Exercise 5. Numerical methods

Šimun Šopar

The goal of Exercise 5 was to use Euler method to solve the harmonic oscillator problem, a very known problem in physics, $m\ddot{x} = -kx$. Analytical solution for this problem is $x = Asin(\omega t) + Bcos(\omega t)$, where A and B can be determined from initial position and velocity, and $\omega^2 = k/m$. This is a great practice example for numerical solving of differential equations because it has an analytical solution and it has a constant of motion, energy is conserved, $E = 0.5m\dot{x}^2 + 0.5kx^2$, so it's easily checkable. In my solution, I chose the inital position to be $x_0 = 1$ and initial velocity is zero, leading to A = 0, B = 1. This is very common in experimental measurement since it is the easiest set-up (its harder to determine initial velocity if it's not zero). C++ code EX5.cpp containts an algorithm for solving this problem numerically, while Jupyter Notebooks contain some graphs comparing numerical and analytical solution for different number of steps.

Euler method and C++ code

Euler method is the simplest method for solving ordinary differential equations with given initial conditions. It constists of discretizing the time scale in some small steps and then, starting from initial time, calculating the value of the function at the next step by exanding it in Taylor series over the previous step. In Euler method, we stop the series at the linear term, its coefficient is the derivative of the function, which is given with differential equation itself, so we don't need to caluclate anything complex. In case of higher order ODEs, like Newtons second law, they can always be written as system of first order ODEs. In that case, we need initial funtion and derivatives for complete solution, so in our case, we require initial position and velocity. If we were given, for example, positions at two different times, then some other method would have to be used, like shooting method.

Main error in Euler method is truncation error, and it is proportional to *h*. For smaller *h* we have, like always, round-off error. For very small *h*, increments in function for two adjecent time steps can be so small that they get neglected, which can then accumulate to a significant error as time goes on. It can also be a problem if the values of derivatives are large, they are multiplied with small *h* which also leads to error. However, in my example I haven't really discover much round-off error, the function isn't that large and so errors in increments are not so obvious. Perhaps if I were to use more steps this error would be greater, however, that would more computation time and, as we will see, for the maximum number of steps I took, the results are quite satisfying.

Euler method is not the most precise one, there are superior ones, however it is the simplest and the quickest (it can even be done by hand relatively easily for simpler examples), so it gives us a quick look in general function properties without always giving us explicitly correct answers, which can be very helpful in later steps of looking for a solution.

C++ code is very simple. It takes number of steps, initial conditions, final time (setting starting time at $t = 0$) and calculates *h* and position, velocity and energy for each time step, which are saved in a three-dimensional array *sol*. For convenience of plotting, it saves the result in text file plot.txt. Results are then copy-pasted to Python Notebook. Easier and less messy solution would be to write Python

algorithm that reads from text file. However, since this code will not only be run on my computer, I did not know how to implement that solution so that it would work on any machine (I don't know where the C++ would save the file, and that is important so Python can open it). I decided to use Jupyter Python .ipynb file instead of regular .py file for two reasons. One is that Jupyter file is devided into cells, so it provided some clearity, since the solution arrays from C++ are quite long. The second is that Jupyter lets you see last saved results without running the program, and the program needed some time to execute in its entirety.

Results, errors and runtime

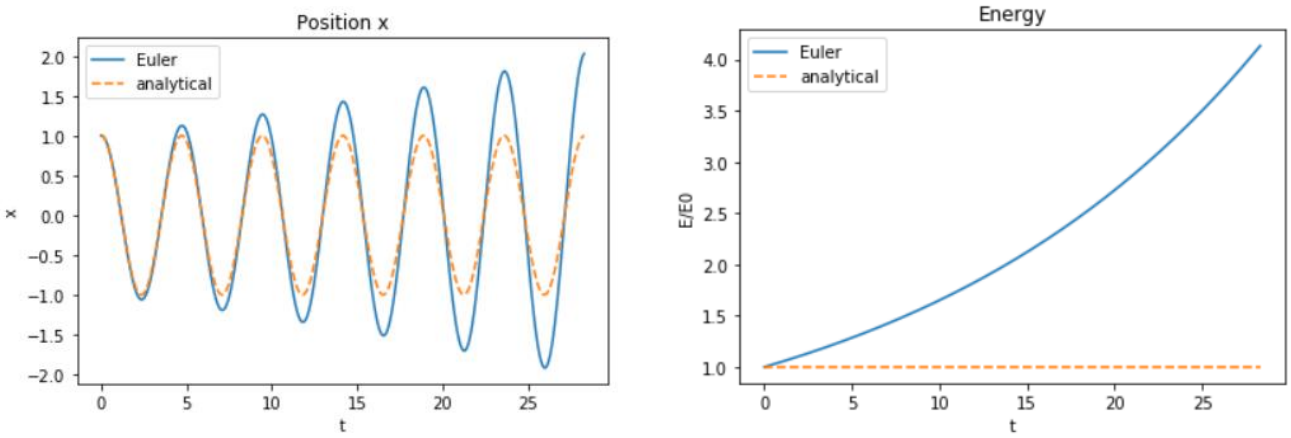
The problem was solved for four values of N (number of steps). Table 1. Shows the time difference in computation times between this values.

N	Time
1000	0s
5000	0s
10000	0.001s
100000	0.006s

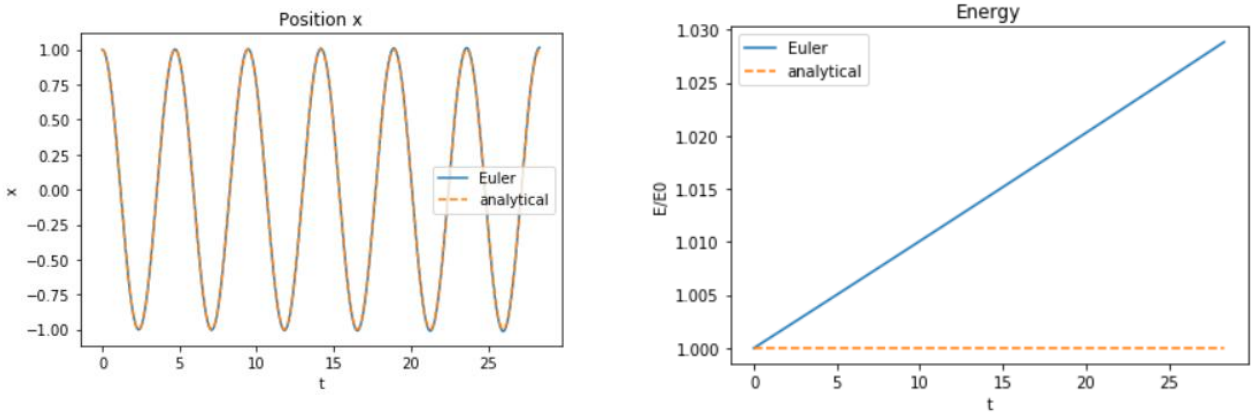
As we can see, there is no difference between N = 1000 and N = 5000, the computation is so fast it cannot be registered. For larger N, the time is not that big, so large N are not the problem for C++ program. It is important to note that computation time depends on the machine the program is run on. Furthermore, it can show different times for the same N on the same machine, so this table is just a rough estimate. Never the less, it is useful, we can see that number of steps doesn't really play a big role in overall execution, so we are not limited to a certain *h*.

In .ipynb files we can see plots for numerical and analytical solution of position, velocity and energy in time dependence. We can see that for low N, the numerical solution starts good, but the amplitude keeps rising, which is a direct cause of truncation error. For every next step, the solution is a bit off, which is not so noticable in the beggining, but as it accumulates, we can see its full effect. For larger time periods, amplitude would rise even higher, going to infinity. This effect is less smaller, but also noticable in larger N. This rise can also be seen in energy graph, numerical energy is slowly rising when it should be constant. This is the inevitable error of Euler method, the method is just too simple to bypass this error.

The following graphs show plots of position and energy for N = 1000 and N = 50000. The rest can be found in Python codes (there are four codes, each one for different number of steps).



Graph 1. Position and energy for N = 1000



Graph 2. Position and energy for N = 50000

As we can see, for N = 50000, we cannot see the difference between analytical and numerical solution in position, however we can see it in energy plot. We can also see that the energy rise in bigger N is much smaller then for smaller N, meaning smaller error for bigger N.

To conclude, we examined Euler method, seen its limitation and abilities. It is a useful tool for first look of a solution, however for more precise solutions, more complex methods should be used.