Übungsblatt 5: Sommersemester 2016

Programmiertechniken

Prof. Dr. L. Pollet J. Greitemann, D. Hügel, J. Nespolo, T. Pfeffer

---

Integration of ordinary differential equations (ODE)

The goal of this worksheet is to develop some numerical methods for the integration of ODEs using templates and the standard template library (STL). In the following we consider the ODE

$$\begin{cases} \frac{dy}{dt} = f(y(t), t), \\ y(t_0) = y_0. \end{cases} \tag{1}$$

Euler method It is the simplest method of integration. By the first order expansion of $y(t)$,

$$y(t) = y_0 + \delta t f(y_0, t_0) + O(\delta t^2). \tag{2}$$

Discretising, we immediately obtain Euler method,

$$y(t_{n+1}) = y(t_n) + f(y(t_n), t_n)\, \delta t, \qquad t_{n+1} = t_n + \delta t. \tag{3}$$

The algorithm is locally $O(\delta t^2)$, hence $O(\delta t)$ globally.

Backward Euler method The previous method can be implemented straightforwardly. However, it may suffer from numerical instability, and diverge exponentially from the exact solution. Consider the coupled equations

$$\begin{cases} \frac{dv}{dt} = F(x(t), v(t), t), \\ \frac{dx}{dt} = v(t). \end{cases} \tag{4}$$

If the force is velocity-independent, then the solution can be stabilised by using the so called *backward difference*

$$v_{n+1} = \frac{x_{n+1} - x_n}{\delta t}. \tag{5}$$

This yields the solution

$$v_{n+1} = v_n + a_n \delta t, \qquad x_{n+1} = x_n + v_{n+1} \delta t. \tag{6}$$

Since this is just a modified Euler algorithm, the error properties do not change.

Runge-Kutta Builds upon Euler method, by carrying out a Taylor expansion to higher orders. We now show the derivation of the second order Runge-Kutta method (RK2). We again start by the Taylor expansion

$$y(t) = y_0 + \delta t f(y_0, t_0) + \frac{\delta t^2}{2} \frac{df}{dt}(y_0, t_0) + O(\delta t^3). \tag{7}$$

By noticing that $df/dt = \partial_t f + f \partial_y f$, we can rewrite the previous expression as

$$y(t) = y_0 + \delta t \left[f(y_0, t_0) + \frac{\delta t}{2} \frac{\partial f}{\partial t}(y_0, t_0) + \frac{\delta y}{2} \frac{\partial f}{\partial y}(y_0, t_0) \right] + O(\delta t^3), \tag{8}$$

where we recognise that the term in square brackets is the Taylor expansion of the function f evaluated at the midpoint, finally yielding, in discretised form,

$$y_{n+1} = y_n + \delta t f(y_{n+1/2}, t_n + \delta t/2) + O(\delta t^3). \tag{9}$$

In practical applications, the midpoint evaluation of f is computed using the Euler method,

$$y_{n+1/2} = y_n + \frac{\delta t}{2} f(y_n, t_n). \tag{10}$$

We report, without proof, the fourth order Runge-Kutta approximation (RK4),

$$k_1 = \delta t\, f(y_n, t_n), \tag{11}$$
$$k_2 = \delta t\, f(y_n + k_1/2, t_n + \delta t/2), \tag{12}$$
$$k_3 = \delta t\, f(y_n + k_2/2, t_n + \delta t/2), \tag{13}$$
$$k_4 = \delta t\, f(y_n + k_3, t_n + \delta t), \tag{14}$$
$$y_{n+1} = y_n + \left[\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right] + O(\delta t^5). \tag{15}$$

1) The one-dimensional harmonic oscillator (HO)

The HO is described by the Hamiltonian

$$H = \frac{p^2}{2} + \frac{k}{2}x^2, \tag{16}$$

where we set the mass of the oscillator to $m = 1$.

(a) Using Hamilton equations, derive the system of two coupled ODE that describe the time evolution of the coordinate x and of its conjugate momentum p.

(b) Write the HO as a C++ class `HarmonicOscillator` that stores the Hamiltonian parameters as well as the value of the conjugate coordinates. Let the class further provide the methods `xdot()` and `pdot()` that return the derivatives $\dot{x}$ and $\dot{p}$, respectively.

(c) Write a routine `Euler` that updates—i.e., evolves—the position and momentum of the HO.

(d) Compute analytically the oscillation period T of the oscillator and set the maximum time for the numerical evolution to $20T$. Evolve—using `Euler`—the HO numerically. Write the analytic solution for $x(t)$ and $p(t)$, and plot them together with the numerical data. Do they agree? Also plot $x[p(t)]$: do the orbits close?

(e) You should have noticed at the previous point that the simple Euler method suffers from stability issues. Implement the backward Euler method `EulerSymplectic()` and the order 2 and order 4 Runge Kutta `RungeKutta()` methods. Evolve the system and plot your results.

(f) (Advanced) Template the `HarmonicOscillator` class so that it can be instantiated in any spatial dimension. Use the STL's `valarray` container to store the coordinates and momenta. Also template the integration routines so that they take as input any class that provides the methods `xdot()` and `pdot()`.

Advanced: You may use iterators or the "ranged for" construct to write dimension-independent code.

Use the resulting program to generate data on the two-dimensional HO. Plot the resulting Lissajous figures using `Python` and `matplotlib`.

2) Kepler's problem

If you still have some time, implement a class that represents the Kepler problem

$$H = \frac{p^2}{2} - \frac{G}{r}\left(1 + \frac{L^2}{r^2}\right). \tag{17}$$

(a) Solve the system initially for $L = 0$.

(b) Then turn on L, which represents the lowest order corrections from general relativity. By plotting the trajectory of the body in the gravity field, you should see the precession of the orbit. This anomalous precession was observed for the planet Mercury in 1859.