Prof. Dr. Gregor Schöner November 18, 2016

**Exercise 3, November 17, 2016 to be handed in November 24!**

This exercise invites you to produce code that numerically approximates the solution of a simple dynamical system. You should use MATLAB for that. MATLAB is freely available for students at RUB (follow instructions at
`http://it-services.ruhr-uni-bochum.de/software/matlab`). You may instead use OCTAVE, which is a free MATLAB clone
(see `http://www.gnu.org/software/octave/`).

Please submit your final version of the source code, one or multiple MATLAB `*.m` files to `jean-stephane.jokeit@ini.ruhr-uni-bochum.de`. Include comments in the code that explain the main steps. Also write a regular report (can be hand-written) that adresses each exercise and includes figures as requested. You can submit this report in paper or electronically.

For those of you, who have no or little experience with programming or MATLAB, a template source code file `simplesimulator.m` is available on the course web page. You can work on the basis of that template. Use the help function of MATLAB to understand function calls (e.g., `help plot`). You are welcome to redo the simulator, create your own, or extend it for your convenience. You may usesimulation tools built into MATLAB like `ODE45` (type `help ODE45` in MATLAB and follow the links).

1. Write a simulator (or use the template) to numerically solve the differential equation

   $$\dot{u} = -\alpha u.$$

   The simulator should allow you to vary the initial value, $u(0)$, the length of the time interval over which you integrate, and the model parameter, $\alpha$. You should also have some control over numerical precision (in the template code, this is the time step of the Euler formula).

2. Use your simulator, run a series of experiments, documenting each through a plot. You may overlay multiple solutions in a plot using `hold on/hold off` in the context of a `figure` command. You will need to store intermediate results in different vectors.

   - Vary the initial condition $u(0)$ to obtain an impression of the flow (you can overlay multiple solutions in a plot using `hold on/hold off` in the `figure`
   - Vary the relaxation rate, $\alpha$.

3. Push the system to the limit of numerical stability. In the Euler procedure of the template, increase the Euler time step (but adjust the number of time steps to cover the same time interval). In other methods you may directly control the largest allowable error of the simulator. Document what happens as you push the limit.

4. Extend the dynamics by adding a negative constant $h < 0$ (resting level) to the right hand side, adding an additive constant, $s$ (input) and a self-excitation term with a sigmoid (the sample code contains a suggestion for such a function in a comment). That self-excitation term should have a multiplicative parameter as well to vary its strength. You may want to plot the right hand side of the equation to get a feeling what it looks like (you can make a vector of values of `variable` using `variablevector=[-5:0.1:5]` and then plot this against the right hand side of the equation evaluated at `variablevector`).

   To probe for a bistable regime of the dynamics, start with a large positive value of activation and contrast that with a large negative initial value. If the dynamics converges to the same value, you are in a monostable regime, if it converges to different values, you are in a bistable regime. Increase $s$ to get into a bistable regime.

   Bonus: Try to demonstrate hysteresis by making time courses of $s(t)$ using the same method for vectors. These should take you through the bistable regime in either direction..

Bonus (for additional credit of 50%) Examine the numerical treatment of noise. For this you need the Euler formulation of the integration step (like in `simplesimulator.m`). Multiply a normally distributed random number `randn` with the square root of the Euler time step and a parameter `noise_strength` and add at each time step.

First simulate the linear dynamics with noise at a reasonable noise strength, a which you can still see the expontential time shape, but also have a visual impression of the level of the induced fluctuations. You may want to extend the time interval to sample noise arond the attractor. Then decrease the Euler time step, adjusting the number of steps to simulate the same time interval. Plot these two solutions on top of each other.

Next modify the code by replacing the square root of the time step with the time step itself. Go back to your original value of the Euler time step, $\Delta t$, and adjust the parameter `noise_strength` so that

$$\sqrt{\Delta t} * \text{noise\_strength}_{\text{old}} = \Delta t * \text{noise\_strength}_{\text{new}}$$

Rerun the simulation: this should yield the same as before. Now again decrease the value of Euler time step, adjusting the number of times steps. Compare these last two simulations and observed if the level of fluctuations has changed.

Bonus (for additional credit of 50%). Back in the original code for stochastic dynamics (with the square root of the time step and the linear dynamics), study the role of noise. Make long simulations, that you start in the attractor ($u = 0$, and observe the level of fluctuations by plotting, but also by computing the standard deviation of the time series (MATLAB function `std`).

- Increase the noise strength and observe how the dynamics becomes more and more stochastic, documenting this by the standard deviation over your time series.
- Keep noise strength constant, but now reduce $\alpha$. What happens to the level of fluctuations?