NumCSE exercise sheet 6
Initial Value Problems and Non-Linear Equations

soumil.gurjar@sam.math.ethz.ch
oliver.rietmann@sam.math.ethz.ch

December 13, 2018

Exercise 6.1. *Explicit vs. Implicit Time Stepping.*

The universal oscillator equation with no forcing term is given by:

$$\ddot{x} + 2\zeta\dot{x} + x = 0, \quad t \in (0, T), \tag{1}$$

for $T > 0$. In (1), $x : \mathbb{R}^+ \to \mathbb{R}$ denotes the position of the oscillator, and $\dot{x}$ its velocity. The real parameter $\zeta > 0$ determines the damping behavior in the transient regime. For $\zeta > 1$ we have overdamping, for $\zeta = 1$ the so-called critical damping and for $\zeta < 1$ underdamping. In this exercise we consider the case $\zeta < 1$, $\zeta \neq 0$.
As initial conditions we impose

$$x(0) = x_0, \ \dot{x}(0) = v_0. \tag{2}$$

(a) Equations (1) and (2) can be rewritten as a linear system of first order differential equations with appropriate initial conditions, i.e.:

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} \tag{3}$$
$$\mathbf{y}(0) = \mathbf{y}_0. \tag{4}$$

Specify $\mathbf{A} \in \mathbb{R}^{2\times 2}$ and $\mathbf{y}, \mathbf{y}_0 \in \mathbb{R}^2$.

(b) Compute the solution to (1) with initial conditions given by (2) with

$$x_0 = 1, \ v_0 = 0.$$

Hint: Diagonalizing $\mathbf{A}$ is the key to finding an analytic solution. Recall that $\zeta < 1$ and that, for a real number α, it holds that $e^{\alpha i} = \cos\alpha + i\sin\alpha$, where $i = \sqrt{-1}$ denotes the imaginary unit.

(c) Recall the explicit Euler timestepping introduced in the lecture. Using `template_harmonic_oscill.cpp`, implement the function `explicitEuler` to compute the solution $\mathbf{y} = \mathbf{y}(t)$ to (3) up to the time $T > 0$. The function should take as input the following parameters:

- The initial position x_0 and initial velocity v_0, stored in the 2×1 vector `y0`.
- The damping parameter ζ.
- The step size h.
- The final time T, that we assume to be a multiple of h.

In output, the function returns the vectors `y1`, `y2` and `time`, where the i-th entry contains the particle position, the particle velocity, and the time, respectively, at the i-th iteration, $i = 1, \ldots, \frac{T}{h}$. The size of the output vectors has to be initialized inside the function according to the number of time steps.

(d) Recall the implicit Euler timestepping introduced in the lecture. Using `template_harmonic_oscill.cpp` provided in the handout, implement the function `implicitEuler` to compute the solution $\mathbf{y} = \mathbf{y}(t)$ to (3) up to the time $T > 0$. The input and output parameters are as in the function `explicitEuler` from the last subproblem.

(e) In `template_harmonic_oscill.cpp`, complete the function `Energy` that, given in input a vector containing velocities at different time steps, returns the kinetic energy $E(t) = \frac{1}{2}v^2(t)$, where $v(t)$ denotes the velocity of the particle at time t.

(f) We consider two time steps $h_1 = 0.1$ and $h_2 = 0.5$. We choose $T = 20$, $\zeta = 0.2$.

Using the `main` already implemented in `template_harmonic_oscill.cpp`, plot the positions and the energies obtained with the explicit Euler time stepping and the implicit Euler for the two choices of time steps. For the position, plot the exact solution from subproblem (b), too. What do you observe?

Exercise 6.2. *Heun's method.*

Heun's method is the Runge-Kutta method defined by the *Butcher tableau*

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

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

(b) Fix $t_0, T > 0$ and $y_0 \in \mathbb{R}$ and let $f : [t_0, T] \times \mathbb{R} \to \mathbb{R}$. Moreover, let $y : [t_0, T] \to \mathbb{R}$ solve the ODE

$$\begin{aligned} y'(t) &= f(t, y(t)), \quad t \in [t_0, T], \\ y(t_0) &= y_0. \end{aligned} \tag{5}$$

Fix $N \in \mathbb{N}$ and consider the time grid $t_0 < t_1 < \ldots < t_N = T$. For simplicity, we assume equidistant time steps, i.e. $h := t_{k+1} - t_k$ is the same for all $k \in \{0, \ldots, N-1\}$. Let $y_1, \ldots, y_N \in \mathbb{R}$ denote the approximations $y_k \approx y(t_k)$ according to Heun's method. For fixed $k \in \{0, \ldots, N-1\}$, give an expression to compute y_{k+1} from t_k, t_{k+1} and y_k.

(c) We keep the previous task. Implement a function

```
std::vector<double> Heun(const std::function<double(double, double)> &f,
                         const std::vector<double> &t, double y0);
```

that takes the quantities f and y_0 in Equation (5), and a time grid $t_0 < t_1 < \ldots < t_N$. It shall return a vector containing the approximations $y_0, \ldots, y_N$.

(d) Show that Heun's method is consistent.

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

(e) Compute the stability function $S : \mathbb{C} \to \mathbb{C}$ and the stability region of Heun's method.

(f) From now on, we consider the special case of Equation (5) given by

$$\begin{aligned} y'(t) &= e^{-2t} - 2y(t), \quad t \in [0, T], \\ y(0) &= y_0. \end{aligned} \tag{6}$$

Which is the biggest time step $h > 0$ for which Heun's method is stable for the ODE (6)?

Hint: Consider the homogeneous version of (6) and use the stability region computed in the previous exercise.

(g) Compute the solution of Equation (6) on $[0, 1]$ (i.e. $T = 1$) using Heun's method with step size $h_j := 2^{-(j+2)}$ for $j = 0, 1, 2, 3$. In all four cases, compute the error to the exact solution $y(t) = te^{-2t}$ at time $t = 1$. Use this data to determine the order of convergence (e.g. by a log-log plot).

Exercise 6.3. *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:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\ \hline & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{array} \tag{7}$$

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

(b) Consider the scalar ODE

$$u'(t) = f(t,u), \quad t \in (0,T), \tag{8}$$

for some $T > 0$.

Let us denote the time step by h and the time levels by $t^n = nh$ for $n = 0, 1, 2, \ldots, \frac{T}{h}$. 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 $n = 0, 1, 2, \ldots, \frac{T}{h} - 1$.

(c) 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 second and third order accuracy of an explicit, consistent, s-stage RK method by hand (using Taylor) is tedious; a simpler criteria is to check if the following holds:

$$\sum_{j=1}^{s} b_j c_j = \frac{1}{2} \text{ (for second order);}$$

$$\sum_{j=1}^{s} b_j c_j^2 = \frac{1}{3} \quad \text{and} \quad \sum_{j=1}^{s}\sum_{i=1}^{s} b_i a_{ij} c_j = \frac{1}{6} \text{ (for third order).}$$

(d) 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 *absolute stability*.

Determine the inequality that the quantity $w := \lambda h$ has to satisfy so that SSPRK3 is absolutely stable.

(e) From now on, we consider a particular case of (8), with some initial conditions. Namely, we take

$$u'(t) = e^{-2t} - 2u(t), \quad t \in (0,T), \tag{9}$$

$$u(0) = u_0. \tag{10}$$

Complete the template file `template_ssprk3.cpp` provided in the handout, implementing the function `SSPRK3` to compute the solution to (9) up to the time $T > 0$. The input arguments are:

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

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, $i = 0, \ldots, \frac{T}{h}$. The size of the output vectors has to be initialized inside the function according to the number of time steps.

Compute the solution to (9) at each time-step for $u_0 = 0$, $h = 0.2$ and $T = 10$ and plot it using the matlab or python script that has been provided.

(f) According to the discussion in subproblem (d), which is the biggest timestep $h > 0$ for which SSPRK3 is stable for problem (9)?

Hint: Focus on the homogeneous part of (9) for stability considerations.

(g) Create a copy of the completed file `template_ssprk3.cpp` and name it `template_ssprk3conv.cpp`. Modify the `main()` function to perform a convergence study for the solution to (9) computed using SSPRK3, with $u_0 = 0$ and $T = 10$. More precisely, consider the sequence of timesteps $h_k = 2^{-k}$, $k = 1, \ldots, 8$, and for each of them, compute the numerical solution $u_{\frac{T}{h_k}} \approx u(T)$ and the error $|u_{\frac{T}{h_k}} - u(T)|$, where u denotes the exact solution to (9). Produce a double logarithmic plot of the error versus h_k, $k = 1, \ldots, 8$ using the matlab or python script that has been provided. What rate of convergence do you observe?

Hint: The exact solution to (9) is $u(t) = (t + u_0)e^{-2t}$, $t \in [0, T]$.

Exercise 6.4. *Newton's method.*

Consider the following system of non-linear equations

$$\begin{aligned} f_0(x_0, x_1) &:= x_0^2 + 2x_1^2 - 1 = 0, \\ f_1(x_0, x_1) &:= x_1 - x_0^2 = 0, \end{aligned} \tag{11}$$

where $x_0, x_1 \in \mathbb{R}$. As shown in Figure 1 below, this system admits exactly two real solutions. One easily verifies that they are given by

$$x_0 = \pm\frac{1}{\sqrt{2}} \quad \text{and} \quad x_1 = \frac{1}{2}.$$

Our goal is to find these solutions numerically using *Newton's method*.

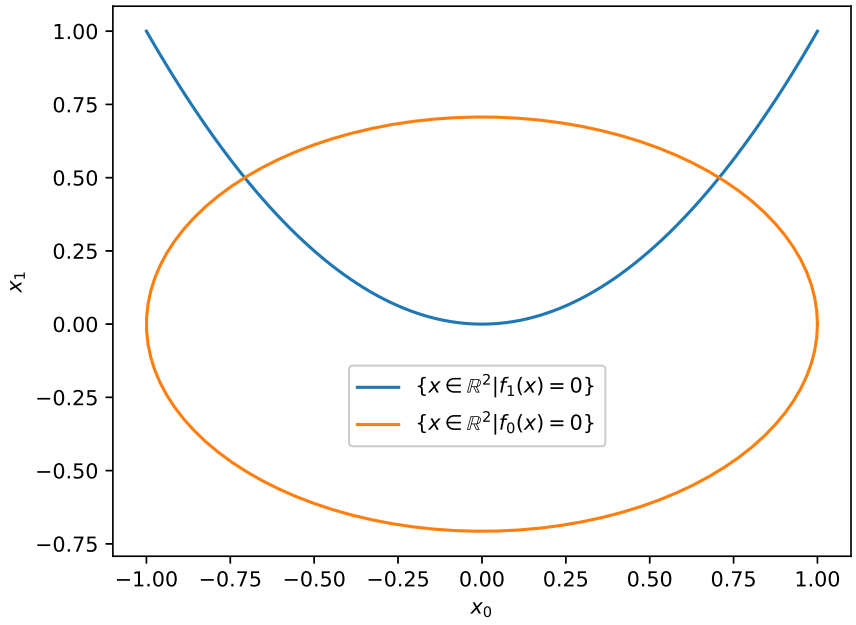

Figure 1: The real solutions are exactly the intersection points of these two curves.

(a) Write $x = (x_0, x_1)$ and let $f : \mathbb{R}^2 \to \mathbb{R}^2$ satisfy

$$f(x) = \begin{pmatrix} f_0(x_0, x_1) \\ f_1(x_0, x_1) \end{pmatrix}$$

for all $x_0, x_1 \in \mathbb{R}$. Write a `C++/Eigen` function

```
Eigen::Matrix2d Jacobian(const Eigen::Vector2d &x);
```

that returns the Jacobian of f evaluated at $x \in \mathbb{R}^2$.

(b) Fix $n \in \mathbb{N}$ and $x \in \mathbb{R}^2$. Write a `C++/Eigen` function

```
Eigen::Vector2d Newton(Eigen::Vector2d x, int n);
```

that returns the approximate solution to (11) obtained from n iterations of Newton's method with starting point x.

(c) Apply Newton's method to Equation (11) with $n = 100$ iterations. Try the starting points $(-1, 1), (1, 1)$ and $(-2, -2)$ and explain the results.