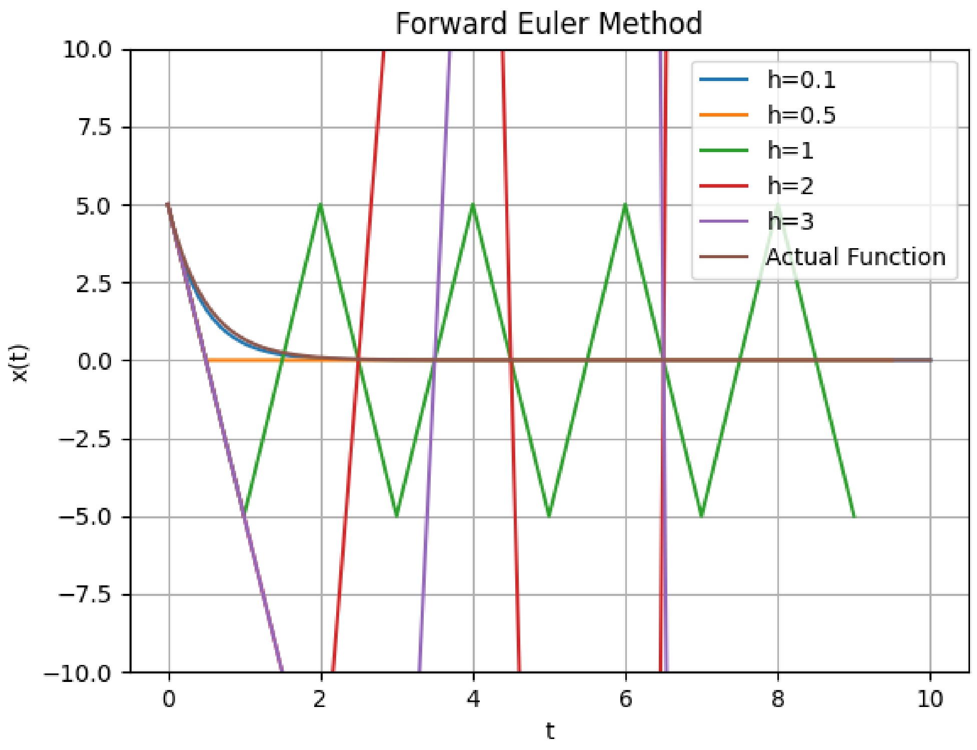
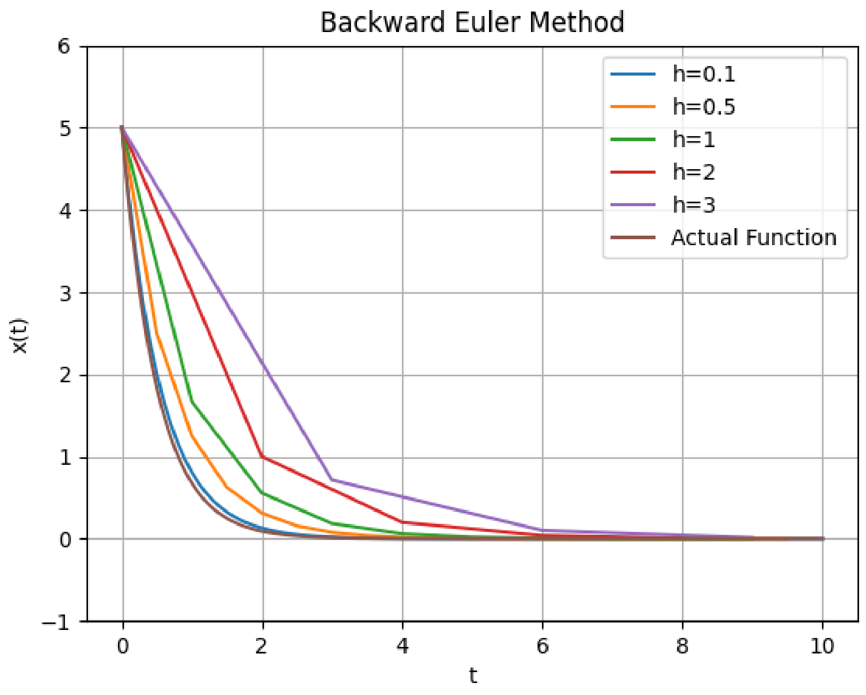
# CS5016: Computational Methods and Applications
# Assignment 6: Ordinary Differential Equations

**Sujit Mandava**
**112001043**

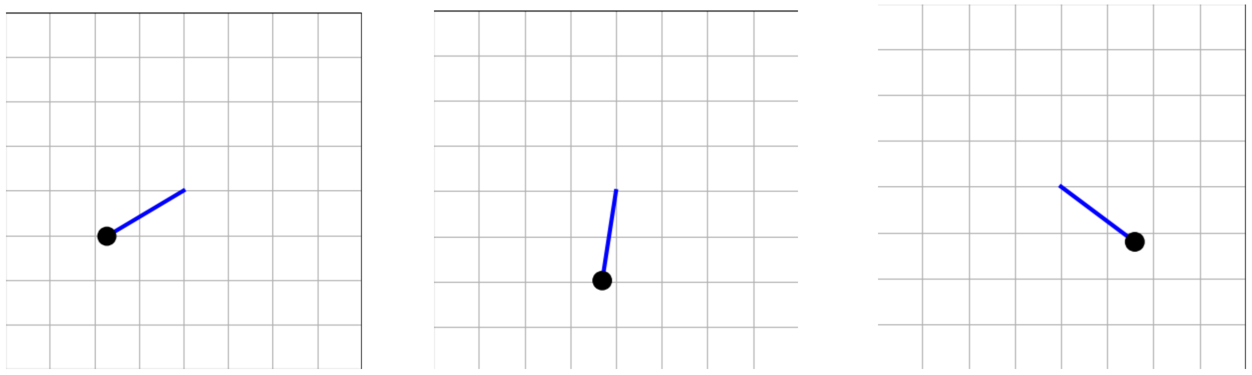
**Q1.** We create a method forwardEuler() that uses the forward Euler method to solve an ordinary differential equation. We create two helper functions: f(x) and ode(t,x) which return the value of the function and the value of the given ordinary differential equation at the given inputs respectively. The forwardEuler() method first initializes the variables and then iterates over the list of step sizes H. At each step size, it computes the solution using the Forward Euler method and stores the computed values of x and t in xList and tList, respectively. It then uses these computed values to plot the approximate solution of the ODE along with the actual solution f(x). The plot obtained looks similar to the following:



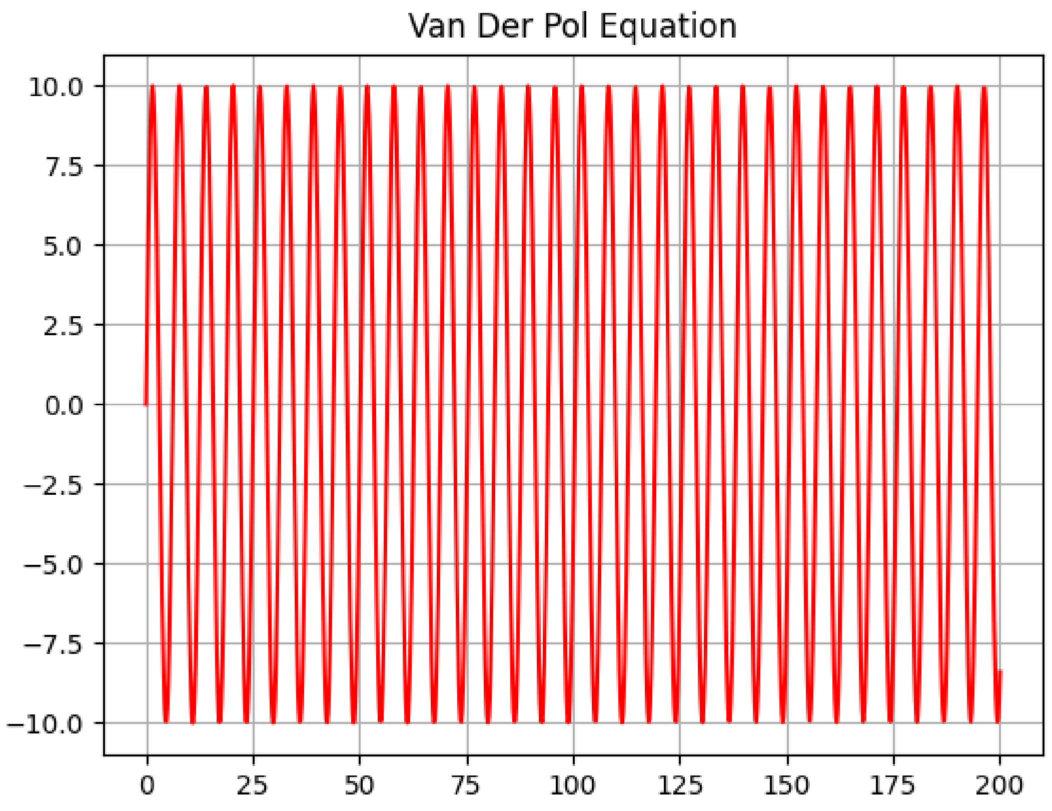
**Q2.** We create a method backwardEuler to solve the given ordinary differential equation (ODE) in the backward Euler method. We create two helper functions: f(x) and ode(t,x) which return the value of the function and the value of the given ordinary differential equation at the given inputs respectively. method uses an iterative numerical algorithm to solve the ODE. It starts with the current value of x and the current time t, and then calculates the next value of x by calling the ode function. The ode function takes in the current time t, the current value of x, and the step size h and returns the new value of x. This process is repeated until the final time T is reached. We plot the value of the x(t) against t for every value of h, and the obtained graph looks similar to:



**Q3.** We simulates the motion of a simple gravity pendulum by solving the given ordinary differential equations (ODEs) and produce an animation describing the pendulum's motion. The ODEs of the simple gravity pendulum are defined in the function ode(), which takes as input the current time, the current angle of the pendulum, the current angular velocity, the acceleration due to gravity, and the length of the pendulum. The function returns the current angular velocity and angular acceleration of the pendulum. The function pendulumODE() then takes the ODEs, the acceleration due to gravity, the length of the pendulum, the initial angular velocity and angle, the start time, the end time, and the step size as input. It uses the forward Euler method to numerically integrate the ODEs and compute the angles of the pendulum at each time step. Finally, we animate the solutions obtained in the previous step using funcAnimation and we obtain an animation similar to:



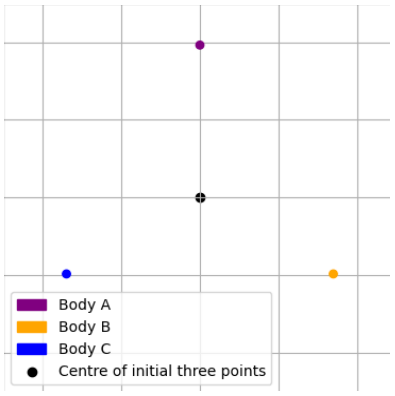
**Q4.** We solve the Van Der Pol equation(non-linear second-order differential equation) to find the time period of the oscillation for a given initial condition and damping parameter. The solveVanDerPol() function takes the initial position, velocity, damping parameter, start time, end time, and the number of time points n as input. We define an ordinary differential equation (ODE) function ivpFunc() that takes time and the position-velocity vector as input and returns the derivatives of the position and velocity. solveVanDerPol() then uses the solve_ivp() function from scipy.integrate module to solve the ODE and obtain the position as a function of time. Finally, we plot the position against time and obtain the following graph:

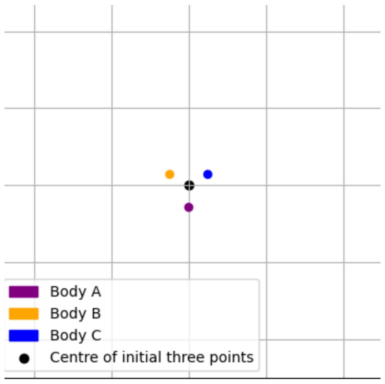


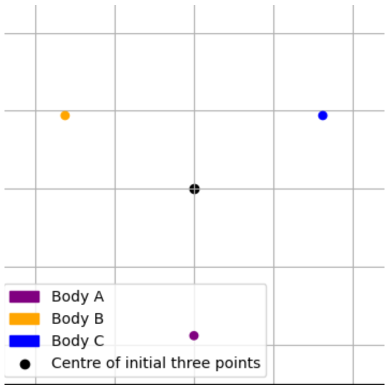
**Q5.** We simulate the Three-Body Problem, a classical problem in physics that involves studying the motion of three bodies under their mutual gravitational attraction. The ODEs are defined by the ivpFunc(), which takes the current time t and the current state of the system y as inputs, and returns the derivatives of the state variables. We define multiple helper functions, such as:

1. norm12(): Calculates and returns the norm between two vectors, or returns a dummy when the norm is zero.
2. rdd(): Calculates the double derivative of a position vector r(t) as governed by the ODEs.
3. calcCentre() Calculates the centroid of the three initial position vectors.

solveODE() takes the initial state of the three bodies and uses the ivpFunc() method to solve the three given ODEs, by virtue of the solve_ivp() function from the scipy.integrate module. Then, it animates the obtained solution using funcAnimation to get an animation that looks similar to:


Body A
Body B
Body C
Centre of initial three points


Body A
Body B
Body C
Centre of initial three points


Body A
Body B
Body C
Centre of initial three points