handed out: May 19, 2020
handing in: May 28, 2020
presentation/discussion: May 29, 2020

Some results of this exercise are already given in the lecture notes. Here, you should (i) focus on some interesting aspects, (ii) produce a tool – Poincaré section/stroboscope – that is also applicable to other systems, and (iii) learn how to work with problems that take a really long time for computing.
Hints for computationally expensive problems (CPU times longer than some tens of minutes):

- Always start with minimal but complete examples that are then expanded to the full size if everything works. Keep all system parameters, including size, at one place at the beginning of your code.
- Have the code provide a signal on its progress, e.g., by printing a line on the terminal at appropriate intervals of some relevant counter.
- Store the data in a file such that later analysis, plotting,... can be modified without redoing the simulation. For long simulations like calculating large ensembles, already save intermediate results in case your machine or code breaks and implement the corresponding restart option. Keep data files write-protected!

Be aware that computational precision is an issue because some of the trajectories demand rather precise calculations, specifically those for the fragile regime (for instance Figures 4.16-4.20 were done with 10^{-10}) but that these are expensive. If in doubt, do a convergence test on a small representative problem by starting with a low precision (10^{-6}), increase it in steps of 10, and observe the results.

For Python users: Python is a slow language if left to its own devices. For computationally expensive tasks it is therefore advisable to:

- Use **numpy**. The numerical algorithms behind it are written in C and FORTRAN, hence provide top speed.
- Use **numba** (`https://numba.pydata.org`), a just-in-time compiler for Python, for purely numerical tasks. It can be conveniently used with Jupyter via decorators:

```
from numba import jit
@jit
def function(x,y,z)
```

- Use **Cython**, a superset of Python which allows for glueing together C functions and Python code, such that the speed of C can be leveraged by Python. This however, requires some extra coding work and some reading, so use it only for long computations, bifurcation diagrams for instance.
- In general: If speed is a concern, prefer built-in functions over custom code, because the former usually are implemented in C behind the scenes.

1. **Periodically Driven, Viscously Damped Pendulum** present ☐

   Implement the state-forced pendulum using the ODE-solver of your choice. You may either use the 3d autonomous formulation

$$\begin{aligned}\dot{u}_1 &= u_2 \\ \dot{u}_2 &= -2\gamma u_2 - \sin(u_1) + \mu\sin(\omega u_3) \\ \dot{u}_3 &= 1\ ,\end{aligned}$$

   or the corresponding 2d formulation with explicit external forcing

$$\begin{aligned}\dot{u}_1 &= u_2 \\ \dot{u}_2 &= -2\gamma u_2 - \sin(u_1) + \mu\sin(\omega t)\ .\end{aligned}$$

   ODE-solvers work with both forms. [Recall that you already implemented a 2d ODE for Exercise 1 and an externally forced system for Exercise 2.]

   Run a few simulations and plot the phase diagrams $\{u_1(t), u_2(t)\}$ to demonstrate your code. [Let the lecture notes guide you to choose interesting parameters.]

Hint: Plotting trajectories of a pendulum can lead to jumps between $\pm\pi$ and ugly vertical lines. This may be prevented in Python by inserting something like

```
pos = np.where(np.abs(np.diff(data[:,1])) >= 0.5)[0]+1

t = np.insert(data[:,0], pos, np.nan)
x = np.insert(data[:,1], pos, np.nan)
y = np.insert(data[:,2], pos, np.nan)
```

2. **Stability of Trajectories** present ☐

The essential eigenvalues of the system's Jacobian matrix in the u_1u_2-plane are

$$\sigma_\pm = -\gamma \pm \sqrt{\gamma^2 - \cos(u_1)}$$

with corresponding eigenvectors

$$\mathbf{e}_\pm = \begin{pmatrix} -\gamma \mp \sqrt{\gamma^2 - \cos(u_1)} \\ \cos(u_1) \end{pmatrix} .$$

Notice that for $\gamma^2 - \cos(u_1) < 0$ the eigenvalues are complex with $\mathrm{Re}(\sigma) = -\gamma$.

(a) Plot $\sigma_\pm(t)$ along a trajectory that starts at $\mathbf{u} = (0,0,0)$ for (i) the periodic regime, e.g., with $\mu = 0.91$, and (ii) the chaotic regime, e.g., with $\mu = 1.15$ (see Figure 4.6 for phase diagrams).

(b) Study the development of two solutions $\mathbf{u}^a(t)$ and $\mathbf{u}^b(t)$ with slightly different initial conditions. Choose the initial separation (i) along the stable manifold and (ii) along the unstable one. Plot $\varepsilon(t) := |\mathbf{u}^a(t) - \mathbf{u}^b(t)|$ for the two cases.

(c) Calculate the mean maximum and minimum real part of the eigenvalues along the trajectory $\mathbf{u}^a(t)$, hence

$$\overline{\sigma_\pm} := \frac{1}{T}\int_{t_0}^{t_0+T} \mathrm{Re}\big(\sigma_\pm\big(u_1(\tau)\big)\big)\,\mathrm{d}\tau$$

where $u_1(t)$ is the 1-component of $\mathbf{u}^a(t)$ and t_0 is chosen so large that the system has relaxed from its initial state.

Hint: The numerical simulation yields the sequence (t_i, u_{1_i}, u_{2_i}). From this calculate the real value of the eigenvalues $\sigma_\pm$ at each location, denote them by $r_i^\pm := \mathrm{Re}\big(\sigma_\pm(u_{1_i})\big)$, and from there estimate the integral using the trapezoidal rule as

$$\frac{1}{T}\int_{t_0}^{t_0+T} \mathrm{Re}\big(\sigma_\pm\big(u_1(\tau)\big)\big)\,\mathrm{d}\tau \approx \frac{1}{2T}\sum_i [r_{i+1}^\pm + r_i^\pm][t_{i+1} - t_i] .$$

3. **Stroboscope on Pendulum** present ☐

Reproduce the stroboscope view of Fig. 4.9 in the lecture notes. You may want to

(a) zoom into some small part (prescribe a window and only mark transits within it; gets expensive as the window becomes smaller) and

(b) choose different system parameters, even a forcing of the second kind (parameter forcing).

Hint: The timing of the stroboscope flashes must be exact, particularly when zooming in. This may be accomplished along three lines: (i) Choose an appropriate constant time step for the entire simulation. (ii) Choose adaptive time steps and linearly interpolate the state between the two time steps that bracket the flash. (iii) If you have sufficient control over the ODE integrator, you may choose to have an adaptive time step but force them to hit the flash times exactly.

4. **Motion in Fragile Region** present ☐

For $\gamma = 0.1$, $\omega = 0.8$, and $\mu = 0.915$ study the motion and, as far as possible, the nature of the bifurcations (check Figures 4.15 and 4.16 in the lecture notes to understand the setting and the situation). Specifically:

(a) Look at the trajectory $u_1(t)$ during the spinup phase.

(b) Once converged, look at the phase diagram and determine

i. the periodicity (possibly using the stroboscope from above to get the time right) and

ii. the symmetry (to understand if there are multiple attractors as is illustrated by Figures 4.18 and 4.19 for different value of μ).

(c) For understanding the nature of the bifurcation it may be useful to look at phase diagrams before and after the bifurcation. Carefully study the intersections of the trajectory with the positive u_1-axis. It is these points that are plotted in Figure 4.15.

5. **The Last Question** want to discuss ☐

(a) What are the key messages you took home from the lecture?

(b) What are still open questions?

(c) What associated issues did you miss?