S. Mishra
K. O. Lye
L. Scarabosio

Autumn Term 2015

Computational Methods for Engineering Applications II

ETH Zürich
D-MATH

Homework Problem Sheet 2

This assignment sheet contains some tasks marked as **Core problems**. If you hand them in (see deadline at the end of the problem sheet), these tasks will be corrected. Full mark for the total of the core problems in all assignments will give a 20% bonus on the total points in the final exam. This is really a bonus, which means that at the exam you can still get the highest grade without having the bonus points (of course then you need to score more points at the exam).

The total number of points for the Core problems of this sheet is **5 points**. (The total number of points over all assignments will be around 20).

Problem 2.1 Heun's Method for Time Stepping

In this exercise, we consider Heun's method for time stepping, a particular Runge-Kutta method. The *Butcher tableau* for this scheme is

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} . \tag{2.1.1}$$

(2.1a) Is Heun's method an implicit or an explicit scheme? How can you see it?

(2.1b) Consider the scalar ODE

$$u'(t) = f(t,u), \quad t \in (0,T), \tag{2.1.2}$$

for some $T > 0$.

Let us denote the time step by Δt and the time levels by $t^n = n\Delta t$ for $n = 0, 1, 2, \ldots, \frac{T}{\Delta t}$. Formulate Heun's method, i.e. write down how to perform the time stepping from $u_n \approx u(t^n)$ to $u_{n+1} \approx u(t^{n+1})$ for $n = 0, 1, 2, \ldots, \frac{T}{\Delta t} - 1$.

(2.1c) Show that Heun's method is *consistent*.

HINT: Check that the consistency conditions for Runge-Kutta methods seen in class hold.

(2.1d) Show that Heun's method is a *second* order method. We recall that a time stepping method is of order $k \in \mathbb{N}$ if we obtain a truncation error that is $\mathcal{O}(\Delta t^k)$ when inserting the exact solution $u(t)$ in the consistent form of the method (e.g. $u(t^n)$ instead of u_n).

(2.1e) We have seen in the lecture that the concept of convergence is necessary for a time stepping method to give accurate results, but it's not sufficient. Due to this, we introduced the concept of stability and in particular of *absolute stability* (*A-stability*).

We recall that to study the A-stability of a method, one considers the numerical method applied to the ODE

$$u'(t) = \lambda u(t), \quad t \in (0, +\infty), \tag{2.1.3}$$
$$u(0) = 0, \tag{2.1.4}$$

for $\lambda \in \mathbb{C}$ (with the primary focus on $\operatorname{Re}\lambda < 0$), and analyses for which values of $\lambda\Delta t \in \mathbb{C}$ it holds that u_n remains bounded as $n \to \infty$, or, in other words, that $\frac{|u_{n+1}|}{|u_n|} \leq 1$, $n \in \mathbb{N}$. This analysis allows to identify the so-called *stability region* in the complex plane. (We suggest you to revise the lecture material to recall why studying A-stability for (2.1.3) is sufficient also for A-stability of linear systems of equations.)

Determine the inequality that the quantity $w := \lambda\Delta t$ has to satisfy so that Heun's method is A-stable. Solve the forementioned inequality for $\lambda \in \mathbb{R}$ and draw the restriction of the stability region on the real line.

From now on, we consider a particular case of (2.1.2), with some initial conditions. Namely, we take

$$u'(t) = e^{-t} - u(t), \quad t \in (0, T), \tag{2.1.5}$$
$$u(0) = u_0. \tag{2.1.6}$$

(2.1f) **(Core problem)** Complete the template file `heun.cpp` provided in the handout, implementing the function `Heun` to compute the solution to (2.1.5) up to the time $T > 0$. The input arguments are:

- The initial condition u_0.
- The step size Δt, in the template called `dt`.
- The final time T, which we assume to be a multiple of Δt.

In output, the function returns the vectors `u` and `time`, where the i-th entry contains, respectively, the solution u and the time t at the i-th iteration, $i = 0, \ldots, \frac{T}{\Delta t}$. The size of the output vectors has to be initialized inside the function according to the number of time steps.

(2.1g) Using the code from subproblem (2.1f), plot the solution to (2.1.5) for $u_0 = 0$, $\Delta t = 0.2$ and $T = 10$. Note that the function `main` is already implemented in the template.

(2.1h) According to the discussion in subproblem (2.1e), which is the biggest timestep $\Delta t > 0$ for which Heun's method is stable?

(2.1i) **(Core problem)** Make a copy of the file `heun.cpp` and call it `heunconv.cpp`. Modify the file `heunconv.cpp` to perform a convergence study for the solution to (2.1.5) computed using Heun's method, with $u_0 = 0$ and $T = 10$. More precisely, consider the sequence of timesteps $\Delta t_k = 2^{-k}$, $k = 1, \ldots, 8$, and for each of them, compute the numerical solution $u_{\frac{T}{\Delta t_k}} \approx u(T)$ and the error $|u_{\frac{T}{\Delta t_k}} - u(T)|$, where u denotes the exact solution to (2.1.5). Produce a double logarithmic plot of the error versus Δt_k, $k = 1, \ldots, 8$. Which rate of convergence do you observe?

HINT: The exact solution to (2.1.5) is $u(t) = te^{-t}$, $t \in [0, T]$.

Problem 2.2 Chemical reactions and DIRK

We will visit the chemical reactions once more. We consider a chemical reaction

$$A \xrightarrow{k_1} B \xrightarrow{k_2} C$$

and the reactions are given by the following ODE

$$\vec{u}'(t) = \begin{pmatrix} u_1'(t) \\ u_2'(t) \\ u_3'(t) \end{pmatrix} = A\vec{u}(t), \tag{2.2.1}$$

where u_1, u_2 and u_3 denotes the concentration of A, B and C respectively, and

$$A = \begin{pmatrix} -k_1 & 0 & 0 \\ k_1 & -k_2 & 0 \\ 0 & k_2 & 0 \end{pmatrix},$$

where k_1 and k_2 are constants. In this exercise we set[1]

$$k_1 = 10^4 \qquad k_2 = 1.$$

We set the initial concentrations as

$$\vec{u}_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

(2.2a) Compute the exact solution $\vec{u}(t)$. Plot $u_1(t), u_2(t)$ and $u_3(t)$ for $t \in [0, 1]$.

(2.2b) We have implemented a Forward-Euler solver in template_code2/chemical/forward_euler.cpp. Modify and run the program with the following number of timesteps:

$$N_1 = 10^2$$
$$N_2 = 5 \cdot 10^3$$
$$N_3 = 10^4.$$

Plot the solution for the various N and compare against the exact solution. What you observe?
HINT: You only have to edit the following line in main that sets `N`:

```
int N = 1e4;
```

or you can run the program with a command line argumetn

```
./forward_euler 1e4
```

[1] If you have already read section 5.7 in the script, you may observe that k_1 is different here. The sole reason for this is to make the numerical experiments faster.

(2.2c) In the exercise above, we saw that we need a high number of timesteps in order to get anything close to the exact solution. In this exercise, we will test a Diagonally Implicit Runge-Kutta (DIRK) method.

Recall that the 3-stage, second order accurate DIRK method to solve the system

$$\vec{v}'(t) = F(\vec{v}(t), t)$$

is given as

$$\begin{aligned}
\vec{y}_1 =& \vec{v}_n \\
\vec{y}_2 =& \vec{v}_n + \frac{\Delta t}{4}\left[F(\vec{y}_1, t^n) + F\left(\vec{y}_2, t^n + \frac{\Delta t}{2}\right)\right] \\
\vec{v}_{n+1} =& \vec{v}_n + \frac{\Delta t}{3}\left[F(\vec{y}_1, t^n) + F\left(\vec{y}_2, t^n + \frac{\Delta t}{2}\right) + F(\vec{v}_{n+1}, t^n + \Delta t)\right].
\end{aligned}$$

Write down the linear equation for y_2 and u_{n+2} for the 3-stage DIRK method for (2.2.1) in the following form

$$\begin{aligned}
B\vec{y}_2 &= \vec{b} \\
C\vec{u}_{n+1} &= \vec{c}
\end{aligned}$$

for some matrices B and C and some vectors $\vec{b}$ and $\vec{c}$.

HINT: You do not have to solve the system linear systems by hand!

(2.2d) **(Core problem)** Write a C++ program that implements the DIRK method. Compute the solution up to $T = 1$ and for the following number of timesteps

$$\begin{aligned}
N_1 &= 10^2 \\
N_2 &= 5 \cdot 10^3 \\
N_3 &= 10^4.
\end{aligned}$$

Plot the solution for the different simulations. How does this compare against the results using Forward-Euler?

See template_code2/chemical/dirk.cpp for a template.

Problem 2.3 Multistep methods

We are given the ODE

$$y''(t) + 5y'(t) + 6y(t) = 0 \tag{2.3.1}$$

with initial values

$$y(0) = 1 \qquad y'(0) = 2.$$

(2.3a) Write (2.3.1) in the form

$$\vec{u}'(t) = A\vec{u}(t).$$

(2.3b) Compute the analytic solution (2.3.1).

(2.3c) Write a C++-program that solves (2.3.1) using the Forward-Euler method. Plot the numerical solution up to $T = 1$ for $\Delta t = 2^{-12}$.

Furthermore, compute the difference

$$|u_N - u(T)|$$

between the numerical solution u_N and the analytical solution u (computed in b)) at time $T = 1$ for

$$\Delta t = \frac{1}{2^k} \qquad \text{for } k = 5, \dots, 12.$$

Plot the difference-versus-resolution in a log-scale plot (use `loglog` to plot).

See template_code/multistep/forward_euler.cpp for a template.

(2.3d) Recall that the second order Adam-Bashforth method is given as

$$u_{n+2} = u_{n+1} + \frac{\Delta t}{2}(-f(u_n) + 3f(u_{n+1})).$$

We will approximate the start value u_1 using Forward-Euler, that is

$$u_1 = u_0 + \Delta t f(u_0).$$

Implement the second order Adam-Bashforth method in a C++-program. Plot the solution up to time $T = 1$. Furthermore, compute the difference

$$|u_N - u(T)|$$

between the numerical solution u_N and the analytical solution u (computed in b)) at time $T = 1$ for

$$\Delta t = \frac{1}{2^k} \qquad \text{for } k = 5, \dots, 12.$$

Plot the difference-versus-resolution in a log-scale plot (use `loglog` to plot). How does this compare to the Forward-Euler method?

See template_code/multistep/forward_euler.cpp for a template.

Problem 2.4 Multiple Choice: Comparing different time stepping methods

We consider four kinds of time stepping schemes, namely explicit Runge-Kutta methods, implicit Runge-Kutta methods, explicit multistep methods and implicit multistep methods. We want to compare these methods highlighting the advantages and disadvantages of each of them. Suppose that each of these methods is applied to the scalar ODE $u'(t) = f(t, u(t))$, $u(0) = u_0$, where f is supposed to be nonlinear.

(2.4a) How many function evaluations of f are required to perform one time step?

- Explicit q-stage Runge-Kutta:

  (i) 1 (ii) q (iii) it depends on f.

- Implicit q-stage Runge-Kutta:
  (i) 1 (ii) q (iii) it depends on f.
- Explicit multistep with q steps:
  (i) 1 (ii) q (iii) it depends on f.
- Implicit multistep with q steps:
  (i) 1 (ii) q (iii) it depends on f.

(2.4b) Does the method need other initial conditions apart from $u(0) = u_0$?

- Explicit q-stage Runge-Kutta:
  (i) yes (ii) no.
- Implicit q-stage Runge-Kutta:
  (i) yes (ii) no.
- Explicit multistep with q steps:
  (i) yes (ii) no.
- Implicit multistep with q steps:
  (i) yes (ii) no.

(2.4c) Is it easy to implement an adaptive time stepping (i.e. Δt not the same for all time steps)?

- Explicit q-stage Runge-Kutta:
  (i) yes (ii) no.
- Implicit q-stage Runge-Kutta:
  (i) yes (ii) no.
- Explicit multistep with q steps:
  (i) yes (ii) no.
- Implicit multistep with q steps:
  (i) yes (ii) no.

(2.4d) Are there methods of this kind that are (unconditionally) A-stable?

- Explicit q-stage Runge-Kutta:
  (i) yes (ii) no.
- Implicit q-stage Runge-Kutta:
  (i) yes (ii) no.

- Explicit multistep with q steps:

  (i) yes (ii) no.

- Implicit multistep with q steps:

  (i) yes (ii) no.

(2.4e) All implicit Runge-Kutta methods are unconditionally A-stable:

(i) true (ii) false.

(2.4f) All unconditionally A-stable Runge-Kutta methods are implicit:

(i) true (ii) false.

Published on October 14.

To be submitted on November 3.

Last modified on October 20, 2015