AMSC661, Spring 2023 Maria Cameron

Homework 2. Due Feb. 15

Please upload a single pdf file on ELMS. Link your codes to your pdf (i.e., put your codes to dropbox, Github, google drive, etc. and place links to them in your pdf file with your solutions.

1. The goal of this exercise to become familiar with various ODE solvers available in Matlab and/or Python and apply them to a few test problems.

   (a) Read the description of available built-in ODE solvers in Matlab https://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html and in Python https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html#scipy.integrate.solve_ivp. (For Python, see also more syntaxis options here and examples here)

   (b) **(5 pts)** Test the ODE solvers available in Matlab **or** Python on the Van Der Pol oscillator

   - Matlab: `ode45` (explicit Runge-Kutta method of order 5 DOPRI5(4)) and `ode15s (linear multistep method of variable order from 1 to 5 for stiff problems)`;
   - Python: `RK45` (explicit Runge-Kutta method of order 5 DOPRI5(4)) and LSODA (linear multistep method based on Adams and BDF with automatic stiffness detection):

   $$y_1' = y_2,$$
   $$y_2' = \mu((1-y_1^2)y_2) - y_1.$$

   with $\mu = 10$, 10^2, and 10^3. The greater is μ, the stiffer is the problem. Set `tmax = 1000.0`. Set error tolerances $\epsilon \equiv atol \equiv rtol$, $\epsilon = 10^{-6}$, 10^{-9}, and 10^{-12}. You can pick the initial condition $y_1(0) = 2$, $y_2(0) = 0$. Plot the solution on the phase plane (y_1, y_2). For each value of μ, make a single plot of the solution. Observe how the limit cycle changes at μ increases. Measure the CPU time T_{CPU} required to compute one cycle for each of the solvers for each of the values of μ and plot $\log\epsilon$ versus $\log(CPUtime)$. Comment on how the CPU time depends on the error tolerance for each value of μ for each method. Try to explain your observations.

   (c) **(5 pts)** Now test appropriate ODE solvers on the Arenstorf problem. More details are found in this note on pages 3–4. Use exactly the same values for the parameters and the initial conditions as in the note. First set the $T_{\max}$ equal to the period, compute the periodic orbit using DOPRI5(4), and plot it (x vs y). Then set $T_{\max} = 100$. Set error tolerance $\epsilon \equiv atol \equiv rtol = 10^{-12}$.

- If you program in Matlab, compute the numerical solution using `ode45`, `ode78`, and `ode89`.
- In Python, use `RK45`, `DOP853`, and `Radau`.

Measure CPU times and plot the orbit of the satellite for each solver. Comment on the CPU times and plots that you are observing and compare the solvers.

2. **(5 pts)** On the same coordinate plane, plot the regions of the absolute stability for the following methods ERK methods:

   - Forward Euler;
   - Midpoint rule with Euler Predictor (this is the example in my code `RK_RAS.ipynb` available on ELMS in Files/Codes);
   - Kutta's method (three-stage, third order) (see the last page of this PDF file);
   - the standard 4-stage, fourth order Runge-Kutta method (see the first method in this Wiki article);
   - DOPRI5(4) – see the last page of this PDF file). Use $\hat{y}$ for the method of order 5.

3. **(5 pts)** Prove convergence for the backward Euler method from scratch. You should mimic the general proof of convergence for one-step methods, but simplify and adapt it for backward Euler.

0			
$\frac{1}{2}$	$\frac{1}{2}$		
1	-1	2	
	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$

Table 1: Kutta's method

0			
$\frac{1}{3}$	$\frac{1}{3}$		
$\frac{2}{3}$	0	$\frac{2}{3}$	
	$\frac{1}{4}$	0	$\frac{3}{4}$

Table 2: Heun's method

0						
$\frac{1}{5}$	$\frac{1}{5}$					
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$			
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$		
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$	
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$
y_1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
$\hat{y}_1$	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

Table 3: Embedded Runge-Kutta Method: Dormand and Prince 5(4).