AMATH 301 – Winter 2023
Homework 7

Due: 11:59pm, February 30, 2023.

Instructions for submitting:

- Coding problems are submitted to Gradescope as a python script/Jupyter notebook (`.py` or `.ipynb` file). (If you use Jupyter Notebook, remove all "magic": the lines that begin with %). You have **8 attempts** (separate submissions to Gradescope) for the coding problems.
- Writeup problems are submitted to Gradescope as a single `.pdf` file that contains text, plots, and code at the end of the file for reference. Put the problems in order and label each writeup problem. When you submit, **you must indicate which problem is which on Gradescope (including the code that belongs to the problem). Failure to do so will result in a 10% penalty.. All code used for each problem should be included at the end of that problem in your .pdf file and be marked as part of the problem on Gradescope. Failure to include code will result in a 25% penalty.**

Coding problems

1. So far in class the only *implicit* method we have discussed is the *Backward-Euler* method, but there are many more (infinitely many!). For example, consider the second-order Adam's Moulton method,

$$y_{k+1} = y_k + \frac{\Delta t}{2}\left(f(t_{k+1}, y_{k+1}) + f(t_k, y_k)\right),$$

(can you tell why it's implicit?). In this problem, we will solve the stiff ODE discussed in Activity 5,

$$\begin{aligned} y'(t) &= (5 \times 10^5)(-y + \sin(t)), \qquad 0 \le t \le 2\pi \\ y(0) &= 0, \end{aligned}$$

using 100 equally spaced points between 0 and 2π.

   (a) To solve using the Adam's Moulton method given above, you will need to solve a root-finding problem, $g(z) = 0$, at each step. To make sure that you are setting this up correctly, create the anonymous function g at $t = 0$ and evaluate $g(3)$. Save the result to `A1`.

(b) Implement the Adam's-Moulton method given above to solve the IVP at the 100 equally spaced points between 0 and 2π. Save the resulting array of 100 elements to the array `A2`.

2. In this problem we will compare the built-in explicit solvers that implement RK45 `scipy.integrate.solve_ivp` with the implicit solvers, `scipy.integrate.solve_ivp` with `method = 'BDF'`.

   The Belousov-Zhabotinsky (BZ) reaction is a chemical reaction that gives rise to a *nonlinear chemical oscillator*. The BZ reaction has many important applications, but most importantly for me, it looks cool. Really, I highly recommend looking it up on YouTube.

   A model for the BZ reaction is given by

$$\begin{aligned} y_1'(t) &= s(y_2 - y_1y_2 + y_1 - qy_1^2) \\ y_2'(t) &= \frac{1}{s}(-y_2 - y_1y_2 + y_3) \\ y_3'(t) &= w(y_1 - y_3). \end{aligned} \tag{1}$$

   Here y_1 corresponds to the concentration of Bromous Acid, ($HBrO_2$), y_2 corresponds to the concentration of Bromide (BR^-), y_3 corresponds to the concentration of Cerium (Ce^{4+}), and s, w, and q are dimensionless parameters relating to the rate of conversions between compounds (you can tell I am a chemist). Here we will take

$$s = 77.27, \qquad \text{and} \qquad w = 0.161,$$

   and we will change q to see how it changes the behavior of the solutions (changing q changes the stiffness of the ODE). Throughout, we will use the initial condition $y_1(0) = 1,\ y_2(0) = 2,\ y_3(0) = 3$ and solve for $0 \le t \le 30$.

   (a) Create a function `odefun` implementing (1) with $q = 1$. Your function `odefun` should be of the form

```
odefun = lambda t, y: ...
```

   where `y` is an array input. Test your `odefun` solution by evaluating it at $t = 1$ and $(y_1, y_2, y_3) = (2, 3, 4)$. Save the result as an array to the variable `A3`.

   (b) Create a list of 10 logarithmically spaced points between 10^0 and 10^{-5} (`np.logspace`). We will solve (1) for each of these 10 values (e.g., solve first for $q = 1$, then for $q = 10^{-0.5556} \approx 0.2783$, etc.). For each q calculate the solution using RK45 from `scipy.integrate.solve_ivp`. Save and record the solution at the final time ($[y_1(30), y_2(30), y_3(30)]$). Doing this 10 times will give a 3×10 array (3 values for each of the 10 times). Save the result to the variable `A4`.

   (c) Do the same thing as above using BDF from `scipy.integrate.solve_ivp`. Save the resulting 3×10 array to the variable `A5`.

3. In this problem we are going to consider a famous problem, that can have stiffness. It is the Van der Pol oscillator,

$$x''(t) - \mu(1 - x(t)^2)x'(t) + x(t) = 0. \tag{2}$$

The Van der Pol oscillator represents an oscillator with non-linear damping. The damping term is the $\mu(1-x^2)x'$ term. In this problem we will use $\mu = 200$ and the initial conditions $x(0) = 2$, $x'(0) = 0$.

(a) Recall that second-order ODEs can be written in the form of two first-order ODEs. Define $y = x'(t)$ to find a system of two first-order ODEs for the Van der Pol oscillator. Your system should look like

$$x'(t) = ...$$
$$y'(t) = ...$$

Create two anonymous functions, `dxdt` and `dydt`, that implement the system you've defined above. Plug in $(x, y) = (2, 3)$, save the result of x' to the variable `A6` and y' to the variable `A7`.

(b) Solve the ODE using the built-in RK45 solver for $0 \leq t \leq 400$. Save the values of $x(t)$ to the array `A8`.

Hint: think about how we use the initial conditions $x(0) = 2$ and $x'(0) = 0$ with the new variables $x(t)$ and $y(t)$. How is $y(0)$ related to $x'(0)$?

(c) Solve the ODE using the built-in implicit BDF solver with the same configuration as above. Save the values of $x(t)$ to the array `A9`.

(d) With $\mu = 200$ the ODE is stiff (you will explore this more in Writeup Problem 1). To measure the stiffness, record the ratio between the number of points used in the RK45 solver to the number of points used in the implicit BDF solver. Save your answer to the array `A10`.

(e) The nonlinear Van der Pol equation is too difficult to solve analytically, so people often resort to a simpler model. When $|x|$ is small, one can use *linearization* (a useful tool!) to approximate the ODE. This gives a *linear* second-order differential equation for the Van der Pol oscillator,

$$x''(t) - \mu x'(t) + x(t) = 0. \tag{3}$$

Again convert this to a system of two ODEs using $y = x'(t)$ and create two functions `dxdt` and `dydt` representing $x'(t)$ and $y'(t)$ respectively. To check your work, plug in $(x, y) = (2, 3)$, save the result of x' to the variable `A11` and y' to the variable `A12`.

(f) Because the system of ODEs you found above is *linear*, it can be written in matrix form (more on this in Week 9!). The above system can be written

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}, \tag{4}$$

where

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}, \qquad \mathbf{x}'(t) = \begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix}, \qquad \text{and} \qquad \mathbf{A} = \begin{pmatrix} 0 & 1 \\ -1 & \mu \end{pmatrix}$$

Create the matrix A in python as a `numpy` array. For example, if we had

$$\mathbf{B} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix},$$

this could be saved as a `numpy` array using
`B = np.array([[1, 2], [3, 4]])`. Save the matrix $\mathbf{A}$ to the variable `A13`.

(g) We are going to solve this system of differential equations in terms of x and $y = x'$ for $0 \leq t \leq 400$. We will use the same initial conditions and $\Delta t = 0.01$.

(h) The Forward Euler formula for a system of linear differential equations is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \mathbf{A}\mathbf{x}_k.$$

Implement the Forward Euler method to solve the IVP for the linear Van der Pol oscillator (3). To do the multiplication $\mathbf{A}\mathbf{x}_k$ in python, use `@`. In other words, if we had the matrix `B` and another array `z = np.array([5, 6])` defined, then $\mathbf{w} = \mathbf{B}\mathbf{z}$ would be `w = B@z`.

Make an array consisting of the forward Euler solution for $x(t)$ over the time interval $0 \leq t \leq 400$. Save this array to the variable `A14`.

(i) The backward Euler formula for a system of linear differential equations is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \mathbf{A}\mathbf{x}_{k+1}.$$

Using some matrix algebra (Week 9!), we can write this as the following linear system

$$(\mathbf{I} - \Delta t \mathbf{A})\mathbf{x}_{k+1} = \mathbf{x}_k,$$

where

$$\mathbf{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

is the 2×2 identity matrix implemented in `numpy` using `I = np.eye(2)`. At each time step of backward Euler, we must solve the linear system (Week 9!)

$$\mathbf{C}\mathbf{x}_{k+1} = \mathbf{x}_k,$$

there $\mathbf{C} = \mathbf{I} - \Delta t \mathbf{A}$. You can do this in python using `np.linalg.solve`. For instance, if I was solving the linear equation $\mathbf{B}\mathbf{u} = \mathbf{z}$, where $\mathbf{B}$ and $\mathbf{z}$ are defined above, I would use `u = np.linalg.solve(B, z)`.

To check your work, create and save the matrix $\mathbf{C}$ to the variable `A15`. Note that you can calculate this once outside of the for loop because it does not depend on the iteration count, k!

Finally, save the Backward-Euler solution $x(t)$ to the array `A16`.

Writeup problems

1. This problem is related to Coding Problem 2. In it, we will compare the built-in explicit solver that implements RK45 `scipy.integrate.solve_ivp`, with the implicit solver, `scipy.integrate.solve_ivp` with `method = 'BDF'`.

   In this part of the problem, we will plot the solution and the time it takes to find the solution, as q changes.

   (a) Using the list of 10 logarithmically spaced points created in Coding Problem 2, solve the resulting ODE and for each q, record:

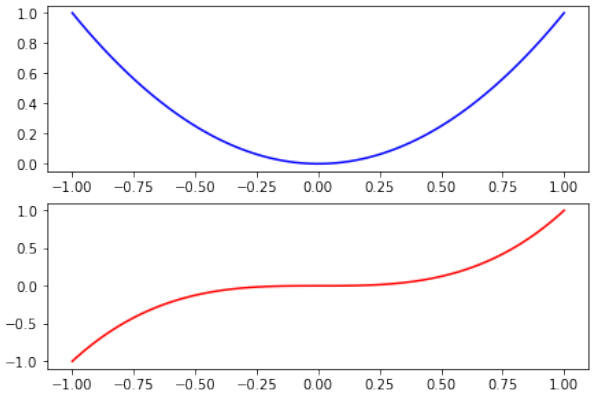
      i. the time it takes to solve the method using RK45; and

      ii. the time it takes to solve using the BDF implicit method.

   (b) Create a `loglog` plot with the logarithmically spaced q values on the horizontal axis and the corresponding times for RK45 and the BDF implicit method on the vertical axis. The times should be plotted as dots of different colors. Your plot should include a legend, axis labels, and title.

   (c) Create one more figure with 2 panels (one column of plots, with 2 rows). On the top, plot y_1 as a function of t for the smallest q value in the list. In the bottom panel, plot y_1 as a function of t for the second largest value of q in the list ($q \approx 10^{-0.5556}$). This is using `fig, ax = plt.subplots(2, 1, constrained_layout=True)`. You do not need a legend or to label the axes (you can add one if you like), but you should title your plot.

   An example figure is below, with the corresponding code in the Writeup Template Jupyter Notebook.



   (d) Comment on the two plots you have created.

      i. Compare the two methods for solving this problem. Which method is better for small q? Is this method always better for solving this problem?

      ii. How does the time for solving using RK45 increase as q decreases? Is it linear, quadratic, exponential? You do not need to, but you may want to use a trendline here to explain your point.

      iii. From the plots of the solution, what makes the solution slower to calculate using RK45 for small q than for larger q?

iv. For small q, is this equation stiff or not? How can you tell from the plot of the solution?
Hint: We have discussed how to tell if an ODE is stiff from the vector field. How does the solution relate to the vector field? Can you tell if a solution corresponds to a stiff vector field?

2. This problem is related to Coding Problem 3.

   (a) First, how many more points were used for the RK45 solution than the BDF solution (`A10`)?

   (b) Using $\mu = 200$, plot the solution $x(t)$ found using RK45 on one figure and using the BDF method on another figure. For these figures you will need to use `t_eval = ...` using enough points to make the plots look smooth. Use blue dots with lines in between for RK45 and red dots with lines in between for BDF. Include axis labels and a title.

   (c) Create another figure plotting $x(t)$ versus $y(t)$ from the BDF solution (again using `t_eval = ...` to make the plot look smooth). Label your axes and give a proper title.

   (d) Using the information above, particularly the ratio of the points used for RK45 and BDF and the shape of the plots, how can we tell that this ODE is stiff?
   Hint: We have discussed how to tell if an ODE is stiff from the vector field. How does the solution relate to the vector field? Can you tell if a solution corresponds to a stiff vector field?