MATH 210 Assignment 6

Solving Systems of ODEs with SciPy

INSTRUCTIONS

- ◇ Create a new Jupyter notebook, set the kernel to Python 3, present your solutions in the notebook and clearly label the solutions
- ◇ There are 3 questions and each is worth 5 points for 15 total points
- ◇ Submit the `.ipynb` file to Connect by **11pm Friday March 18**
- ◇ You may work on these problems with others but you must write your solutions on your own

QUESTIONS

1. Write a function called `mass_spring_damper` with parameters `m`, `c`, `k`, `F`, `y0`, and `t` (in that order) where

   `m` is a positive number, mass

   `c` is a positive number, damping coefficient

   `k` is a positive number, spring constant

   `F` is a function, the forcing function

   `y0` is a list of length 2, initial conditions $[y(t_0), y'(t_0)]$

   `t` is an array of t values where the first entry is t_0 as in the initial conditions

   The function should use `scipy.integrate.odeint` to solve and then plot the solution $y(t)$ of the mass-spring-damper system

   $$my'' + cy' + ky = F(t)$$

   over the t interval defined by the array `t` and given the initial conditions. If any of the input arguments `m`, `c` or `k` are negative, the function should print an message "Error: Negative coefficients" and return `None`.

2. Write a function called `linear_system` with parameters `A`, `x0` and `t` (in that order) where

   `A` is a 2 by 2 NumPy array, coefficient matrix of the linear system

   `x0` is a list of length 2, initial conditions $[x_1(t_0), x_2(t_0)]$

   `t` is an array of t values where the first entry is t_0 as in the initial conditions

The function should use `scipy.integrate.odeint` to plot (in the same figure and including a legend) the solutions $x_1(t)$ and $x_2(t)$ of the linear system of differential equations

$$\frac{d\mathbf{x}}{dt} = A\mathbf{x} \quad , \quad A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad , \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

over the t interval defined by the array `t` and given the initial conditions.

3. Write a function called `improved_euler` with parameters `f`, `y0` and `t` where

   `f` is a function, the right side of a first order scalar ODE $y' = f(y, t)$

   `y0` is a number, the initial condition $y(t_0)$

   `t` is an array of t values such that the first entry is t_0 as in the initial condition

   The function should plot the solution given by the improved Euler method:

$$\begin{aligned} k_{n+1} &= y_n + f(y_n, t_n)(t_{n+1} - t_n) \\ y_{n+1} &= y_n + \frac{f(y_n, t_n) + f(k_{n+1}, t_{n+1})}{2}(t_{n+1} - t_n) \end{aligned}$$

   The function should plot the values y_n versus t_n to approximate the solution.

   The idea with the improved Euler method is to compute the next value y_{n+1} in two stages: compute a first approximation k_{n+1} by the usual Euler method, and then use the average of the slopes $f(y_n, t_n)$ and $f(k_{n+1}, t_{n+1})$ to compute the next value y_{n+1}. Read more about the improved Euler method (aka Heun's method): `https://en.wikipedia.org/wiki/Heun%27s_method`. It is a 2-stage Runge-Kutta method whereas the method behind `scipy.integrate.odeint` is combination of higher order Runge-Kutta methods.