99.) Your final program should store all the intermediate values of \( N \) in a list but, depending on your approach, you may find it easier to start without this. HINT: What parts of your code need to change from one step to the next? You may want to review Lab 3.

**Exercise 3.** Modify your code to experiment with different initial conditions. (Try at least three.) What happens when \( N(0) > k \)? (Recall that in the logistic model, \( k \) represents the carrying capacity.)

In plotting your results, you may have noticed that the horizontal axis of your plot is labeled from 0 to 1000, even though the simulation only ran through \( t = 100 \). This happens because `list_plot(somelist)` plots each value in the list against its position within the list. You can specify a list of time values to use instead, by using the function `zip`. (You may remember this from Lab 2. If not, you might want to review that lab now, around pages 7–8.) Briefly, the `zip` function takes two lists and turns them into a list of ordered pairs. (It can also take more than two lists and turn them into a list of n-tuples.) This is exactly the kind of input that `list_plot` needs. Therefore, to plot \( N \) as a function of \( t \), you have to use

```python
list_plot(zip(my_t_list, my_N_list))
```
Exercise 4. Create a list of time steps, from 0 to $t_{\text{final}}$, in steps of $h$. Plot $N$ as a function of $t$. What happens if you omit `zip`?

It would be very useful to be able to use your code to simulate any differential equation for any initial condition. This can be done by taking advantage of SageMath’s ability to have functions take other functions as inputs. The user of this function should first create a math function for computing $N'$. (Remember how a differential equation is actually a function.) The name of this function (say, `Nprime` or `vectorfield`) will be one of the inputs to your Euler’s method function.

Exercise 5. Turn your implementation of Euler’s method into a function. (You may want to review the advice in Lab 3 on turning scripts into functions.) This function should take a differential equation (see the previous paragraph), an initial value, a step size, and the number of steps as inputs. It should return a list of state values as its output.

Exercise 6. Modify this function so it returns a list of pairs of time and state values.

Exercise 7. Modify this function so that instead of taking the number of steps as an input, it takes the number of time units over which to run Euler’s method.

Remember that Euler’s method provides an approximate solution to differential equations, not an exact one. The smaller the step size, the more accurate the solution. If the step size is too large, things can go badly wrong.

Exercise 8. Using Euler’s method, simulate the differential equation $X' = -2.1X$ for 10 time units using a step size of 1. (When plotting the result, use the option `plotjoined=True`.) Briefly describe the result.

Exercise 9. Do the same simulation with step sizes of 0.5 and 0.1, adjusting the number of steps accordingly. (You may want to use your function that takes a final time rather than a number of steps as input.) What do you observe?