KTH Engineering Sciences

Computer Exercise 1

Ordinary Differential Equations

This computer exercise concerns the numerical solution of the initial value problem for ordinary differential equations (ODEs). The accuracy, stability and computational cost of a few numerical schemes are studied, in particular related to the topics of adaptive (variable) stepsize and stiff problems.

Part 1: Accuracy and stability of a Runge-Kutta method

In this part you will make some numerical experiments with the following Runge-Kutta method:

$$\begin{aligned} k_1 &= f(t_n, u_n), \\ k_2 &= f(t_n + h, u_n + hk_1), \\ k_3 &= f(t_n + h/2, u_n + hk_1/4 + hk_2/4), \\ u_{n+1} &= u_n + \frac{h}{6}(k_1 + k_2 + 4k_3), \quad t_n = nh, \quad n = 0, 1, 2, \ldots \end{aligned}$$

The goal is to apply the method to a linearized Landau–Lifshitz equation for the magnetization vector $\boldsymbol{m}(t) \in \mathbb{R}^3$, given as the ODE system

$$\frac{d\boldsymbol{m}}{dt} = \boldsymbol{a} \times \boldsymbol{m} + \alpha\, \boldsymbol{a} \times (\boldsymbol{a} \times \boldsymbol{m}), \qquad \boldsymbol{m}(0) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Here $\alpha = 0.07$ is a damping parameter and $\boldsymbol{a} = \frac{1}{4}[1, \sqrt{11}, 2]^T$ is a fixed vector.

(a) Implement the Runge–Kutta method and solve the ODE for $t \in [0, T]$ with $T = 50$. Show plots of the components of $\boldsymbol{m}$ as a function t. Also plot the trajectory of $\boldsymbol{m}$ in a three-dimensional plot using Matlab's `plot3` command and `axis equal`. Add the fixed vector $\boldsymbol{a}$ to the plot (using `hold on`) to verify visually that the points $\boldsymbol{m}(t)$ stays in a plane perpendicular to $\boldsymbol{a}$ and that the vector $\boldsymbol{m}(t)$ becomes parallell to $\boldsymbol{a}$ as $t \to \infty$.

(b) Check the order of accuracy of the method as follows. Run the problem with constant stepsizes using $N = 50$, 100, 200, 400, 800 and 1600 steps with t in the interval $[0, T]$, i.e. $h = T/N$. Compute the approximation $\tilde{\boldsymbol{m}}_N(T)$ of $\boldsymbol{m}(T)$ for each N and plot the differences $|\tilde{\boldsymbol{m}}_N(T) - \tilde{\boldsymbol{m}}_{2N}(T)|$ between[1] these approximations as a function of h in a `loglog`-plot. Estimate the order of accuracy from the graph. (See the notes on order of accuracy in Canvas for how (and why) this works. Note that by using differences you do not need the exact solution to find the order of accuracy.)

Hint: Be careful to take the correct number of steps to reach $t = T$. If you get the order of accuracy to be one, there is some mistake in your MATLAB-code!

[1] Here $|\cdot|$ is the Euclidean (2-) norm, $|\boldsymbol{m}| = \sqrt{m_1^2 + m_2^2 + m_3^2}$

(c) Compute the theoretical step size stability limit h_0 for the problem. To do this, first rewrite the system on the standard form

$$\frac{d\boldsymbol{m}}{dt} = A\boldsymbol{m}, \qquad A \in \mathbb{R}^{3\times 3}.$$

Then compute the eigenvalues of A (with MATLAB) and use the stability region for the Runge–Kutta method to find h_0. The stability region is defined as those z in the left half of the complex plane for which $|1 + z + z^2/2 + z^3/6| \leq 1$. (You do not need to derive this.)

Explain how you compute h_0 based on the eigenvalues and the stability region. It will typically involve solving a nonlinear equation numerically. State which one and how you solve it. (You may use MATLAB's built-in functions or you can use your own.)

(d) Verify the step size stability limit empirically by testing different step sizes h and inspecting the results. Plot one stable and one unstable solution.

Part 2: Multistep method for a three-body problem

Consider a satellite that orbits around the earth and the moon. The three bodies form a (moving) plane in which we put a coordinate system, where the x-axis is the line passing through the earth and the moon. The origin is the two bodies' centre of mass. We use the distance between the earth and the moon as our length unit, and let $\mu = 1/82.45$ be the ratio of the moon and earth masses. The earth center is then at $\boldsymbol{r}_0 = (-\mu, 0)^T$, the moon center at $\boldsymbol{r}_1 = (1-\mu, 0)^T$ and the satellite trajectory, denoted $\boldsymbol{r}(t)$, satisfies the following ordinary differential equation,

$$\frac{d^2\boldsymbol{r}}{dt^2} = -(1-\mu)\frac{\boldsymbol{r}-\boldsymbol{r}_0}{|\boldsymbol{r}-\boldsymbol{r}_0|^3} - \mu\frac{\boldsymbol{r}-\boldsymbol{r}_1}{|\boldsymbol{r}-\boldsymbol{r}_1|^3} + 2B\frac{d\boldsymbol{r}}{dt} + \boldsymbol{r}, \qquad B = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},$$

if an appropriate unit of time is used. Here the first two terms are the gravitational forces. The next two are the centrifugal and Coriolis forces induced by the moving coordinate system. Assume the initial data is $\boldsymbol{r}(0) = (1.15,\ 0)^T$ and $\boldsymbol{r}'(0) = (0,\ -0.975)^T$.

(a) Compute the solution of the ODE for $0 \leq t \leq T_{\text{end}} = 10$. Use the Adams–Bashforth 4 method,

$$u_{n+1} = u_n + \frac{h}{24}\left(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right), \qquad f_n = f(t_n, u_n).$$

For the initial values u_1, u_2 and u_3, you can use the Runge–Kutta method from part 1.

Plot the position coordinates $\boldsymbol{r}(t) = (x(t), y(t))$ and velocities $\boldsymbol{r}'(t) = (x'(t), y'(t))$ as a function of t for $t \in [0, T_{\text{end}}]$.

Also plot the trajectory of the satellite in the (x, y)-plane (i.e. $x(t)$ against $y(t)$; do *not* use `plot3` here or in subsequent questions!). Indicate the location of the earth and the moon in the plot. Make sure to use `axis equal`.

Compare solution plots for different time steps h to verify that your solution is qualitatively correct; h must be fairly small.

Hint: You first need to rewrite the second order ODE system to a first order system of four equations. Write it in vector form $\boldsymbol{u}' = \boldsymbol{f}(\boldsymbol{u})$, where you define the vector valued function $\boldsymbol{f}$. Use this form and function also when you program the method. For efficiency, make sure the right hand side $\boldsymbol{f}$ is only evaluated once per time step. (Store f_n in every step and reuse in subsequent steps.)

(b) When solving ODEs the numerical error generally increases with time. For longer times it is therefore difficult to maintain accuracy with lower order methods, and using higher order methods become necessary.

In this part of the exercise you shall compare three different methods and see how they perform for different final times. The methods are Explicit Euler, the Runge–Kutta method from Part 1, and Adams–Bashforth 4.

Solve the three-body problem above with $h = 1/N$. Compare the numerical solution $\tilde{\boldsymbol{r}}$ with the exact solution $\boldsymbol{r}$ after time $t = T_{\text{end}}$. Determine the smallest N (roughly) for which the approximation is within 0.1 of the exact solution, i.e. for which the error $|\tilde{\boldsymbol{r}}(T_{\text{end}}) - \boldsymbol{r}(T_{\text{end}})| < 0.1$.

Accurate values of $\boldsymbol{r}$ for different t are given in the following table.

$\boldsymbol{r}$	$t = 5$	$t = 20$	$t = 40$
$x(t)$	0.4681	-0.2186	-1.4926
$y(t)$	0.6355	-0.2136	-0.3339

Collect your results in the table below. You do not need to report a result if the required N is excessively large.

method	$t = 5$	$t = 20$	$t = 40$
Explicit Euler			
Runge–Kutta 3			
Adams–Bashforth 4			

Comment on the result. Is it as expected? Which method is best to use? (Take into account the relative costs *per time step* of the methods.)

Hint: N needs to be at least 500 for all cases and much much larger for explicit Euler. For $t = 40$ it must be of the order $4000 - 5000$, even for Adams-Bashforth 4. Saving values for plotting may make the code slow. You do not need to do that here. It is enough to compute the final values.

(c) Use the MATLAB built-in function[2] `ode23` to solve system for $0 \leq t \leq T_{\text{end}} = 20$. This is an explicit third order method with adaptive time stepping. Increase the default tolerance by setting `RelTol`=10^{-4} using the `odeset` command[3]. Verify that the solution satisfies the same accuracy as above, $|\tilde{\boldsymbol{r}}(T_{\text{end}}) - \boldsymbol{r}(T_{\text{end}})| < 0.1$.

- Compare the number of time steps used by `ode23` with the number you used by AB4 above.
- How big is the largest and the smallest time step? How do they compare to the time step used by AB4?
- Plot the size of the time steps as a function of time. (If `t` contains the time points you can get the time steps with `h=diff(t)`.) Explain the plot. Why is the time step large in some places and small elsewhere? Where, in the trajectory plot, is it small/large?

[2]There are several IVP-solvers in MATLAB. Use the command `help funfun` to see which are available. To get more information about one of them, say `ode23`, give the command `help ode23`.

[3]It creates on `option` structure that you pass to `ode23` as an argument. See `help odeset` and `help ode23`.

(d) *Voluntary:* Compute the trajectory for longer times and investigate if the system is stable or if the satellite eventually achieves escape velocity and sets off into interstellar space. (It becomes hard to differentiate between numerical and real instabilities here. Increase both `RelTol` and `AbsTol`!) Also feel free to test other initial data, like $\boldsymbol{r}(0) = (1,\ 0)^T, \boldsymbol{r}'(0) = (0,\ -2)^T$ or $\boldsymbol{r}(0) = (1.7,\ 0)^T, \boldsymbol{r}'(0) = (-0.2,\ -1.5)^T$. The satellite orbit can become very complicated!

Part 3: A nonlinear stiff system

The absolute stability of a numerical method for initial value problems is particularly important when we want to solve *stiff* problems. Here we study an ODE-system known as Robertson's problem modeling the reactions of three chemicals A, B and C,

$$A \to B \qquad B + C \to A + C \qquad 2B \to B + C$$

For the three reactions above, let r_1, r_2 and r_3 denote their *rate constants*, i.e. how fast the reactions are. Furthermore, let x_A, x_B and x_C be the (scaled) concentrations of A, B and C respectively. The following system of ODEs then describe the evolution of the concentrations as a function of time t:

$$\begin{aligned}
\frac{dx_A}{dt} &= -r_1 x_A + r_2 x_B x_C, \\
\frac{dx_B}{dt} &= r_1 x_A - r_2 x_B x_C - r_3 x_B^2, \\
\frac{dx_C}{dt} &= r_3 x_B^2.
\end{aligned}$$

The rate constants have the following values: $r_1 = 5.0 \cdot 10^{-2}$, $r_2 = 1.2 \cdot 10^4$ and $r_3 = 4.0 \cdot 10^7$. We are interested in how the concentrations change over the long time interval $t \in [0, 1000]$ when we start from a state with only chemical A present,

$$x_A(0) = 1, \qquad x_B(0) = 0, \qquad x_C(0) = 0.$$

The large disparity between the rate constants introduces time scales in the system of very different size. The fastest changes happen over the order of $O(10^{-3})$ time units and the slowest over the order of $O(10^3)$.

(a) Start by solving the problem with the explicit Runge–Kutta (RK) method given in Part 1. Since the problem is stiff the stepsize h has to be very small to avoid numerical instability. Therefore, start by solving it for the shorter time $t \in [0, 10]$ and find (empirically) how small h you need to take for the solution to be stable. Report this value. Plot the solutions of x_A, x_B and x_C as a function of $t \in [0, 10]$, both with `plot` (linear scale) and `semilogy` (log scale).

(b) The stability properties of the numerical method are given by the eigenvalues of the Jacobi matrix for the right hand side. Compute those eigenvalues for your numerical solution in every timestep and plot them as a function of $t \in [0, 10]$. There are three eigenvalues. One is always zero. It is enough to plot the other two.

The largest eigenvalue (in magnitude) should determine the stepsize limit for the RK method. Verify that this is (roughly) true[4] by comparing with your empirical finding above. Note that the stability limit depends on the solution itself since the problem is nonlinear. It will therefore change in time. Does it get more or less severe as t increases?

[4]The transition between unstable and stable solutions is not as clear-cut here as in Part 1.

(c) Compute the solution for $t \in [0, 1000]$ with the RK method. Keep in mind that the stability limit will be slightly different for this case than for the case $t \in [0, 10]$. Report which h you used, the final values of x_A, x_B, x_C at $t = 1000$, and how long time it took on your computer. (You can use Matlab's `tic` and `toc` commands to time it.)

Hint: To make the computations faster, do not save the solution (for plotting) or compute eigenvalues. To figure out the $T = 1000$ case it may be good to first do some simpler cases like $T = 20$, $T = 50$, $T = 100$, ...

(d) Use the Implicit Euler (IE) method to solve the problem. It is implemented in the MATLAB function file `impeuler.m` available on Canvas. Learn how to use it with `help impeuler`. Also look through the code and make sure you understand it.

Verify that IE does not have a stability limit by checking empirically that "any" timestep h gives a stable solution. Choose h so that you get, visually, the same solution for $t \in [0, 10]$ as with the RK method. (You can use a rather large h.) Then run for $t \in [0, 1000]$ with the same h and plot the solution.

(e) Compare the efficiency of the two methods by measuring how long time (using `tic/toc`) it takes to compute a solution to $t = 1000$ with different accuracies. Make a table of the form:

method	h	error	computational time
RK			
IE			
IE			
$\vdots$			

Include at least four rows for IE with h of different magnitude.

Compute the errors (in Euclidan norm as before) at $t = 1000$ by using the following accurate values:

$$x_A(1000) \approx 0.293414227164,$$
$$x_B(1000) \approx 0.000001716342048,$$
$$x_C(1000) \approx 0.706584056494.$$

Also, modify the code in `impeuler.m` here so that it does not save the solution in every step, to get a fair timing comparison.

Comment on and discuss the results. In particular, answer the questions below:

- What are the factors that influence the timings and the errors of the methods?
- Why is RK faster than IE when very accurate solutions are sought, but not otherwise?
- For which desired error levels (roughly) is IE faster than RK?
- How would the comparison change if the same experiments were made for the problem in Part 1 above?