Physik auf dem Computer **SS 2013**

Worksheet 11: Solving differential equations

July 3, 2013

General Remarks

- Deadline is **Tuesday, 9th July 2013, 10:00**
- On this worksheet, you can achieve a maximum of 10 points.
- To hand in your solutions, send an email to
  - Olaf (`olenz@icp.uni-stuttgart.de`; Wednesday, 14:00–15:30)
  - Elena (`minina@icp.uni-stuttgart.de`; Wednesday, 15:45–17:15)
  - Tobias (`richter@icp.uni-stuttgart.de`; Friday, 15:45–17:15)
- Attach all required files to the mailing. If asked to write a program, attach the *source code* of the program. If asked for a text, send it as PDF or in the text format. We will *not* accept MS Word files!
- The worksheets are to be solved in groups of two or three people. We will not accept hand-in-exercises that only have a single name on it.
- The tutorials take place in the CIP-Pool of the Institute for Computational Physics (ICP) in Allmandring 3.

Task 11.1 (3 points): Two-dimensional Poisson equation

For this task the two-dimensional Poisson equation

$$\frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} = \rho(x,y)$$

should be solved numerically. Discretize this equation as explained in chapter 8 of the lecture script.

The file `rho.npy` contains a two-dimensional numpy matrix, which represents a charge distribution ρ. You can load it with:

```
rho = numpy.load('rho.npy')
```

To visualize a two-dimensional matrix `matplotlib.pyplot.imshow` can be used. For example `matplotlib.pyplot.imshow(rho,interpolation='nearest')` will plot the charge distribution ρ.

- 11.1.1 (2 points) Implement a Python function `solve_poisson2d(rho, h)` that returns the potential ϕ similar to the function `solve_poisson1d_exact(rho, h)` from task 8.2, where `rho` is a charge distribution and discretization step is `h`.

  Hint

  - You should (re-)read the beginning of chapter 8 from the script carefully.
  - The Python command `rho_new=rho.reshape(N*N)` transforms any $N \times N$-matrix into a one-dimensional array with a length of N^2.

- 11.1.2 (1 point) Use `solve_poisson2d(rho, h)` to solve the two-dimensional Poisson equation for the given charge distribution ρ and create a plot of the potential ϕ with `matplotlib.pyplot.imshow`.

Task 11.2 (7 points): Double pendulum

In this task we consider a double pendulum with masses $m_1 = 1$ and $m_2 = 1$ attached by rigid massless wires of lengths $l_1 = 1$ and $l_2 = 0.5$ as it is shown in Fig. 1. Here ϕ_1 and ϕ_2 are the angles between the wires and y-axis. The forces that act on the masses are $F_1 = -m_1 g$ and $F_2 = -m_2 g$, respectively. For such a system the equation of motion can be written as a system of second order differential equations (1) that is derived from the Lagrangian equations. The trajectory of the mass m_1 just falls on a circle, whereas the trajectory of the second mass m_2 is chaotic. One can obtain the trajectories of both masses by solving the system (1) numerically.

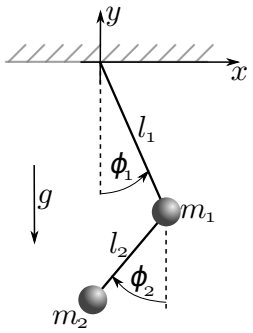

Figure 1: Double Pendulum

$$\begin{cases} M l_1 \ddot{\phi_1} + m_2 l_2 \ddot{\phi_2} \cos(\Delta\phi) + m_2 l_2 \dot{\phi_2}^2 \sin(\Delta\phi) + gM \sin\phi_1 = 0 \\ l_2 \ddot{\phi_2} + l_1 \ddot{\phi_1} \cos(\Delta\phi) - l_1 \dot{\phi_1}^2 \sin(\Delta\phi) + g \sin\phi_2 = 0, \end{cases} \tag{1}$$

where $M = (m_1 + m_2)$ and $\Delta\phi = \phi_1 - \phi_2$. .

- 11.2.1 (1 point) Convert the system (1) to a system of four differential equations of first order which will be suitable for using a Runge-Kutta method, i.e. $\dot{y} = F[t, y(t)]$, where y is the 4-vector $(\phi_1, \dot{\phi_1}, \phi_2, \dot{\phi_2})$. Implement a function `F(t,y)` that takes a 4-vector `y` with angles and velocities and returns a 4-vector with `F(t,y)`.

- 11.2.2 (3 point) Implement a Python function `solve_runge_kutta(F,tmax,y0,h)` that solves the double pendulum for a total time of `tmax` using the 4-th order Runge-Kutta method and returns the angles ϕ_1 and ϕ_2. Here `y0` consists of initial angles $\phi_1(0)$, $\phi_2(0)$ and corresponding initial angular velocities $\dot{\phi_1}(0)$, $\dot{\phi_2}(0)$.

- 11.2.3 (2 point) Implement a Python function `solve_velocity_verlet(f1,f2,x1_in, v1_in,x2_in,v2_in,tmax,h)` that solves the double pendulum for a total time of `tmax` using the Velocity-Verlet method and returns arrays the angles ϕ_1 and ϕ_2. Here `x1_in, x2_in` are the initial angles $\phi_1(0)$, $\phi_2(0)$ and `v1_in, v2_in` are corresponding initial angular velocities $\dot{\phi}_1(0)$, $\dot{\phi}_2(0)$.
- 11.2.5 (1 point) Using both implemented functions solve the system (1) with $h = 0.01$, $t_{max} = 100$ at the following initial conditions: $\phi_1(0) = \pi/2$, $\phi_2(0) = \pi/2$, $\dot{\phi}_1(0) = 0$, $\dot{\phi}_2(0) = 0$, in order to get the trajectory of the second mass m_2 and plot the trajectory in x, y axes.