Kunming School 2018: Lab assignments day 1

Rony Keppens & Jannis Teunissen

You can use any programming language for the excercises below, but we recommend Fortran, as we will also be using Fortran later on.

1 A surprising sequence

You are given the following recurrence relation[1]

$$x_{n+2} = \frac{9}{4}x_{n+1} - \frac{1}{2}x_n,$$

with these starting values

$$x_1 = 1/3$$
$$x_2 = 1/12.$$

The behavior of this recurrence relation is similar to that of (numerically) unstable algorithms.

1. Verify that the exact solution is $x_n = 4^{1-n}/3$
2. Compute x_{60} using double and single precision numbers
3. If possible, compute x_{60} using even higher precision
4. Can you explain what is going on here? Hint: what is the generic solution to the recurrence relation?

2 Harmonic number

The n$^{\text{th}}$ harmonic number H_n is defined as

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}.$$

For sufficiently large n, a good approximation for these numbers is given by

$$H_n \approx \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4}, \tag{1}$$

where $\gamma = 0.57721566490153286060651209$ is the Euler-Mascheroni constant.

[1] This question was taken from Ex. 1.17, Scientific Computing, Heath

1. Compute H_n using double precision (8 byte) floating point numbers, for $n = 2^k$ with $10 < k < 30$ and compare with equation (1).

2. Now compute the numbers again, but reverse the order of your computations. So if before you used e.g. `do i = 1, n`, now use `do i = n, 1, -1` (count downwards). Are the results the same? Which order do you think is more accurate?

3. Repeat these experiments using single precision (4 byte) floating points numbers.

3 A pendulum

We consider a pendulum with a massless, stiff rod for which the ratio of the gravitational acceleration g to the rod lenght l is $g/l = 1\,\mathrm{sec}^{-2}$. The equations of motion for this pendulum in terms of the angle $\theta(t)$ between the rod and the vertical are given by

$$\theta'' + \sin\theta = 0. \tag{2}$$

This second order ODE can be converted to a first order ODE by introducing a new variable $\phi = \theta'$, so that

$$\mathbf{y}' = \begin{pmatrix} \theta' \\ \phi' \end{pmatrix} = \begin{pmatrix} \phi \\ -\sin\theta \end{pmatrix}. \tag{3}$$

1. Make sure you understand the transition from equation (2) to equation (3).

2. Solve the system of equations (3) with the forward Euler method, with the pendulum initially at rest at an angle θ_0. Plot the angle of the pendulum versus time for several time steps (e.g., $h = 10^{-2}$, $h = 2 \times 10^{-2}$, $h = 4 \times 10^{-2}$).

3. Is energy conserved? You can use $\frac{1}{2}\phi^2 - \cos\theta$ as a measure for the energy (do you understand why?).

4. Solve the system with a higher order method, for example a second or fourth order Runge-Kutta scheme. For a small initial amplitude, compare with the approximate solution $\theta \approx \theta_0 \cos(t)$. How well is energy conserved for small and large amplitude oscillations?

For very fast students One way to conserve energy is to use the implicit trapezoidal rule, which is a symmetric combination of the forward and backward Euler methods:

$$y_{n+1} = y_n + \tfrac{1}{2}h\left(f(y_n) + f(y_{n+1})\right).$$

Applied to equation (3), the scheme reads

$$\begin{pmatrix} \theta_{n+1} \\ \phi_{n+1} \end{pmatrix} = \begin{pmatrix} \theta_n \\ \phi_n \end{pmatrix} + \tfrac{1}{2}h \begin{pmatrix} \phi_n + \phi_{n+1} \\ -\sin\theta_n - \sin\theta_{n+1} \end{pmatrix}. \tag{4}$$

Note that these equations are implicit. One way to solve them is to define a function $\mathbf{f}$ which is zero when the equations are satisfied, and then use Newton iterations:

$$\mathbf{q}_{i+1} = \mathbf{q}_i - \mathbf{J}_f^{-1}\mathbf{f}(\mathbf{q}_i),$$

where $\mathbf{q}_i$ is the approximate solution at iteration i and $\mathbf{J}_f^{-1}$ is the inverse of the Jacobian matrix of the function f.

1. Can you think of a reason why the implicit trapezoidal rule is energy conserving? Hint: which symmetry do the original equation have?

2. Write down the Jacobian matrix $\mathbf{J}_f$, and compute its inverse.

3. You now have the ingredients to solve equation (4). If you have time, you can try to implement the solver yourself. Otherwise, have a look at the provided code example, and make sure you understand what the different components do.

4. What kind of tolerance (stop condition) should be used for the Newton iterations?

5. Compare the solution with the implicit trapezoidal rule to those obtained before. Check whether energy is indeed conserved now. Compare errors in θ for the different methods, by making use of a reference solution with a very small time step.