CS370 Fall 2023: Assignment 2

Instructors: Justin Wan (email: justin.wan@uwaterloo.ca)
Leili Rafiee Sevyeri (email: leili.rafiee.sevyeri@uwaterloo.ca)

Due November 1, Wednesday, at 10:00pm EDT

Submit all components of your solutions (written/analytical work, code/scripts, figures, plots, output, etc.) to Crowdmark in PDF form for each question (multiple pages are allowed). Note that you may resubmit as often as necessary until the due date - only the final submission will be marked.

You must also separately submit a single zip file containing any and all code/scripts you write to the Assignment 2 DropBox on LEARN, in runnable format (and if necessary, any data needed to reproduce your results).

You have a number of options of how to prepare your solutions. You can typeset your solutions in a word-processing application (MS Word, LaTeX, etc.), you can write on a tablet computer, or you can write on paper and take photos. **It is your responsibility to ensure that your submission is sufficiently legible. This is particularly important if you are submitting photos of handwritten pages.** TAs have the right to take marks off for illegible answers.

Note that your solutions will be judged not only for correctness but also for the quality of your presentation and explanations.

1. **(15 marks)** Convergence of ODE Methods

   Consider the initial value problem

   $$y'(t) = -y^2 \cos(2t), \quad y(0) = 1.$$

   (a) (2 marks) Show that

   $$y(t) = \frac{2}{\sin(2t) + 2}$$

   is an exact closed-form solution to this particular IVP (including initial conditions). Hint: You do not need to analytically *solve* the ODE; just show that the proposed solution $y(t)$ satisfies it.

   (b) (1 mark) Create a single Jupyter notebook for the rest of this question, `A2Q1.ipynb`. Define a function, `Dynamics(t,y)` that implements the dynamics function for the above problem given `t` and `y`. i.e., it should return the corresponding derivative value $y'(t)$. Define a second function `ExactVal(t)` that returns the exact solution of y for a given time t.

   (c) (2 marks) Define a function, `y = ForwardEuler(t`$_0$`,t`$_{final}$`,N,y`$_0$`)`, which takes as input the initial time, `t`$_0$, the final time, `t`$_{final}$, the number of time steps, `N`, and the initial value, `y`$_0$, and implements the Forward Euler method. It should return the computed *approximate* solution as a vector `y` of length N+1 such that

   $$\mathtt{y}_i \approx y(t_i) \quad \text{for} \quad i = 0, 1, 2, ..., N$$

   where $t_i = t_0 + ih, h = \frac{t_{final} - t_0}{N}$, and $y(t_i)$ denotes the exact solution at time $t = t_i$. Your `ForwardEuler` function should call the function `Dynamics` (created in part (b)) internally when needed.

(d) (2 marks) Next, apply your `ForwardEuler` function to solve the IVP to time $t_{final} = 5$, first with $N_1 = 20$ constant-size timesteps, and then again with $N_2 = 40$ constant-size time steps. Plot these numerical solutions together on the same graph over the domain $0 \leq t \leq 5$, using the '+' symbol for the first solution and the 'o' symbol for the second solution. Also on the same graph, plot the analytical solution using solid lines. (To create the exact solution plot, evaluate it at the same discrete times as the N_2 solution.)

(e) (2 marks) A time stepping rule for another second order Runge-Kutta scheme is given by:

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + 2h/3, y_n + 2hk_1/3), \\ y_{n+1} &= y_n + h(k_1/4 + 3k_2/4). \end{aligned} \tag{1}$$

Define another function, `y = RK2(t`$_0$`,t`$_{final}$`,N,y`$_0$`)`, which implements the Runge-Kutta scheme, similar to what you did in part (c) for Forward Euler.

(f) (2 marks) Next, apply both your `ForwardEuler` and `RK2` functions to solve the IVP to time $t_{final} = 5$, with $N = 20$ constant-size time steps. Plot these two numerical solutions on the same graph over the domain $0 \leq t \leq 5$, using the '+' symbol for the Forward Euler solution and the 'o' symbol for the RK2 solution. Also on the same graph, plot the analytical solution using solid lines. (To create the exact solution plot, evaluate it at the same discrete times as the other two solutions.)

(g) (2 marks) We would now like to calculate numerical evidence illustrating the order of the Forward Euler method. Let $\text{err}(h)$ denote the absolute (global) error at t_{final} after computing with step size h. For a p-order method, $\text{err}(h) \approx Ch^p$ for some constant C, and so the effect of cutting the step size in half is

$$\frac{\text{err}(h/2)}{\text{err}(h)} \approx \frac{1}{2^p}$$

i.e., the error is reduced by a factor of 2^p. Therefore, the order p of a given method can be illustrated by calculating the above ratios for decreasing h.

For the Forward Euler method, calculate a vector of successive computed values for the solution at $t = 2$ based on cutting the step size in half a number of times. Specifically, use $N = 5 \times 2^i$ timesteps, for $i = 0, 1, ..., 9$. Your code should then calculate the errors for each timestep size and the successive ratios $\frac{\text{err}(h/2)}{\text{err}(h)}$, and print this information in rows in a simple text table (just use `print`, no fancy formatting required). From your experimental evidence, what is the order of the Forward Euler method?

(h) (2 marks) Repeat part (g) using your `RK2` function. What is the order of this RK2 method?

2. **(4 marks)** Higher Order and Systems of ODEs

Consider Newton's equations of motion for a two-body problem specified by:

$$\frac{d^2x}{dt^2}(t) = -\frac{x(t)}{(x(t)^2 + y(t)^2)^{3/2}}; \qquad x(0) = 0.4; \;\; \frac{dx}{dt}(0) = 0$$

$$\frac{d^2y}{dt^2}(t) = -\frac{y(t)}{(x(t)^2 + y(t)^2)^{3/2}}; \qquad y(0) = 0; \;\; \frac{dy}{dt}(0) = 2\,.$$

As t ranges from 0 to 2π, $(x(t), y(t))$ defines an ellipse. Convert this initial value problem into an equivalent first order system of ODEs which has the form:

$$\frac{d\mathbf{u}}{dt}(t) = F(t, \mathbf{u}) \,.$$

3. **(8 marks)** Forward Euler and Improved Euler Methods

   Solve the initial value problem by hand calculations

   $$\frac{dy(x)}{dx} = x + y, \quad y(0) = 2,$$

   using Forward Euler and Improved Euler methods with step size $h = 0.5$. Show your work. Compare your results with the exact solution

   $$y(x) = 3\mathrm{e}^x - x - 1,$$

   at $x = 0.5$ and 1. Compute the errors of the approximate solutions given by these methods. (You may use a calculator to perform the individual arithmetic steps, but be sure to show how you arrived at your result. You are not allowed to write Python code for this question.) Display your results in a table which shows the solutions given by the ODE methods and their corresponding errors at each time step.

4. **(6 marks)** Local Truncation Error

   Consider solving a differential equation $y'(t) = f(t, y(t))$ using the time-stepping method given by

   $$\frac{y_{n+1} - y_{n-1}}{2} = hf(t_n, y_n). \tag{2}$$

   The timestep is a constant $h = t_{n+1} - t_n = t_n - t_{n-1}$.

   (a) (1 mark) State whether this is an explicit or implicit method.

   (b) (1 mark) State whether this is a single-step or multi-step method.

   (c) (4 marks) Determine the local truncation error for this method in the form of

   $$LTE = C\, h^\alpha y^{(\beta)}(t_n) + O(h^{\alpha+1}),$$

   where C is a constant, and α and β are integers. Determine the values of C, α, and β.

5. **(6 marks)** Stability

   Consider solving a differential equation $y'(t) = f(t, y(t))$ using a second-order Runge-Kutta method given by

   $$y_{n+1} = y_n + \frac{h}{4}\left(f(t_n, y_n) + 3f\left(t_n + \frac{2h}{3}, y_n + \frac{2h}{3} f(t_n, y_n)\right)\right). \tag{3}$$

   Analyze the stability of this method with our usual test equation

   $$f(t, y(t)) = -\lambda y(t); \qquad \lambda > 0. \tag{4}$$

   Explain whether (and under what conditions) the method is stable.

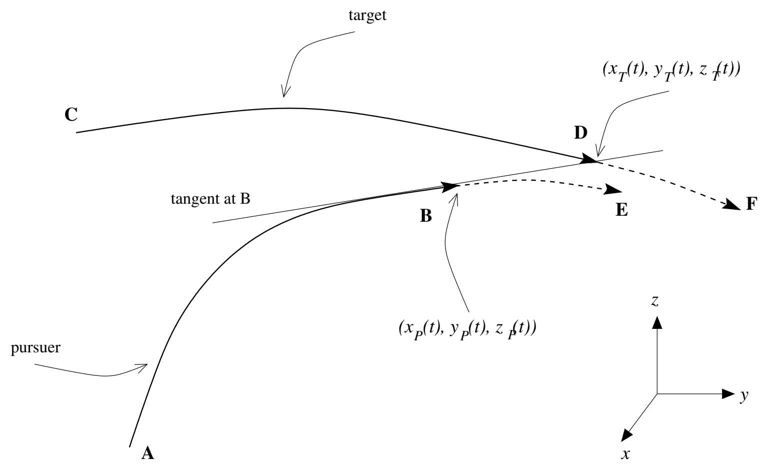

Figure 1: Pursuit Problem: trajectories of target and pursuer.

6. **(11 marks)** Pursuit Problem

A pursuit problem involves a *target* and a *pursuer*, the latter moving in such a manner that its direction of motion is always towards the target. Figure 1 shows the trajectories of the target (CD) and pursuer (AB):

The pursuit problem is to determine the trajectory of the pursuer, given the initial position of the pursuer and the position vector of the target as a function of time for $t \geq 0$. Specifically, the trajectory of the target is represented by a parameter curve with parameter t:

$$T(t) = (x_T(t), y_T(t), z_T(t)),$$

where $x_T(t)$, $y_T(t)$, and $z_T(t)$ are known functions which describes the path of the target. The trajectory of the pursuer is the parametric curve: $P(t) = (x_P(t), y_P(t), z_P(t))$. Our model of the pursuit strategy is as follows:

(i) At any time t, the motion of the pursuer must point directly to the target.

(ii) The speed of the pursuer is assumed to be a known constant, s_P.

The implication of point (i) is that the velocity vector at $P(t)$ should have direction $T(t)-P(t)$ at every time t. Notice that the length of $T(t) - P(t)$ is

$$dist(t) = \sqrt{(x_T(t) - x_P(t))^2 + (y_T(t) - y_P(t))^2 + (z_T(t) - z_P(t))^2}.$$

Thus point (i) implies that $\frac{dP(t)}{dt}$ should have the direction of the unit vector $(T(t)-P(t))/dist(t)$. On the other hand, point (ii) implies that the length of $\frac{dP(t)}{dt}$ is s_P. Consequently, the equations describing the pursuer's motion are given by the following systems of ODEs:

$$\begin{aligned}
\frac{dx_P(t)}{dt}(t) &= \frac{s_P}{dist(t)}(x_T(t) - x_P(t)),\\
\frac{dy_P(t)}{dt}(t) &= \frac{s_P}{dist(t)}(y_T(t) - y_P(t)),\\
\frac{dz_P(t)}{dt}(t) &= \frac{s_P}{dist(t)}(z_T(t) - z_P(t)).
\end{aligned}$$

The starting position of the pursuer; i.e. the initial value $P(0)$ is assumed to be known.

Solve the pursuit problem numerically using Python ODE solvers `RK23` and `RK45`. (You may find `scipy.integrate.solve_ivp` useful.) In particular, complete the Python notebook `A2Q6.ipynb` (download from Crowdmark) which includes the following functions:

- `target`: (1 mark) The trajectory of the target is a helix which is given by:

$$x_T(t) = \sin(t), \quad y_T(t) = \cos(t), \quad z_T(t) = t.$$

- `rocket`: (1 mark) This is the dynamics function of the systems of the ODEs for the pursuit problem. It may call the `target` function above.
- `rocket_events`: (1 mark) This is the event function which detects whether the pursuer has caught the target. More precisely, you should stop the ODE solver when the distance between the pursuer and the target is less than `DMIN`.
- `animate`: This is for animation (you do not need to do anything).

In the main program, you should set the time span to be $[0, 15]$, the initial position $P(0) = (0, 0, -3)$, and `DMIN`=0.1. Set the ODE options: absolute tolerance = `1e-5`, relative tolerance = `1e-5`, initial step size = 1, maximum step size = 1, and `terminal=True` when an event occurs. Also set `args=(SP,DMIN)` when you call the ODE solver.

Use different speeds, `SP`=1.4, 1.5, 1.6, 1.7, to solve the problem. For each speed, do the following:

(a) (2 marks) Solve the pursuit problem using `RK23` and `RK45` using the parameters described above.

(b) (3 marks) Report (1) the number of time steps used by the `RK23` and `RK45`, (2) the number of evaluations of the right-hand side, and (3) the time of intercept if the pursuer and the target intercept; otherwise, report no intercept.

(c) (2 marks) Make a subplot of the target and pursuer trajectories given by `RK23`, and another subplot by `RK45`. Label the axes and the title should include the speed and the name of the solver.

(1 mark) Which of the ODE solver is more efficient? Explain your answer by commenting on the number of time steps and function evaluations.