SCIENTIFIC COMPUTING USING PYTHON

1. PYTHON + SCIENTIFIC COMPUTING, PROJECT

Torben Larsen, Thomas Arildsen, Tobias L. Jensen

June 13, 2023

The Project

This project relates to the Lorenz attractor [5]. In his 1963 paper [5], Edward Lorenz presented a simple mathematical model for atmospheric convection, which was based on three coupled first order Ordinary Differential Equations (ODEs) as:

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y - x)\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= x(\rho - z) - y \\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy - \beta z\end{aligned} \tag{1}$$

where we interpret (x,y,z) as a position in space (time dependence exists but is not part of the notation), t is time, and (σ,ρ,β) are model parameters. As seen from (1) the set of equations is nonlinear, three-dimensional and deterministic. The formulation in (1) is continuous in time. To enable simulation of the equations it is necessary to map the formulations to discrete time. To keep the task fairly simple we assume that uniform sampling is used such that we determine the solutions via:

$$t \leftarrow nt_\delta, \qquad n \in \{0,1,\ldots,N\} \tag{2}$$

It is possible to simulate the behaviour of the equations represented by (1) in several different ways. There exists multiple ODE solvers readily available but it is likely obvious that we want you to implement your own solver, visualisation, data storage etc. However, for comparison you can of course use e.g. a standard ODE solver from e.g. `scipy`. In terms of choosing a specific way to solve the equation system, we propose that you at least start with a simple Euler (finite difference) approach before moving on to more advanced and accurate Runge-Kutta solvers or other types [1–4,6]. We recommend the Euler method and encourage you to focus on that unless you want a challenge.

An Euler approach can be derived from the general approximation using a variable named v as:

$$\frac{\mathrm{d}v}{\mathrm{d}t}((n+1)t_\delta) \simeq \frac{v((n+1)t_\delta) - v(nt_\delta)}{t_\delta} \tag{3}$$

Denoting the general variable $v(nt_\delta) = v[n]$ leads to the specific first equation of (1) as:

$$\frac{x[n+1] - x[n]}{t_\delta} \simeq \sigma(y[n] - x[n]) \tag{4}$$

Separating $x[n+1]$ on the left hand side of the 'equation' means that we can compute $x[n+1]$ from the previous $x[n]$. As long as we have an initial condition for $(x[0], y[0], z[0])$ we can thus simulate all states (x,y,z) of (1).

Regarding the specific choices of t_δ and N we expect that you figure out appropriate values through experimenting. The two variables are linked in the sense that N needs to be larger if t_δ is reduced to keep the same observation interval. However, to give you an idea of some values that may make sense you could consider $t_\delta = 0.01$ and $N = 50000$. But you need to make your own choice here – choose values that make sense to see what happens when you solve the equations.

You must simulate the following parameter cases with different system parameters. We expect that you find a combination of t_δ and N that works for all parameter cases – if possible. The parameter cases are:

1. $\sigma = 10$, $\beta = \frac{8}{3}$ and $\rho = 6$.
2. $\sigma = 10$, $\beta = \frac{8}{3}$ and $\rho = 16$.
3. $\sigma = 10$, $\beta = \frac{8}{3}$ and $\rho = 28$.
4. $\sigma = 14$, $\beta = \frac{8}{3}$ and $\rho = 28$.
5. $\sigma = 14$, $\beta = \frac{13}{3}$ and $\rho = 28$.

For each of the parameter cases we wish to compute the solution for (x,y,z) in discrete time indicated by for example the notation $(x[n],y[n],z[n])$ for $n = 0,1,\ldots,N-1$. As mentioned already, you are bound to experience that initial conditions corresponding to values for $(x[0],y[0],z[0])$ are needed. We expect that you propose some possible initial conditions and see how they affect your results.
When you perform the simulations we want to have access to the following data. This means that we expect that your software makes a folder (or a database if you prefer) containing the following:

- 3D `pdf`-plots of (x,y,z). Optional: Use colours in your plots to indicate for example intensity for the Euclidian distance from one time step n to the next $n+1$.
- 2D `pdf`-plots of (x,y), (x,z) and (y,z). Use colours in your plots to indicate for example intensity for the Euclidian distance from one time step n to the next $n+1$.
- Storing all configuration/system data and all data results in a cross platform format. Put yourself in the situation of having to comply with the reproducible research paradigm – so provide us with all the relevant data and make sure it is all described and possible to figure out what is going on. We do not want to rely on guessing.

You are most welcome to use the Jupyter notebook to document and comment on these examples as long as you ensure that the figures and data are also stored in files.

Documentation

The documentation must include the following:

- Short statement of the problem.

- The core algorithm for the computational part described for example as pseudo-code.
- A description (block diagram or whatever you prefer) to explain the code modules you have planned to implement (ODE solver, 2D- and 3D-plotting, data storage and a tool allowing us to extract the data also, test code and whatever else you see as necessary).
- Your design considerations, e.g., necessary variables and their types. What functionality is needed?
- Test plan, i.e. a description of how you have planned to test your software.

We strongly suggest that the documentation of the code you make is handled via docstrings – e.g. one in the beginning of a module explaining the algorithm and general considerations and one docstring related to describing the functionality of the supplied functions. We also encourage you to use the Jupyter notebook if you find it makes sense. In other words the project can be handled by: 1) the software (incl. docstrings providing the documentation), test files, data files, plots etc.; 2) Jupyter notebooks (or even just script files running the code as long as you provide comments on the results you see for plots, profiling etc.) with the parameter cases mentioned above. We want it to be close to a good working practice you can try out with this small project.

A possible starting point

We have made a suggestion for a basic file and directory structure for this project that you *may* use. You can obtain it by installing `git` and running

```
$ git clone https://git.its.aau.dk/CLAAUDIA/PhD-Python-1.git
```

in the terminal/shell. You can change the structure as you find fit — this is one suggestion and an approach to get you started. You may continue to use this git repository for your project, that is use e.g. `add, commit` etc. but you will not be able to `push`.

Submission

You must deliver a `zip` file including:

- The documentation for the project, 4-8 pages.
- All Python code, including test files etc.
- A `README` file explaining the content of your project, how to use the files etc. Remember: we do not want to guess what is going on. Pay attention to quality.
- Folders including the results for the parameter cases.
- General (and relevant!) information that you find we should have.

Overall we expect that you try to follow the ideas of reproducible research. For example, we expect that executing e.g. `$ python case1.py` generates the figures and results needed for parameter case 1 in its own folder etc. We expect that you to some degree have an idea of the most time-consuming parts of the code (use e.g. `time.time` to get a stopwatch).
Only a few hundred lines of documentation and code should be necessary and it should be doable in a few days. Enjoy – if you get stuck and you have tried handling the problems yourself and asked fellow students then contact us for advice.

Peer Evaluation

We use peer evaulation for this project. This means that each of you will be assigned two other projects to check through and provide feedback on. The Moodle course page will provide a system for managing the peer evaluation process of the submitted projects.
You will all be provided a list of clear criteria to check for in the projects. Peer evaluation will take place in an anonymous fashion so that you cannot see who has evaluated your project.
The course organiser will check the project evaluations and has the final say regarding whether a each project passes the described criteria. Students whose projects do not pass the criteria will be given an opportunity to re-submit the project.

Bibliography

[1] W. A. ADKINS AND M. G. DAVIDSON, *Ordinary Differential Equations*, Springer Science+Business Media, 2012.

[2] J. BERRY AND K. HOUSTON, *Mathematical Modeling*, Elsevier, 1995.

[3] D. BETOUNES, *Differential Equations: Theory and Applications*, Springer Science+Business Media, Second ed., 2010.

[4] D. F. GRIFFITHS AND D. J. HIGHAM, *Numerical Methods for Ordinary Differential Equations – Initial Value Problems*, Springer-Verlag, London, UK, 2010.

[5] E. N. LORENZ, *Deterministic Nonperiodic Flow*, Journal of Atmospheric Sciences, 20 (1963), pp. 130–141.

[6] W. H. PRESS, S. A. TEUKOLSKY, W. T. WETTERLING, AND B. P. FLANNERY, *Numerical Recipes – The Art of Scientific Computing*, Cambridge University Press, third ed., 2007.