PHYC2050 Assignment #5, Winter 2023

Due midnight Wednesday Mar 20

Please submit a single file (jupyter-notebook, or a pdf with your code as snapshots/printout) that has your name in the file name. Remember that suggestions for how to solve a problem are suggestions – there are "many roads to Rome". For each problem, include a test case or two to demonstrate that it is working, and some commentary.

Q1 Peculiar balance

Beta Rabbit is trying to break into a lab that contains the only known zombie cure - but there's an obstacle. The door will only open if a challenge is solved correctly. The future of the zombified rabbit population is at stake, so Beta reads the challenge: There is a scale with an object on the left-hand side, whose mass is given in some number of units. Predictably, the task is to balance the two sides. But there is a catch: You only have this peculiar weight set, having masses 1, 3, 9, 27, ... units. That is, one for each power of 3. Being a brilliant mathematician, Beta Rabbit quickly discovers that any number of units of mass can be balanced exactly using this set.

To help Beta get into the room, write a method called answer(x), which outputs a list of strings representing where the weights should be placed, in order for the two sides to be balanced, assuming that weight on the left has mass x units.

The first element of the output list should correspond to the 1-unit weight, the second element to the 3-unit weight, and so on. Each string is one of:

"L" : put weight on left-hand side
"R" : put weight on right-hand side
"-" : do not use weight

To ensure that the output is the smallest possible, the last element of the list must not be "-".
x will always be a positive integer, no larger than 1000000000.

Test cases

Inputs:
   (int) x = 2
Output:
   (string list) ["L", "R"]

Inputs:
   (int) x = 8
Output:
   (string list) ["L", "-", "R"]

Hint: One easy way to solve this problem is to use recursion. It is equivalent to convert a decimal integer to ternary (base 10 to base 3). A ternary number has 0,1,2 at position i representing 3^i (3 to the power i) . 0 and 1 simply means putting 0 or 1 a weight of mass 3^i on the right; 2 needs some special treatments. 2*3^i =3^(i+1)-3^i, which means putting a weight on the left of the balance (the -3^i part) and adding one to the remaining value to be converted.
At the end, you can use a dictionary to translate 0, 1, 2 to the required strings.

Q2 Integrate with Simpson's rule

Simpson's 1/3 rule uses a parabola to approximate the integrating function within a small interval Δ. It converges faster compared to the trapezoid method with respect to Δ. See the first equation in the link below:

https://en.wikipedia.org/wiki/Simpson%27s_rule

2.1 Derive the summation form of applying Simpson's rule on N equal subdivisions of the integration range [a, b]. As we did for the trapezoid method in class, combine equivalent terms to avoid repeated calculations.

2.2 Implement the above summation in a function.

2.3 Calculate the integral of a Gaussian function $e^{-x^2/2}$ in the range of [0, upper_limit]. Vary the upper_limit from 0 to 50 with an interval of 0.1. Plot the integral as a function of the upper_limit. Set the number of equal subdivisions, N, to 100.

2.4 Repeat 2.3 using the trapezoid method (See Demo_0306/compare_quad.py) and the scipy.integrate.quad function for the integration. The latter can serve as the ground truth.

2.5 Find the upper_limit that gives the maximum difference, error_max, between your Simpson's results and the scipy.integrate.quad's results. Now vary N from 10 to 10000 with an interval of your choice, and plot the error_max as a function of N.

2.6 Repeat 2.5 for the trapezoid method. Compare the error_max(N) plot with that of Simpson.

Q3 Projectile

We know Newton's laws, and we know that the maximum distance a projectile travel (without air resistance) is if it starts at a 45deg angle (from horizontal, in constant gravity). Let's start with that, and then add air resistance.

3.1 Get basic ODE code working for F=ma in 2D. Start with a=F/m, initial speed and angle, and tmax. You can either use the Verlet algorithm from class or call the odeint from scipy.integrate.

3.2 Show numerically that 45deg gives a max distance by looking at 44deg and 46deg and so on..

3.3 Do something adaptive, so that tmax is always large enough for projectile to hit the ground.

3.4 Add air resistance. F_drag = -eta*(v_x, v_y), where eta is the drag coefficient and (v_x, v_y) is the velocity vector. Eta should be provided as a parameter when calling this function.

3.5 Search for the angle that gives max distance (somewhere between 0 and pi/2) for a given eta. Is there a python routine?

3.6 Find the best angles for a range of eta and plot the relationship of the two. You can use these numbers of eta: eta_array= np.logspace(0, 3, 100)

Hint:

F=ma is a 2nd order ODE and can be expressed as a set of first order ODE
$dx/dt = v_x$
$dv_x/dt = F_x/m$
and ditto for y
You also need initial conditions (IC).