Introduction to Computational Physics

Matthias Jamin

(Additional discussion on Friday, 4.7. and Monday, 7.7.)

EXERCISE 8.1: **Eulers Algorithms**

My implementation of the Euler algorithms in Mathematica is given in `Ex8.1.m`, where `SEuler[]`, `MEuler[]` and `IEuler[]` are the simple, modified and improved Euler algorithms respectively. For their arguments see the explicit implementation. I have also written a function `Compare[]` to obtain the relative difference of two solutions. (Note that it only allows to compare solutions with equal step size.) As two examples, I compare all results of the Euler algorithms with the exact result $y = \tan x$ for $h = 0.10$ and $h = 0.05$, and with `Com10` and `Com05`, one can plot the respective comparisons.

EXERCISE 8.2: **Conserved Quantities**

My implementation of this exercise can be found in `Ex8.2.m`. Similarly to exercise 8.1, here `SOszi[]` and `MOszi[]` are the routines corresponding to the simple and modified Euler algorithms respectively. The commands `ComESMx` and `ComESMv` display a comparison of the relative differences of these two solutions with the exact result for both position and velocity.

The algorithms which employ the energy conservation as an additional constraint are given in `SEOszi[]` and `MEOszi[]`. This type of solution fails in the case of the simple Euler algorithm, since both the absolute values of position and velocity become larger than one for certain times, and consequently the square root becomes imaginary, leading to a breakdown of the algorithm. Inspection shows that for the modified Euler algorithm, $|v(t)| < 1$, and thus the procedure works for this case. Nevertheless, the achieved precision is not better than for the modified Euler algorithm without employing the additional information about E.

EXERCISE 8.3: **Runge-Kutta Algorithm**

My implementation of this exercise can be found in `Ex8.3.m`. The respective function for Runge-Kutta with adaptive step size is called `RKA[]`. It is not really fully developed, since it only decreases the step size if the given precision is not matched, but does not increase it if the result is already too precise, thus wasting time. In my program, I start with an initial step size of $h = 2$, to demonstrate that it is decreased to $h = 0.5$ in order to match the precision. According to the structure of the DGL, it can also be solved exactly, with the solution:

$$v(t) \;=\; \sqrt{\frac{mg}{k}}\,\tanh\!\left(\sqrt{\frac{gk}{m}}\,t\right).$$

A numerical comparison of the numerical and exact solution can be obtained with the command `Com`, and plots of the various solutions with `PlotSol`.