Graded Exercise 1 – Advanced Numerical Analysis

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Prof. D. Kressner
M. Steinlechner

Deadline for submission:

17.4.2015, 10.15 AM at the start of the exercise session.

Work on your own. Do not copy or distribute your answers and codes to other students in the class. Do not reuse any other code related to this homework. Cheating or attempts thereof will be handled according to the discplinary actions defined in the EPFL rules, see `http://polylex.epfl.ch/discipline`.

Guidelines for the submission:

- Answers to the theoretical questions can be handwritten or, preferably, in LaTeX.
- Print out your MATLAB code and hand it in, together with your other answers. Please make sure that is clear to which parts of the exercise you are referring to.
- Put all your MATLAB files into a Zip-file `FirstnameLastname.zip`
- Send the Zip-file to `michael.steinlechner@epfl.ch`, with the subject line "AdvancedNA: Exercise 1 submission"
- Make sure your MATLAB code runs without producing error messages. If your code does not run, you get zero points. For each plot and result that you state in your report, you have to include a MATLAB script that reproduces these results exactly.

1 ▶ Adaptive Step-Size Selection

Read Section II.4 of Hairer, Nørsett, Wanner: *Solving Ordinary Differential Equations I* [HNW] carefully (see lecture homepage for a PDF – login required). You can skip the first section on *Richardson extrapolation*.

Answer the following multiple-choice questions. Note that more than one answer may be correct.

a) Why should one use adaptive time-stepping?

- ◯ To guarantee absolute stability of the method.
- ◯ To control the local error.
- ◯ To reduce the number of time-steps / function evaluations needed for achieving a certain local error.
- ◯ To avoid Taylor expansions.
- ◯ To increase the convergence order of the method.

b) It is advisable to use an *embedded* Runge-Kutta method to

- ◯ avoid Taylor expansions.
- ◯ reduce the number of additional function evaluations.
- ◯ guarantee that equation (4.12) in [HNW] is valid.

c) The global error at the final time t_{end} is guaranteed to be less than 10^{-5} if we choose the tolerance sc_i (as defined in equation (4.10) of [HNW]) in the following way:

- ◯ $sc_i < 10^{-10}$.
- ◯ $sc_i < \frac{10^{-10}}{\tau}$, where τ is the length of the time interval, $\tau := t_{\text{end}} - t_0$.
- ◯ $sc_i < \frac{10^{-10}}{\tau L}$, where L is the supremum of the Lipschitz constant of f along the solution trajectory.
- ◯ none of the above strategies makes sense.

2 ▶ Runge-Kutta with adaptive step-size control

In this exercise, we will derive and implement a Runge-Kutta scheme with adaptive error control and step size estimation, the algorithm behind MATLAB's `ode23` function.

a) Show that the Bogacki-Shampine scheme, given by the following Butcher table:

0				
1/2	1/2			
3/4	0	3/4		
1	2/9	1/3	4/9	
y_1	2/9	1/3	4/9	0
$\hat{y}_1$	7/24	1/4	1/3	1/8

is of order 3 with an embedded formula of order 2.

b) Implement the Bogacki-Shampine scheme in MATLAB as a function

```
[t, y, stats] = bs23( f, T, y0, opts )
```

where `f` is the right hand side of the ODE, `T` is the time interval $[t_0, t_{\text{end}}]$, `y0` is the starting value y_0, and `opts` is an optional argument specifying the parameters such as the error tolerance. The output `stats` should contain information such as the number of function evaluations performed.

Make sure that you do not evaluate the right hand side f more often than necessary.

Use an adaptive step size control as described in [HNW], II.4, using the following parameters: `fac` $= 0.8$, `facmin` $= 0.25$, `facmax` $= 4$, `Atol` $= 10^{-6}$, `Rtol` $= 10^{-3}$.

Important: In contrast to the schemes explained in [HNW], II.4, we perform the new time step with the higher order method *(local extrapolation)*, and only use the embedded lower order method for step size control.

Hint: Depending on the final time t_{end}, it could be that no integer number of steps of size h fit into the time interval. Reduce the step size of the final step so that the final time is hit exactly.

c) Implement a starting step size guess as described in [HNW], page 169.

d) Test your code on the following problem of a charged particle with position $\mathbf{x} \in \mathbb{R}^2$ and velocity $\mathbf{v} := \dot{\mathbf{x}} \in \mathbb{R}^2$ in the conservative force field of a current-carrying wire positioned at the origin $(0, 0)$:

$$\ddot{\mathbf{x}} = -\frac{2\mathbf{x}}{\|\mathbf{x}\|^2}, \quad \mathbf{x}(0) = \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \quad \dot{\mathbf{x}}(0) = \begin{pmatrix} 0.1 \\ -0.1 \end{pmatrix}.$$

Convert this second order differential equation into a first order differential equation and run your code on it, using the time interval $T = [0, 5]$. Plot the resulting path of the particle in space. You should obtain a solution as shown in Figure 1. Furthermore, plot the time evolution of each component function x_1, x_2, v_1, v_2 and compare it to the time evolution of the adaptively chosen stepsize h. You should obtain a plot similar to the one shown in Figure 2.

Hint: To plot the adaptive step size, you can make use of the `stairs` function in MATLAB .

e) Show that the general adaptive step-size strategy defined by equation (4.13) [HNW], is scaling-invariant in the time variable t, that is, it produces equivalent step size sequences when applied to the two problems

$$\begin{aligned} \dot{y} &= f(t, y), & y(0) &= y_0, & y(t_{\text{end}}) &= ? \\ \dot{z} &= \sigma \cdot f(\sigma t, z), & z(0) &= y_0, & z(t_{\text{end}}/\sigma) &= ? \end{aligned}$$

with initial step sizes h_0 and h_0/σ, respectively, and $\sigma \in \mathbb{R}$.

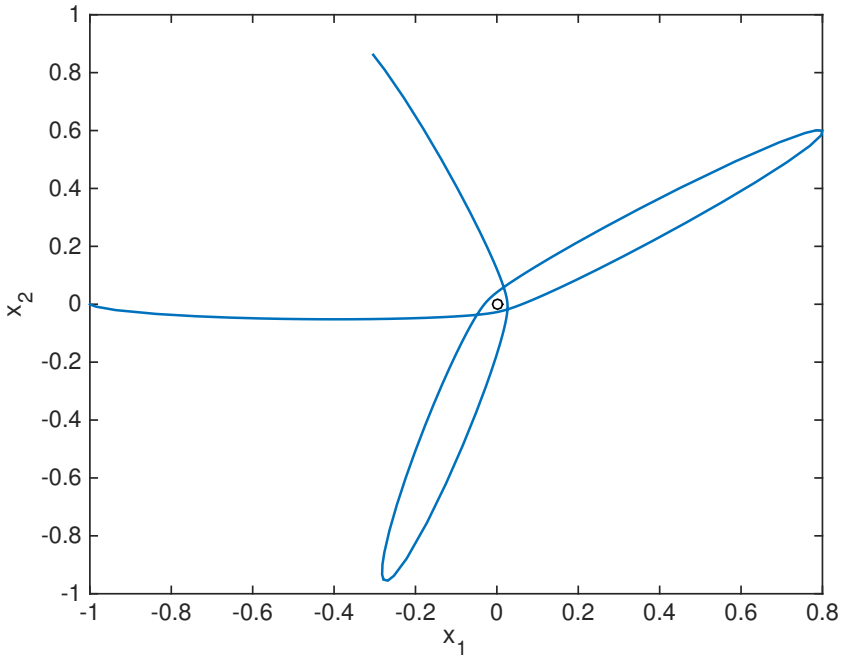

Figure 1: *Solution of the charged particle problem obtained by the adaptive Bogacki-Shampine scheme*

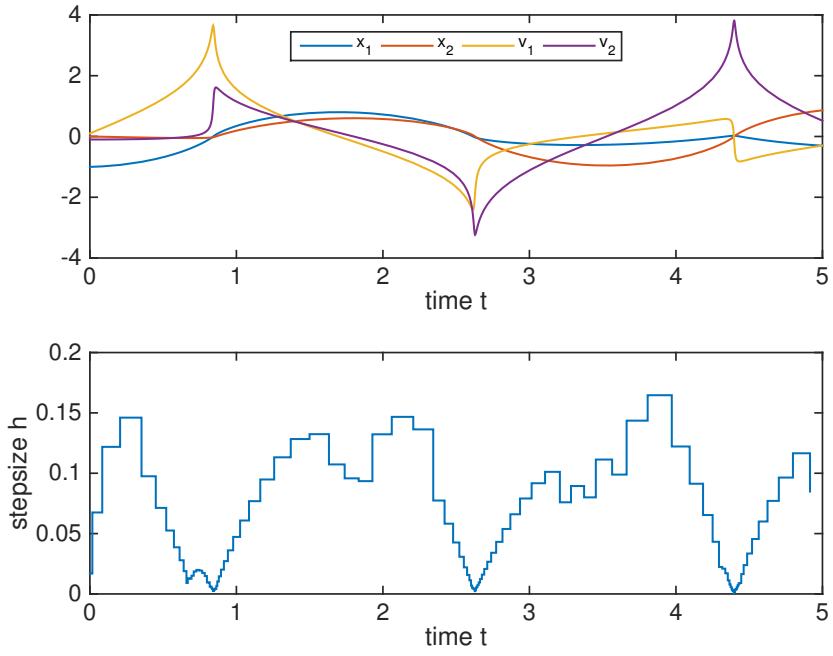

Figure 2: *Adaptive Bogacki-Shampine applied to the charged particle problem and the corresponding step-size selection as functions of time*