Series 2

Computational Methods for Engineering Applications
Last edited: October 14, 2020
Due date: October 24 at 23:59

Template codes are available on the course's webpage at `https://moodle-app2.let.ethz.ch/course/view.php?id=13412`.

Exercise 1 Strong Stability Preserving RK3 Method for Time Stepping

In this exercise, we consider the Strong Stability Preserving, Runge-Kutta 3 method for time stepping, also known as SSPRK3, or the Shu-Osher method. The *Butcher tableau* for this scheme is:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1/2 & 1/4 & 1/4 & 0 \\ \hline & 1/6 & 1/6 & 2/3 \end{array} \tag{1}$$

1a)

Is the SSPRK3 method an implicit or an explicit scheme? How can you see it?

1b)

Consider the scalar ODE

$$u'(t) = f(t,u), \quad t \in (0,T), \tag{2}$$

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 explicitly the SSPRK3 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 $n = 0, 1, 2, \ldots, \frac{T}{\Delta t} - 1$.

1c)

Show that SSPRK3 is *consistent*, and that it is *at least* third order accurate (i.e. its one-step error is fourth order).

Hint: Proving this by hand (using Taylor) is difficult; you can find a simpler criterion in the lecture notes.

1d)

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. One approach to determine stability of a method is *A-stability*.

To study the A-stability of a method, one considers the numerical method applied to the ODE

$$u'(t) = \lambda u(t), \quad t \in (0, +\infty), \tag{3}$$
$$u(0) = 0, \tag{4}$$

for $\lambda \in \mathbb{C}$ (with the primary focus on $\mathrm{Re}(\lambda) < 0$), 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 (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 SSPRK3 is stable. Solve the aforementioned 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), with some initial conditions. Namely, we take

$$u'(t) = e^{-2t} - 2u(t), \quad t \in (0, T), \tag{5}$$
$$u(0) = u_0. \tag{6}$$

1e)

Complete the template file `sspk3.cpp` provided in the handout, implementing the function `SSPRK3` to compute the solution to (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.

1f)

Using the code from subproblem **1e)**, plot the solution to (5) for $u_0 = 0$, $\Delta t = 0.2$ and $T = 10$. Note that the function `main` is already implemented in the template.

1g)

According to the discussion in subproblem **1d)**, which is the biggest timestep $\Delta t > 0$ for which SSPRK3 is stable for problem (5)?

1h)

Make a copy of the file `ssprk3.cpp` and call it `ssprk3conv.cpp`. Modify the file `ssprk3conv.cpp` to perform a convergence study for the solution to (5) computed using SSPRK3, 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 (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 (5) is $u(t) = (t + u_0)e^{-2t}$, $t \in [0, T]$.

Exercise 2 Multistep methods

We are given the ODE

$$y''(t) + 5y'(t) + 6y(t) = 0 \tag{7}$$

with initial values

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

Using the diagonalization approach (see Assignment 1), it is easy to show that the solution to this problem is

$$y(t) = 5\exp(-2t) - 4\exp(-3t)$$

2a)

Write (7) in the form

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

2b)

Write a C++-program that solves (7) using the Forward-Euler method. Plot the numerical solution up to $T = 1$ for $\Delta t = 2^{-12}$.

Furthermore, compute the difference

$$|u_N - y(T)|$$

between the numerical solution u_N and the analytical solution y) at time $T = 1$ for

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

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

See series2_handout/multistep/forward_euler.cpp for a template.

2c)

The second order Adam-Bashforth method is given as

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

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

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

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

See series2_handout/multistep/forward_euler.cpp for a template.

Exercise 3 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.

3a)

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.

3b)

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.

3c)

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.

3d)

Are there methods of this kind that are 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.

3e)

All implicit Runge-Kutta methods are A-stable:

(i) true (ii) false.

3f)

All A-stable Runge-Kutta methods are implicit:

(i) true (ii) false.