3 Differential equations

The key difference between difference equations and differential equations is that the former model a system in discrete time, i. e. the evolution between one point in time and the next, while the latter model a system in continuous time. This raises an important issue for simulations of such models: computers only work in discrete states (1 & 0, ultimately), so "true" continuity cannot be implemented. Hence, we have to work with approximations. Usually that creates a trade-off between accuracy (how close is the solutions to the true, e. g. analytically derived solution?) and computational efficiency (how much work does the computer have to do, and how long is it going to take?).

In this tutorial, two different timestepping methods will be introduced: the explicit Euler method and Runge's central difference quotient method. You will write functions to simulate the behaviour of a first-order differential equation. We can then compare the accuracy of both solutions by calculating the errors relative to the analytical solution, and also plot the behaviour of the system over time.

A Taylor expansion, finite differences

Differential equations are of the basic form

$$\frac{\partial y}{\partial t} = y'(t) = f(y(t), t) \tag{1}$$

i. e. the rate of change of y, its time derivative, is a function of y itself, and sometimes time as well. In order to simulate one time step of size δ, apply a second-order Taylor expansion and rearrange in the following way:

$$y(t+\delta) = y(t) + \delta y'(t) + \frac{\delta^2}{2} y''(\eta) \tag{2}$$

$$\Leftrightarrow \quad \frac{y(t+\delta) - y(t)}{\delta} = y'(t) + \frac{\delta^2}{2} y''(\eta) \tag{3}$$

with $\eta \in [t, t+\delta]$. The difference quotient on the left-hand side converges to $y'(t)$ as $\delta \to 0$.

B Explicit Euler

The first method, the explicit Euler method, applies an approximation of this relationship in a straightforward fashion:

$$\frac{y(t+\delta)-y(t)}{\delta} \approx y'(t) = f(y(t),t) \tag{4}$$
$$\Leftrightarrow \quad y(t+\delta) \approx y(t) + \delta f(y(t),t) \tag{5}$$

i. e. we move along the tangent line of the function at the current position. In terms of time-stepping, this leads to the scheme

$$y(0) = y_0 \tag{6}$$
$$y(\delta) = y(0) + \delta f(y(0),0) \tag{7}$$
$$\vdots \tag{8}$$
$$y(T) = y(T-\delta) + \delta f(y(T-\delta), T-\delta) \tag{9}$$

There is a small error related to the higher order terms of the Taylor expansion, but the smaller δ, the higher the accuracy we achieve. On the other hand, with smaller δ, we need more time steps, leading to higher computational effort, as more calculations have to be performed.

B.1 Exercise

Consider the continuous-time version of the cobweb model (tutorial 3, exercise 2), leading to the first-order differential equation $p' = 45 - 3p$. Complete the function `cobweb_ee_1step` by implementing one time step using this scheme! It should take the current price p, as well as the size of the time step δ as inputs and return the estimated new price after a step of size δ.

C Central difference quotient

Instead of applying a Taylor expansion around the point $y(t)$ to approximate $y(t+\delta)$, we can apply the Taylor expansion in the mid-point between the two, i. e. at $y(t+\frac{\delta}{2})$. In that case we obtain expressions for both $y(t)$ and $y(t+\delta)$:

$$y(t) \approx y\left(t+\frac{\delta}{2}\right) + \frac{\delta}{2}y'\left(t+\frac{\delta}{2}\right) + \frac{\delta^2}{8}y''\left(\frac{\delta}{2}\right) + \frac{\delta^3}{48}y'''(\eta_+), \quad \text{and} \tag{10}$$
$$y(t+\delta) \approx y\left(t+\frac{\delta}{2}\right) - \frac{\delta}{2}y'\left(t+\frac{\delta}{2}\right) + \frac{\delta^2}{8}y''\left(\frac{\delta}{2}\right) - \frac{\delta^3}{48}y'''(\eta_-) \tag{11}$$

with $\eta_+ \in [t, t+\frac{\delta}{2}]$ and $\eta_+ \in [t+\frac{\delta}{2}, t+\delta]$.

Taking the difference of both:

$$y(t+\delta) - y(t) = \delta y'\left(t+\frac{\delta}{2}\right) + \frac{\delta^3}{24}y'''(\eta) \tag{12}$$
$$\frac{y(t+\delta)-y(t)}{\delta} = y'\left(t+\frac{\delta}{2}\right) + \frac{\delta^2}{24}y'''(\eta) \tag{13}$$

with $\eta \in [t, t+\delta]$. While using the standard difference quotient (as in Euler's method) yields an error that is proportional to δ, a method based on the central difference quotient will produce errors proportional to δ^2, which is much smaller (considering $\delta < 1$). This is the idea behind the method of Runge.

D Runge's method

Use the central difference quotient to implement a time-stepping method:

$$\frac{y(t+\delta) - y(t)}{\delta} \approx y'\left(t + \frac{\delta}{2}\right) \tag{14}$$

$$\Leftrightarrow \quad y(t+\delta) \approx y(t) + \delta y'\left(t + \frac{\delta}{2}\right) \tag{15}$$

Of course we do not know $y\left(t + \frac{\delta}{2}\right)$, but we can simply approximate it with an explicit Euler step of size $\frac{\delta}{2}$. From the second step onward, we can re-use the values we already calculated in the previous step.

D.1 Exercise

Implement a single step of Runge's method for the continuous-time cobweb model! I. e. complete the function `cobweb_runge_1step`. You will have to calculate the midpoint state by implementing a single Euler step of size $\delta/2$ to calculate the next step. you can use the function you created above for that purpose.

E Simulation

Ultimately, we are not interested in tiny steps of a model, but longer time series and the convergence to equilibrium, or divergence to $\pm\infty$. Hence, in the next step, your task will be to implement many of those single steps in a row. Both functions for single steps are using the same structure of inputs 'p' and 'delta', and an output of the price after one incremental time step, so we can write a function that can flexibly apply either time-stepping method.

E.1 Exercise

Complete the function `simulate_cobweb`! As inputs, it takes an initial value `p`, the number of unit steps `T`, and the number of increments per unit step, `n`. Hint: $\delta = 1/(n+1)$. It should not return a whole time series of values, but only the final value at time T. The final input is the specific time-stepping function to be applied in this simulation function, i.e. `timestep_func`. Most programming languages support passing functions as inputs to other functions, along the lines of the following pseudo code:

```
FUNCTION my_func(other_func, param)
  other_func(param)
END FUNCTION
```

I.e. you simply pass the function as an input without evaluating it - often this can be achieved by leaving out the parentheses, but please check the specifics of the programming language of your choice. In the function block, you then run this function "`other_func`" with the parameter "`param`" that had also been passed as an input. As both time-stepping functions require the same parameters, we can write a single function to simulate the model, where the exact time-stepping method can be exchanged flexibly.

Ultimately, we use this function to compare convergence behaviour, i.e. accuracy of both functions relative to the known analytical solution. The code to run this comparison is provided in the script. If you implemented the time-stepping methods correctly, you should see that as the step size δ is halved, the errors are also halved using the Euler method, but quartered if we apply Runge's method. Hence, central difference exhibit better convergence behaviour and we can reach the same level of accuracy with lower computational effort.

F Simulating time series

In the last simulation exercise, we were only interested in calculating the value of the state variable p after a number of time steps T. Often, we are however interested in the time series of the model, starting from a given initial value. To practice the two time-stepping schemes a bit more, we will introduce a different first-order differential equation, simulate it over time, and finally plot the result.

F.1 Exercise

Consider the equation $y' + 3y = 2$. Complete the three functions for a single incremental step using the Euler and Runge methods (`ts_ee_1step` and `ts_runge_1step`), and a function that can apply either of these methods to simulate a time series for a given number of time steps (`ts_simulate`). The output should be of a suitable data structure that can store the time series from the initial value to the final one (array, list...).

The output is then visualised with the code in the script, alongside the analytical solution. If everything is correct, you should observe that the error (distance to the analytical solution) is substantially smaller when Runge's method is applied.