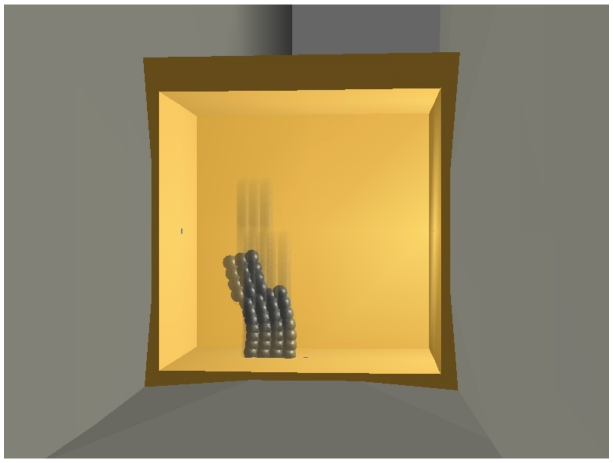
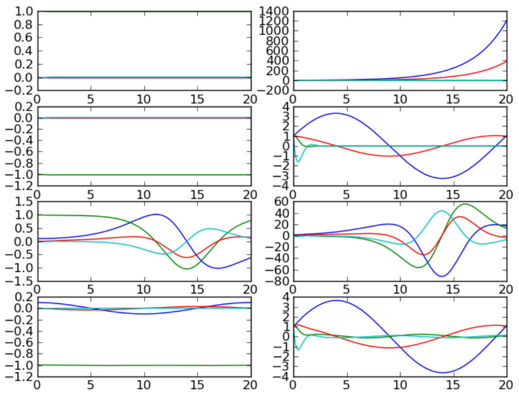
ODE: SYSTEMS, STABILITY AND ERROR CONTROL

JOHAN JANSSON   KATARINA GUSTAVSSON





1. To read

Planetary system (ch. 19)

Particle-Spring Systems (ch. 39)

Stability of solutions to $\dot{u} = f(u)$ (ch 80)

Time Stepping Error Analysis (ch 82)

2. Goals

2.1. Understanding.

- The concept of stability (growth of perturbations) and stability factors.
- A posteriori error control by introducing a dual problem.
- The mass-spring and gravitational model: how to construct systems of many particles.

2.2. Skills.

- Model and simulate large-scale mass-spring systems representing elastic objects and wave propagation

Simulation Technology Module.

- Be familiar with modeling of gravitational and chemical reaction systems.
- Symbolically (with pen and paper) derive a linearized equation for the growth of perturbations.
- Compute stability factors for given ODEs and discuss stability properties.

3. Software Interfaces

Follow the interface specifications in the template Python file in the "python" subdirectory. The file **systems.py** includes the functions (some of them to be finished) needed for this module, continue adding your own code to the functions. When you're finished or want to check how well you're doing, submit your "systems.py" (must be named exactly that) to the Web-CAT system at http://webcat.csc.kth.se.

You will also find a file **main.py** where you can implement the different test cases specified in the questions below.

4. Questions

4.1. Game/Interactive simulation.

Enter the "game" directory and run: `python particle_simulator.py`. The simulation is a time-stepping of a particle-spring model (implemented as the right hand side $f(t, u)$, representing an elastic string or solid elastic bodies.

A simplified version is given in `particle_3body.py` with only 3 particles, where it's very clear how the initial data is set, and where the force is computed for all pairs of particles, rather than just the pairs of particles with springs between them.

Play around with the parameters and the model `f3body(t, u)` and try to understand how it works.

4.2. Systems.

- Modify `f3body(t, u)` to model the gravitational force between a system of particles (see ch. 19). Try to model a planet with an orbiting moon and a comet entering and disrupting the orbit. Copy `f3body(t, u)` into `systems.py` to be able to submit to Web-CAT.
- A system of chemical reactions with four substances; oxidation of NO to NO_2.
  Let $u(t) = [u_1(t), u_2(t), u_3(t), u_4(t)]$ denote the concentration of four substances where u_1 is the concentration of NO, u_2 the concentration of O_2, u_3 the concentration of N_2O_2 and finally u_4 the concentration of NO_2. Then, the following system of ODEs describes the rate of formation of the four substances:
  $$\begin{aligned}\dot{u}_1 &= -2k_{11}u_1^2 + 2k_{12}u_3\\ \dot{u}_2 &= -k_{21}u_3u_2 + k_{22}u_4^2\\ \dot{u}_3 &= k_{11}u_1^2 - k_{12}u_3 - k_{21}u_3u_2 + k_{22}u_4^2\\ \dot{u}_4 &= 2k_{21}u_3u_2 - 2k_{22}u_4^2\end{aligned}$$
  where $k_{11} = 10$, $k_{22} = 10$ (**fast reactions**), $k_{21} = 0.01$ and $k_{22} = 0$ (**slow reactions**). Initally (at t=0) we have that $u_1(0) = 0.5$, $u_2(0) = 1$, $u_3(0) = 0$ and $u_4(0) = 0$. Compute the solution until the substances no longer react with each other and plot the

concentration as a function of time for all four substances. Use your function `solve()` that you implemented in Module 3.

4.3. **Linearization/stability.**

- Derive (with pen and paper) the linearized form of the ODE $du/dt = f(u(t))$ where $f(u(t)) = u(1-u)$. Show (with pen and paper) that this ODE has two stationary solutions, and show that one is stable and one is unstable (to perturbations in initial data). (What does it mean for a solution to be unstable? Think about the solution to the basic models $du/dt = f(u(t))$ where $f(u(t)) = u$ and $f(u(t)) = -u$.)
- Also solve this ODE with your `solve()` function and verify experimentally your pen and paper predictions.

4.4. **Stability/error estimation I.**

- Implement the function `Sd_estimate()`, estimating the growth of perturbations in initial data, using the function `linearization()` which computes df/du (the Jacobian) of the function $f(u(t))$. See Chapter 80 for a definition of S_d.
- Using `Sd_estimate()`, determine the stability of
  1) The normal pendulum
  2) The inverted pendulum

  The pendulum can be modeled as a particle-spring system with two particles, where one particle is fixed. This model is implemented in the function `f_pendulum()`. The normal pendulum has initial data so that the pendulum begins pointing down, whereas the inverted pendulum begins pointing up (and ready to fall down).

4.5. **Stability/error estimation II (hard).**

- Implement the function `Sc_estimate()`, estimating the growth of the residual errors, using the function `linearization()` to formulate the dual problem. See Chapter 82 for a definition of S_c.
- Compute stability factors for end times $T = 2, 4, 8$ for $du/dt = f(t,u)$ where $f(t,u)$ is given by
  1) The harmonic oscillator, $f(t,u) = [u_2(t), -u_1(t)]$
  2) The decaying exponential, $f(t,u) = -u$
  3) The growing exponential, $f(t,u) = u$

  How are the stability factors growing?