Dr Andreas Dedner

MA261

Assignment 3 (20% of total module mark) due Friday 24th March 2023 at 12pm

Do not forget to provide us with the following information in a **legible** way and read through the regulations given below carefully! If you have any question ask me.
Note: we might be marking only some parts of each question.
Assignment is based on material from weeks 1 to 6.

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

Regulations:

- Python is the only computer language to be used in this course
- You are *not* allowed to use high level Python functions (if not explicitly mentioned), such as `diff, int, taylor, polyfit`, or 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`.
- In your report, for each question you should add the Python code, and worked-out results, including figures or tables where appropriate. Use either a Python script and a pdf for the report or combine everything in a Jupyter notebook. We need to be able to run your code.
- Output numbers in floating point notation (using for example the *npPrint* function from the quiz).
- A concisely written report, compiled with clear presentation and well structured coding, will gain marks.
- Do not put your name but put your student identity number on the report.
- The **first part** of each assignment 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 their pdf for the first part the student id of their partner. Both will then receive the same mark for the second part of the assignment.

Part 1

Q 1.1. **[3/20]** Consider the two step LLM

$$y_{n+2} + ay_{n+1} + by_n = hcf(y_{n+s})$$

with fixed constants a, b, c. Answer the following questions for $s = 0, s = 1$, and $s = 2$.

(i) Give conditions for a, b, c so that the method converges.
(ii) Find values for a, b, c so that the method converges and has consistency order of at least 2 (if such a method exists for a given value of s).

Q 1.2. **[3/20]** Consider the non-linear oscillator given by

$$x''(t) + x(t) + \varepsilon\big(x^2(t) - 1\big)x'(t) = 0\ , \qquad x(0) = x_0, x'(0) = 0$$

which is a non-dimensionalized form of the van der Pol equation. It can be shown that this problem has a solution that will (eventually) be periodic (the model has a so called limit cycle). Since the problem is homogeneous we can start anywhere on the cycle, e.g., we can choose a point on the cycle with $x' = 0$ (which should explain the initial conditions). The correct value for x_0 to be on the cycle is unkown.
For the following you will need to be able to solve linear ODEs of the form $y'' + y = F(t)$ where the right hand side is of the form $F(t) = F_0 \cos(\omega t)$ (or $F(t) = F_0 \sin(\omega t)$) for some ω. The general form of the solution is with constants of integration C_1, C_2:

$$Y(t) := \begin{cases} \frac{1}{1-\omega^2} F(t) + C_1 \cos(t) + C_2 \sin(t) & \omega \neq 1\ , \\ \frac{1}{2} t\tilde{F}(t) + C_1 \cos(t) + C_2 \sin(t) & \omega = 1\ . \end{cases}$$

where $\tilde{F}' = F$. This should be known from MA133 or e.g. follow this link: Example 6.1.4 and Example 6.1.5.
Note that the problem is linear so a right hand side of the form $A\cos(\omega_1 t) + B\sin(\omega_2 t)$ can be handled with the same formula, i.e., the solution is the sum of the solutions Y_1, Y_2 given above with right hand sides $F(t) = A\cos(\omega_1 t)$ and $F(t) = B\sin(\omega_2 t)$, respectively. The central thing to take into account is the difference in the solution for $\omega \neq 1$ and $\omega = 1$ - the second case is know as *secular solution*.

(i) Using regular perturbation theory of the form $x_0(t) + \varepsilon x_1(t) + O(\varepsilon^2)$ derive a system of ODEs for x_0, x_1.
(ii) Solve the ODE you obtain for x_0, x_1 under the assumption that we are looking only for periodic solutions (i.e. the limit cycle).

Q 1.3. **[4/20]** Consider the problem from Q1.2. Using an expansion up to $O(\varepsilon^3)$ and after substituting x_0, x_1 into the ODE for x_2 one obtains

$$x_2'' + x_2 = \frac{1}{4}\cos(t) + 2C\sin(t) - \frac{3}{2}\cos(3t) + 3C\sin(3t) + \frac{5}{4}\cos(5t)$$

with some free parameter C.
You can see that the right hand side contains two terms with $\cos(t)$ and $\sin(t)$ which lead to secular terms in x_2. There is only one free constant C to choose which is not enough to eliminate both of these terms. So we can not obtain a periodic expansion above $O(\varepsilon^2)$. To construct a periodic expansion of higher order we need to allow the frequency of the oscillator to adapt to ε. To this end we define a *stretched time variable* $\tau = \omega t$ where the frequency ω also has an expansion of the form $\omega = 1 + \epsilon\omega_1 + O(\varepsilon^2)$.

(i) Derive an ODE for the function $y(\tau) = x(\frac{\tau}{\omega})$.
(ii) Use the expansion for ω and $y_0(t) + \varepsilon y_1(t) + O(\varepsilon^2)$ to derive a system of ODEs for y_0, y_1 (which should depend on ω_1).
(iii) Find periodic function y_0, y_1 - what choice for ω_1 does this require?

Remark: Expanding both ω and y up to $O(\varepsilon^3)$ leads to an equation for y_2 that looks so unpleasant that I didn't want to force you to write it down. After substituting y_0, y_1, ω_1 into the ODE for y_2 one obtains

$$y_2'' + y_2 = (4\omega_2 + \frac{1}{4})\cos(\tau) + 2C\sin(\tau) - \frac{3}{2}\cos(3\tau) + 3C\sin(3\tau) + \frac{5}{4}\cos(5\tau)$$

with some constant C. It is now possible to obtain a periodic solution y_2 by choosing C, ω_2 in such way that the secular terms are removed.

Part 2

Hand in one program listing which covers all the questions given below in a single script/notebook reusing as much as possible. Avoid any code duplications (or explain why you decide to duplicate some part of the code).
Important: 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 or from assignments.

Scipy initial value solver: Scipy provides a solver based on adaptive solvers as discussed in week 9. It quite easy to write an `evolve` type method using this solver:

```
from scipy.integrate import solve_ivp
def evolveScipy(method, f,Df, t0,y0, T, rtol, atol):
    res = solve_ivp(f, [t0,T], y0, method=method, rtol=rtol, atol=atol)
    return res.t, res.y.transpose()
```

Note that the Scipy solver returns y differently from our setup which can be solved by using `transpose` as shown above. Have a look here to get an idea of the parameters to call this method https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html. In this assignment we will use the `method="RK45"` which is similar to the embedded explicit Runge-Kutta method you will be implementing. The aim is to see if your code is competitive with the Scipy implementation.
Since these methods are adaptive they do not use a constant time step but one that it adapts to the solution. You can obtain the time steps used in the following way:

```
t,y = evolveScipy("RK45", f,Df, ...)
h   = np.diff(t)
```

Think about ways to visualize the varying time steps by for example overlaying the logarithm of the time step with the actual solution on one graph or coloring the phase space portrait according to the time step used - I would be interested to see what that looks like; so if you tried that and decided it is not so informative mention that in the report.

For these methods you do not chose a value of N (or h) to increase accuracy (or decrease computational cost) but you chose tolerances. Instead of having two different values for the tolerances (for the absolute and relative error), I suggest to combine the two by choosing one tolerance `tol` and then use for example `rtol=tol,atol=tol*1e-2`.
A lot of this assignment is about coming up with ways to visualise the outcome of simulations and comparing different methods, which is as you hopefully figured out by now, a major challenge in computational mathematics. I've made some suggestions below you don't need to follow all of them but some...

Q 2.1. [7/20]
We will solve the *van der Pol* oscillator discussed in part 1 (and in the second part of the lecture in the context of Hamiltonians).
Start by having a look at the problem using the build-in method. You can do this question for a partial mark using only the build-in method.
Model:
Solve the *van der Pol* equation

$$x'' - \mu(1 - x^2)x' + x = 0$$

with $\mu = 200$ which you of course will have to rewrite as a first order system.
We don't know the exact solution but it is possible to obtain an estimate Θ for the period of the oscillation

$$\Theta = \big(3 - 2\log(2)\big)\mu + 2\pi\mu^{-\frac{1}{3}} \ .$$

We will use $T = 3.1\Theta$ as final time so that we get three estimates for the period which we can average.

Here is a webpage discussing this problem including some Python code https://www.johndcook.com/blog/2019/12/22/van-der-pol. There is also a link discussing the calculation of the period - although you will have to adapt that for your own implementation.

Task:

Implement your own adaptive time stepping method - the idea is explained in the recording for week 9. A short summary of the method is also given on the final page of this assignment. Compare your method with the build-in method.

Setup:

Compare your implementation with the build-in version by trying to obtain a relative error in the period of below 10^{-3} without too much computational cost. For each method, try to find a value for the tolerance `tol` that is as large as possible so that the methods produce solutions with a relative error in the period of less than $1e-3$. Note that the build-in implementation is of course not identical to the one you implemented so you might need different tolerance to get a sufficiently accurate result. My suggestion is to automate finding the "optimal" tolerance `tol` by starting with a fairly large value (not too large because the method might fail) and then decreasing this value by a fixed factor, e.g., 2, until the desired accuracy for the period is reached.

You will need some way to estimate the period Θ given a discrete approximation $(t_n, y_n)_n$. You could for example find $n_1, n_2, .., n_k$ where y_n crosses the x- or y-axis. Then $t_{n_i} - t_{n_{i-2}}$ is an approximation to the period for $i = 2, 3, 4, ..., k$. You could average these approximations to obtain an average discrete period Θ_h produced by the approximation. First test your approach by seeing that you can get the build-in method to produce an approximation with $|\Theta - \Theta_h|/|\Theta| < 10^{-3}$.

Discussion:

Once you obtained the "optimal" tolerance for each method, discuss how the two method compare with respect to number of steps required. Are the time step chosen over the time interval similar? Also estimate the computational cost of both methods - you can use the Python `time` module for example or add a counter to the function f to find out the number of evaluations of the right hand side required (you can use a `global` variable which you increment each time f is called).

Final remark:

the build-in method has some way of finding a good value for the first h while we need to prescribe one - you could start with the same one that the build-in method uses. Also try to chose `hmax` large enough so that it does not influence the outcome - it should not be needed for this problem.

Q 2.2. [3/20]

You can do this question only using the build-in method for partial marks if you didn't manage to implement you own adaptive method.

Have a look at the problem described here: https://www.johndcook.com/blog/2020/02/08/arenstorf-orbit. Try to reproduce the phase portrait included on this page showing the stable periodic orbit which apparently was the basis for the Apollo mission. This problem is also described in one of the books on the reading list ("Solving ordinary differential equations" by Hairer, Nørsett, and Wanner page 127). Use the more accurate initial conditions given there $y_0 = (0.994, 0, 0, -2.00158510637908252240537862224)$. That source also provides an approximation to the period $\Theta = 17.065216560157962558891720 6249$. Chose $T = 3\Theta$ and try to find tolerances so that the final value is close to the initial conditions, e.g., $|u_N - u_0|/|u_0| < 0.01$. You might have to chose quite a small tolerance. Can you manage to maintain a periodic orbit with $T = 5\Theta$ for example?

Adaptive time stepping based on embedded RK methods:
The idea of automatically computing a time step is based on so called pairs of *embedded Runge-Kutta methods* where the same stages are used to compute two different approximations y_{n+1}, y^*_{n+1} by using two different vectors γ, γ^*. A Butcher tableau for example an DIRK version of an embedded RK pair would be

$$\begin{array}{c|cccc} \alpha_1 & \beta_{1,1} & & & 0 \\ \alpha_2 & \beta_{2,1} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ \alpha_s & \beta_{s,1} & \dots & \beta_{s,s-1} & \beta_{s,s} \\ \hline & \gamma_1 & & \gamma_{s-1} & \gamma_s \\ & \gamma^*_1 & & \gamma^*_{s-1} & \gamma^*_s \end{array}$$

We will use the following pair of order 4 and 5:

```
alpha = array([ 0,1/5,3/10,3/5,1,7/8 ])
beta  = array([ [0,           0,         0,         0,             0,         0 ],
                [1/5,         0,         0,         0,             0,         0 ],
                [3/40,        9/40,      0,         0,             0,         0 ],
                [3/10,       -9/10,      6/5,       0,             0,         0 ],
                [-11/54,      5/2,      -70/27,     35/27,         0,         0 ],
                [1631/55296,  175/512,  575/13824,  44275/110592,  253/4096,  0 ]
              ])
gamma = array([ 37/378, 0, 250/621, 125/594, 0, 512/1771 ])
gammaStar = array([ 2825/27648, 0, 18575/48384, 13525/55296, 277/14336, 1/4 ])
```

So there is little extra computational cost to compute both y_{n+1}, y^*_{n+1} compared to simply computing y^*_{n+1}. We assume that the order of consistency of the method using γ is p while the consistency using γ^* is $p+1$. One can now estimate the error done in this step by computing $E := |y_{n+1} - y^*_{n+1}|$ and use this to estimate a good value for the time step to take. Details on where that formula comes from can be found in the recording for week 9.
Algorithm: Given t_n, y_n and a initial step size h_n, e.g., the one estimated in the previous time step:

```
TOL = atol + rtol * |y_n|
E = 2 * TOL
H = h_n
while E > TOL
  h = H
  compute y_{n+1} and y*_{n+1} using the embedded RK methods
  (make sure you only compute the stages once!)
  E = |y_{n+1} - y*_{n+1}|
  if E < 1e-15
    need to come up with some strategy, e.g., use some h_max
  end
  H = 0.9h (TOL/E)^{1/(p+1)}   p is order of lower order RK method
  if H < 1e-10
    error: scheme is probably not working
  end
end
t_{n+1} = t_n + h          new time
y_{n+1} = y*_{n+1}         we use the higher order version
h_{n+1} = min{h_max, H}    suggestion for next time step
```

where $h_{\max}$ and $atol, rtol$ are fixed constant chooses a-priori. As you can see we are using the approximation given by the higher order method $y^\star_{n+1}$ although the theory says to use the lower order approximation.
You will have to change your `evolve` method a bit since the *adaptive stepper* computes t_{n+1}, y_{n+1} and a suggestion for h to use for the next step. Also you should make sure (at least for Q2.2) that you don't go past T, i.e., that $t_N = T$.