CSCI 2202 Computer Modelling for Scientists Due: 10 Feb 2021

Lab 7 Floats, Errors and Plotting

1. Carry out the following in IDLE: $7.8 + 0.2 + 8.9 - 2.1$. What is the answer from Python? What is the exact answer? To get to the bottom of this, compute the *machine epsilon:* ϵ_M.
   Defn: ϵ_M, is the *smallest float*, which added to 1, gives a float greater than 1.

   Write a program to compute ϵ_M. Start with $\epsilon_M = 1$. In each iteration of the loop, halve ϵ_M, add it to 1 and test the result. If the result is different from 1, continue looping.
   The utility of machine epsilon is that it allows us to estimate the distance between any two adjacent float values (in lecture).

2. Floating point operations need to be done with care. An example of a simple computation when things go awry: The Verhulst equation arises from population modelling studies Here we examine compute with the Verhulst equation in two ways: *(We will explore this equation in later labs).*

$$p(n+1) = p(n) + r \cdot p(n) \cdot (1 - p(n))$$

   - Evaluate the equation in two ways:
     (i) As written above.
     (ii) $p(n+1) = (1+r) \cdot p(n) - r \cdot p(n)^2$.
     Use $r = 3$ and use $p(0) = 0.01$ as the *initial condition.* Print the two columns side-by-side with their difference in the third column.
   - **Formatted printing**. Formatting floats, allows us to specify the number of digits **before** and **after** the decimal point: The new version of the format string for floats in Python:
     Assume `p, q` are floats declared in your program. The formatted string:
     `print('p is : {0:12.10f}, q is: {1:8.3f}'.format(p, q))`
     prints *argument 0* (`p`) in 12 digits with 10 digits after the decimal and *argument 1* (`q`) with 8 digits, 3 of them after the decimal. The 'f' tells the interpreter to print the number as a 'float'

     Print `p(n)` calculated the two ways specified with 12 digits after the decimal. When (i.e. at what iteration) do the values start to differ?
     Run your program for 50 iterations and observe the difference in the two output answers.
     The point of this exercise is that the interactions of the slight round-off errors in floating points can be magnified by propagation of the error.

3. In this lab exercise you will re-visit projectile motion (motion in 2-D), plotting the trajectory with a slightly different method.

   The vector equations of motion governing the projectile are:(a - acceleration, $v(t)$ - velocity; s - displacement, t - time):

$$\overrightarrow{v} = \overrightarrow{v_0} + \overrightarrow{a} * t \tag{1}$$

$$\overrightarrow{s} = \overrightarrow{s_0} + \overrightarrow{v_0} * t + \frac{1}{2}\overrightarrow{a} * t^2 \tag{2}$$

   The motion resolved into components is:

x-component	y-component
$v_x = v_0 * \cos\theta + a_x * t$	$v_y = v_0 * \sin\theta + a_y * t$
$x = x_0 + v_0 * \cos\theta * t + (1/2) * a_x * t^2$	$y = y_0 + v_0 * \sin\theta * t + \frac{1}{2}a_y * t^2$

   The Method:

   Use `numpy` for the following exercise. To use `numpy`, first:
   `import numpy as np`
   `import matpoltlib.pyplot as plt`
   Now any `numpy` functions can be used, for example: `np.cos(x*np.pi)`. `numpy` has its own library of functions. In the program use the functions from the `numpy` module.
   `https://numpy.org/doc/stable/reference/routines.math.html`

   (a) Choose a time interval Δt (a value in the range 0.01 to 0.05) and define the initial values of x, y, v_x, v_y, t. Start the projectile off at $(-200, 0)$ with a specified initial velocity and angle.

   (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} time-step: $t_k = k * \Delta t$. Experiment with various values for N and Δt, to find a suitable plot for the trajectory (as an estimate $t_{max} \approx 2 * v_0/g$).

   (c) Note that acceleration is a constant and only acts in the $-y$ direction (*i.e.* $a_x = 0$ & $a_y = -g$)

   (d) Start by setting the *current position* to the initial position $x = -200, y = 0$; with *current velocity* components to the initial velocity components $v_x = v_{0x},\ v_y = v_{0y}$.

   (e) Store the initial values of x, yt in `numpy` arrays.. Also store v_{0x} and v_{0y} in arrays. (see the previous lab for `numpy` arrays).

(f) Next, iterate (loop) over the next 4 steps, repeating the steps *while* $n < N$ (or alternatively $t < t_{max}$). 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`).

(g) Compute the new velocity components at a time Δt later:
$v_x + \Delta v_x = v_x + a_x * \Delta t$ & $v_y + \Delta v_y = v_y + a_y * \Delta t$
(The Δs here indicate change in the variable. For ex. above, $\Delta v_x = a_x * \Delta t$. You have chosen a value for Δt. The same holds for the Δ's below)

(h) Compute the new position coordinates at a time Δt later:
$x + \Delta x = v_x * \Delta t + \frac{1}{2} * a_x * (\Delta t)^2$ & $y + \Delta y = v_y * \Delta t + \frac{1}{2} * a_y * (\Delta t)^2$.

(i) Store the new t, x, y values and the new v_x, v_y values in the arrays.

(j) Set the **current** positions and velocities to the **new** positions and velocities and loop back to step *(g)* and repeat. (In effect, the x, y, v_x, v_y computed in each time-step become the initial value for the next time-step)

(k) Loop over the arrays and determine the max height the projectile reaches: `yMax` and the time (after the start) it reaches this height.

(l) Loop over the arrays and find the x distance the projectile travels (the range): `xMax`. This is the x value corresponding to the y value going to zero. Also print the time at which `xMax` is reached.

(m) Print the three arrays (t, x, y) side-by-side, formatted to have three digits to the left of the decimal point and three digits to the right of the decimal point. for example: `'{:6.3f}'.format(x[i])` will print `x[i]`, as a `float` with 6 digits; 3 of them after the decimal.

(n) Plot y v/s x using `matplotlib.pyplot`. This is a sophisticated plotting package (see `matplotlib.org`).
A very basic plot is made by adding the following: (after the (x, y, t) arrays are computed).

- `plt.plot(x, y, 'b-')` plots a (blue) line through the points (x, y) of the trajectory.
- `plt.xlabel("x")` labels the x axis and similarly `plt.ylabel("y")` the y axis.
- Finally, the last command **must** be `plt.show()`.

- Examples are available at: `https://matplotlib.org/3.1.0/tutorials/introductory/pyplot.html`