2 Systems of difference equations

If there are multiple variables that are related through some kind of linear function, we can express the system of equations in matrix-vector notation. Simulating the system then works exactly the same way as previously, with the only difference that we are iterating an equation that consists of matrices and vectors, rather than scalars. In this tutorial, we will also touch upon transforming a higher-order system into a first-order system of higher dimensionality, as it makes the computation somewhat simpler.

In the Python version, the NumPy (Numerical Python) package is introduced, since it has excellent support for linear algebra. It is a package with specific array structures and highly optimised computational procedures. It is the standard that most machine learning and artificial intelligence applications are built upon, and, as a rule of thumb, should be used whenever you want to process numbers in Python. In MATLAB, it is more straightforward: ultimately it is the MATrix LABoratory, so linear algebra is what it really excels at.

A First-order 2D system

Consider the 2-dimensional system from tutorial 2, exercise 1:

$$y_{t+1} = 0.7y_t + 0.3z_t \tag{1}$$
$$z_{t+1} = 0.1y_t + 0.75z_t + 15 \tag{2}$$
$$y_0 = 225 \tag{3}$$
$$z_0 = 25 \tag{4}$$

A.1 Exercise

Do the following:

- re-write the system as a single equation of the form $\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{g}_t$
- implement vectors as arrays and the matrix as a 2-dimensional array (nested array, i. e. arrays in an array)
- implement one time step of the model in the function `system_1step`. As you only need to implement a matrix-vector multiplication, it is fairly general and will also be applied to other models later on in the exercise.

Try the first part (re-writing the equation) on paper first to get a better intuition for the problem.

A.2 Exercise

Implement the coefficient matrix $\mathbf{A}$, the vector of initial values $\mathbf{yz}_0$, and the offset $\mathbf{g}$ in the script.

A.3 Exercise

In the script, complete the for loop to iterate through the system T times. Store the results in a $2 \times (T + 1)$ array (one row per variable).

Implementing the for-loop is identical to the last tutorial, but now you have to be careful to use correct indexing, so that simulation results are stored correctly in the predefined array.

To examine the results, we will create one plot in which both variables are plotted over time, and one in the phase space. Note the use of subplots to combine both plots in a single figure.

Moreover, note that the isoclines are plotted in the phase space. Those are lines at which *one* of the variables is in steady state. The intersection is obviously the point at which the entire system is in steady state. We calculate it by using the equation of one variable, setting it to its steady state value, and solving it so that we have a functional form of one variable in dependence of the other. Here that means:

Isocline of y:

$$\bar{y} = 0.7\bar{y} + 0.3z_t \quad \Leftrightarrow \quad \bar{y} = z_t$$

and of z:

$$\bar{z} = 0.1y_t + 0.75\bar{z} + 15 \quad \Leftrightarrow \quad \bar{z} = 0.4y_t + 60$$

Since this is a linear system, the isoclines are also linear functions in the phase space.

B Second-order difference equation as first-order 2D system

In the second exercise, we will convert the second-order difference equation from the first PC tutorial,

$$y_t = 1.1y_{t-1} - 0.6y_{t-2} + 1100, \tag{5}$$

into a 2-dimensional system of first-order equations.

B.1 Exercise

First, set up the first-order system (2D system of first-order differential equations).

Use as initial conditions the same as last time: $y_0 = 1300$, $y_1 = 1600$. You can then apply your 1-step function `system_1step` from above, with a new initial vector `yz_0`, and a different coefficient matrix `A` and inhomogeneous part `g`.

Try to re-write the system on paper first, that will make it easier. Finally, the results are visualised in a similar plot to the one for the first exercise. There is no need to plot z_t over time, since it is y_{t-1} and therefore does not add any information. Note what the oscillatory behaviour translates to in the phase space: the convergence happens in circles around the steady state. This is because the coefficient matrix relates to a rotation in the phase space, if we interpret it geometrically.