Dr Andreas Dedner

# MA261

## Assignment 1 (15% of total module mark) due Thursday 28th January 2020 at 12pm

Do not forget to fill this out in a **legible** way and read through the regulations given below carefully! If you have any question ask me.
Assignment is based on material from weeks 1, 2, and the live lecture on Monday week 3.

| | |
|---|---|
| Student id | |
| Second part joint work with (enter student id of partner) | |

**Mark:**

| Part 1 | | | Part 2 | | | Sum |
|---|---|---|---|---|---|---|
| | | | | | | |

**Regulations:**

- Matlab or Python are the only computer language to be used in this course
- You are *not* allowed to use high level Matlab or Python functions (if not explicitly mentioned), such as diff, int, taylor, polyfit, and ODE solvers or corresponding functions from Scipy etc. You can of course use mathematical functions such as exp, ln, and sin and make use of numpy array etc.
- In your report, for each question you should add M-codes or Python scripts, and worked-out results, including figures or tables where appropriate. Output numbers in floating point notation either using (for example in Matlab) *format longE* or *fprintf('%1.15e',x)*. Have a look at the solution suggestion from the quizzes.

  A concisely written report, compiled with clear presentation and well structured coding, will gain extra marks.
- Do not put your name but put your student identity number on the report.
- Each assignment will be split into two parts, the **first part** needs to be done **individually** by each student for the second part submission in pairs is allowed in which case one student should submit the second part and both need to indicate at the top of the first page that they have worked together on this. Both will then receive the same mark for the second part of the assignment.

## Part 1

**Q 1.1.** **[3/30]** Consider a scalar linear ODE $y' = \lambda y$ with $\lambda \in \mathbb{R}$ fixed. Assume that an approximation is given by

$$y_{n+1} = y_n + P(\lambda h) y_n$$

where $P(\mu) = \sum_{k=0}^{m} \alpha_k \mu^k$ is a polynomial.
Prove that the truncation error satisfies $\tau_n = O(h^{m+1})$ if $P$ is the $m$-th order Taylor polynomial of $e^\mu - 1$ around 0.

**Q 1.2.** **[6/30]** Consider a function $u \in C^3(\mathbb{R})$ and define for $h > 0$

$$d_h u(x) := \frac{u(x+h) - u(x)}{h} \ .$$

Show that with $\theta \in [0, 1]$ the following holds

$$\frac{d}{dx} u(x + \theta h) = d_h u(x) + O(h^p)$$

where $p = 1$ for $\theta \neq \frac{1}{2}$ and $p = 2$ for $\theta = \frac{1}{2}$.

**Q 1.3.** **[6/30]**
Consider a numerical method of the form $y_{n+1} = y_n + h\varphi(t_n, y_n; h)$ for solving an ODE $y' = f(t, y)$. Assume that $\varphi$ is continuous in $h$ and that the ODE has an exact solution $\underline{Y} \in C^1(0, T)$ for any initial condition.
The method is called *consistent* if the local truncation error is $o(h)$, i.e., for every $\epsilon > 0$ and $t \in [0, T - h]$ there exists an $H$ such that $|\tau(t; h)| < \epsilon h$ for all $h < H$ with truncation error as given in the lecture:

$$\tau(t; h) := \underline{Y}(t + h) - \underline{Y}(t) - h\varphi(t, \underline{Y}(t); h)$$

for all $t \in [0, T - h]$.
Show that the method is consistent if $\varphi(t, y; 0) = f(t, y)$.

## Part 2

Hand in one program listing which covers all the questions given below in one single code reusing as much as possible. Avoid any code duplications (or explain why you decide to duplicate some part of the code). The problem considered is scalar but your code should be able to handle vector valued ODEs. So think of the scalar case as also being vector valued but with vectors of size one, i.e., not to have $y$ being a float value. The implementation quiz will test a vector valued version.
In your brief discussion of your result (a paragraph with a plot or a table is sufficient) refer back to the theoretical results discussed in the lecture.

**Q 2.0. [see online quiz on moodle]**
Implement the following three method and test them individually (unit testing). Take a look at the quiz questions which also include some tests and more detail on the function signature to use.
After the quiz closes you can use the provided solutions if you encountered problems implementing the required functions.

(1) Write a function that computes a single step of the forward Euler method

$$y_{n+1} = y_n + hf(t_n, y_n)$$

given $h, t_n \in \mathbb{R}$ and $y_n \in \mathbb{R}^m$ and a general vector valued function $f$ which should be provided as a function handle to your function.
(The solution to this is available in the bonus quiz).

(2) Write a function *evolve* to solve a vector valued ODE of the form

$$y'(t) = f(t, y(t)) \ , \qquad t \in (0, T) \ , \qquad y(t_0) = y_0 \ .$$

using a method of the form:

$$y_{n+1} = \Phi(t_n, y_n; h) \ .$$

The code should be general with respect to the dimension of the problem and the right hand side $f$ and method function $\Phi$ which should both be a parameter to your function together with $t_0, y_0, T$, and the step size $h$.

(3) Write a function *computeEocs* to compute the experimental order of convergence (EOC) for a given sequence $(h_i, e_i)_{i=0}^r$. So the function should return

$$\mathrm{eoc}_i = \frac{\log\left(\frac{e_i}{e_{i-1}}\right)}{\log\left(\frac{h_i}{h_{i-1}}\right)} \ , \qquad i = 1, \ldots, r$$

**Note:** there is a short recorded lecture on this concept which will be uploaded during week 2.

**Q 2.1. [5/30]**
Test your implementation using the scalar ODE (Example 5 and recorded lecture *w1-01*):

$$y'(t) = (c - y(t))^2, \ y(0) = 1, \ c > 0$$

which has the exact solution

$$\underline{Y}(t) = \frac{1 + tc(c-1)}{1 + t(c-1)} \ .$$

Compute errors at the final time $T$ and the EOC for the sequence of time steps given by $h_i = \frac{T}{N_0 2^i}$ for $i = 0, \ldots, 10$ using $N_0 = 20$. Discuss the approximation using

$$c = 1.5 \ , T = 10 \ .$$

**Q 2.2. [5/30]**
Write a program to solve a vector valued ODE of the form

$$y'(t) = f(t, y(t)) \ , \qquad t \in (0, T) \ , \qquad y(0) = y_0 \ ,$$

using the following method

$$y_{n+1} = \Phi(t_n, y_n; h) := y_n + \frac{h}{2}\big(f(t_n, y_n) + f(t_{n+1}, y_n + hf(t_n, y_n))\big) \ .$$

Use the same *evolve* method as before but replace the function handle for the forward Euler method by a function handle implementing the above function. Make sure you optimize your implementation with respect to the number of calls to $f$.
Repeat the experiments from **Q 2.1**.

**Q 2.3. [5/30]**
Compare both methods with respect to their order of convergence (compare the EOCs) and efficiency. To check efficiency compare the errors of the two methods with respect to the number of $f$ evaluations needed to compute $y_0, \ldots, y_N$. Make sure your implementation of the method from **Q 2.2** does not make any unnecessary evaluations of $f$!