Computer Modelling for Scientists

CSCI 2202 Assignment 2 Due 14 March 2021 11:30pm

Note: Please put all your programs (each titled `ex_n` n = # of exercise) into a single folder titled: `A2_your_name`, zip it and submit it through Brightspace. You may discuss the problems with anyone, but please do not copy/share/show code from/to each other. Acknowledge any help received (except that from the course instructor and TAs).

(1) (a) **[8]** Solve the non-linear equation: $\lambda \cosh\left(\frac{50}{\lambda}\right) = \lambda + 10$ for λ. The origin of the problem is explained in the notes. $\left(Note: \ \cosh x = \frac{\exp x + \exp -x}{2}\right)$. Choose any method introduced in lecture on root-finding. Specify the number of iterations required to converge. Specify your answer to 4 decimal places.

(b) **[7]** Find the point of intersection of the curves: $y = x^3 - 2x + 1$ and $y = x^2$. Specify your answer to 4 decimal places.

(2) **[15]** Consider a game where the contestant is presented with three doors. Behind one of the doors is a car (the "prize"'), behind the other two are goats. The contestant chooses a door, but the host (who knows where the prize is) opens one of the other two doors (never revealing the prize). The contestant is now given the choice of *switching to the other door*. Create a simulation that plays the game N times (input by the user), using each of the two strategies (**switch door selected** or **not switch door selected**). Estimate the chance of winning for each of the two strategies as a ratio of wins to the total number of attempts N. Run the simulation and report your results for $N = 10000$ and $N = 100000$ games.

(3) **[20]** In **labs 3 and 7** you computed and plotted the trajectory of a projectile, neglecting air-resistance (*drag*) on the projectile. In this problem you will incorporate air resistance. The *drag force* f, for a projectile (like a football or a tennis ball) is proportional to the square of the projectile's speed: $f = Dv^2$ and the direction of f is **opposite** to the direction of the projectile's velocity.

$$\overrightarrow{f} = -Dv\overrightarrow{v} \rightarrow f_x = -Dv \cdot v_x \;\&\; f_y = -Dv \cdot v_y$$

Where f_x, f_y are the components of the drag force and the proportionality constant D is given by:

$$D = \frac{\rho C_D A}{2}$$

where A is the cross-sectional area of the projectile (a ball of mass m), C_D is the drag coefficient (dependent on the shape of the projectile) and ρ is the density of air.

From Newton's second law, the acceleration of the "ball" is proportional to the external unbalanced force: The xv and y component of the net force on the ball are:

$$F_x = -Dv \cdot v_x = ma_x \qquad F_y = -g - Dv \cdot v_y = ma_y$$

The particle moves in the $+x$ and $+y$ direction. The negative signs on the x and y components of the acceleration indicate that these are opposite to the direction of motion. Knowing the mass of the projectile (ball), the acceleration can be computed.

• *(i)* Use `numpy` and `Matplolib` to compute and plot the trajectory of a baseball, projected at a speed and angle input by the user. The program should print the maximum height and the range that the ball reaches. The steps to compute the positions and velocities are given below.
• *(ii)* Plot the trajectory of the baseball with **and** without air resistance on the same plot.

For a regulation baseball, assume $m = 0.145Kg$, the radiu s$r = 0.0366m$, $C_D = 0.5$ and the density of air (at standard temperature and pressure) is $\rho = 1.2kg/m^3$.

(a) Choose a time interval Δt and define the initial values of x, y, v_x, v_y, t. (choose Δt to be a suitable value in the range 0.05 to 0.1).
(b) Choose N - the maximum number of intervals (this gives the max. time: $t_{max} = N * \Delta t$ for the numerical solution). So at the k^{th} timestep: $t_k = k * \Delta t$. Experiment with various values for N and Δt, to find a suitable plot for the trajectory.

(c) Iterate (loop) the next 4 steps repeating the steps *while* $n < N$ or $t < t_{max}$. A convenient way to do this is to use the `range` function in the form `range(1, N+1)`.
(See: `https://docs.python.org/3/library/functions.html#func-range`).

(d) Compute $a_x = -(D/m)v \cdot v_x$ and $a_y = -g - (D/m)v \cdot v_y$

(e) Compute the new velocity components at a time Δt later:
$v_{x-new} = v_{x-curr} + \Delta v_x = v_{x-curr} + a_x * \Delta t$
&
$v_{y-new} = v_{y-curr} + \Delta v_y = v_{y-curr} + a_y * \Delta t$

(f) Compute the new position coordinates at a time Δt later:
$x_{new} = x_{curr} + \Delta x = v_{x-curr} * \Delta t + \frac{1}{2} * a_x * (\Delta t)^2$
&
$y_{new} = y_{curr} + \Delta y = v_{y-curr} * \Delta t + \frac{1}{2} * a_y * (\Delta t)^2$.

(g) Set the current positions and velocities to the new positions and velocities and jump back to step *(d)* and repeat.