AMATH 301 – Winter 2023
Homework 2

Due: 11:59pm, January 20, 2022.

Instructions for submitting:

- Coding problems are submitted to Gradescope as a python script (`.py` file) or a Jupyter Notebook (`.ipynb` file) (warning, this may not work as well as a python script). You have **10** attempts (separate submissions to Gradescope) for the coding problems.
- Writeup problems are submitted to Gradescope as a single `.pdf` file that contains text and plots. Put the problems in order and label each writeup problem. When you submit, **you must indicate which problem is which on Gradescope**. Failure to identify pages on Gradescope will lead to a **5% grading penalty. All code you used for this part of the assignment should be included at the end of your .pdf file. Failure to do so results in a 25% penalty.**

Coding problems

This section is worth 10 points, with each variable having equal weight.

1. Use a for loop to create the sequence of the first 32 Square pyramidal numbers, which are natural numbers counting the number of stacked spheres in a pyramid with a square base (looking at the Wikipedia page may help you understand this). The formula for the number of spheres in a with n layers is

$$P_n = \frac{n(n+1)(2n+1)}{6}.$$

   Save the resulting array to the variable `A1`. The 0th element in your array should be P_1 (plug in $n = 1$ to the above formula).

2. Defining

$$y_1 = \sum_{k=1}^{100{,}000} 0.1, \quad y_2 = \sum_{k=1}^{100{,}000{,}000} 0.1, \quad y_3 = \sum_{k=1}^{100{,}000{,}000} 0.25, \quad y_4 = \sum_{k=1}^{100{,}000{,}000} 0.5,$$

   the following four expressions are exactly equal to zero:

$$x_1 = |10{,}000 - y_1| \qquad x_2 = |y_2 - 10{,}000{,}000|$$
$$x_3 = |25{,}000{,}000 - y_3| \qquad x_4 = |y_4 - 50{,}000{,}000|\,.$$

Since computers store floating-point numbers (numbers with decimals) with a finite storage space, most numbers are stored with a small truncation error (recall when we did $\sin(\pi)$ in class and found a number $\times 10^{-16}$). Usually this error is too small to worry about, but it accumulates if you add up the number multiple times. In this problem, and the accompanying writeup problem, we are going to verify and explore this effect.

Use a `for` loop in python to first compute $y_1,\ y_2,\ y_3$, and y_4.
Note: You should be adding in your `for` loop, this is a sum after all. You may want to look at the January 13 lecture notes for examples on how this is done. These sums are easy to calculate using mathematical formulae, but that's not what we want to explore here. You should do no simplification to the sums before computing them. If you find that all of the x_j are zero below, then you are not adding things in the sum correctly.

(a) Compute $y_1,\ y_2,\ y_3$, and y_4 and save them to the variables `A2`, `A3`, `A4`, and `A5` respectively.

(b) Once you have computed y_1–y_4, compute $x_1,\ x_2,\ x_3$, and x_4 and save them to the variables `A6`, `A7`, `A8`, and `A9` respectively.

3. In this problem we will find all Fibonacci numbers that are *less than 1,000,000*. We will do so using a `for` loop and an `if` statement.

(a) Create an array, `Fibonacci`, with 200 zeros. This is the initialization step, we will fill this in with Fibonacci numbers.

(b) Initialize the first two Fibonacci numbers as 1 and 1, as done in lecture.

(c) Now use a `for` loop and an `if` statement to compute all of the Fibonacci numbers that are *less than 1,000,000*. Save these to `Fibonacci` and save the resulting array with 200 elements to the variable `A10`.
Hint: You may want to use a `break` statement.

(d) The largest Fibonacci number less than $1,000,000$ occurs at some $N < 200$. Use an `if` statement in your for loop to calculate that N and save it to the variable `A11`. Remember that indexing begins at 0.
Note: Don't do this by printing the array and manually counting. Instead, practice finding this *programatically*.

(e) Now select only the first N elements from the array `Fibonacci` so that you are left with only those Fibonacci numbers that are less than 1,000,000. Save your answer to the variable `A12`.
Hint: Slicing may be helpful here

4. The *n-th order Taylor-series approximation* to cosine is given by

$$T_n = \sum_{k=0}^{n} \frac{(-1)^k}{(2k)!} x^{2k},$$

because $\cos(x) = \lim_{n\to\infty} T_n$. In this problem you will calculate the third-order Taylor-series approximation, i.e.,

$$T_3 = \sum_{k=0}^{3} \frac{(-1)^k}{(2k)!} x^{2k} = 1 - \frac{x^2}{2} + \frac{x^4}{4!} - \frac{x^6}{6!}.$$

(a) Create the array `x` with 100 equally spaced points in the interval $[-\pi, \pi]$. Save that array to the variable `A13`.

(b) Use a `for` loop to calculate the third-order Taylor-series approximation to cosine using the sum formula above. You should **not** be writing each term, instead use a `for` loop to calculate the sum! Save the result, an array with 100 elements, to the variable `A14`.

Writeup problems

This section is worth 10 points.

You may want to start a new python script for the writeup portion of the assignment, or you may want to use a Jupyter notebook.

1. Consider Coding Problem 2.

   (a) Record the values of x_1, x_2, x_3, and x_4 you found in Coding problem 2 and rank them in order from smallest to largest.

   (b) Discuss what you found in part (a). Why do you think some are larger than others? Are there any outliers to your conclusion? Your discussion should include, at least, a comparison between x_1 and x_2 and also a comparison between x_2 and x_3.

   Saying "this one is larger than that one" in the discussion is not enough. You should discuss why one would be larger than the other."

   (c) Which of your answers is exactly zero. Can you guess why that is the case?

   You will not be graded on whether you have this right, but you should have something written here.

   Hint: Think about the denominators if you were to write the decimal as a fraction, then note that computers store information in base 2, e.g., $1/2 = 0.5 = 2^{-1}$.

2. The *n-th order Taylor-series approximation* to cosine was given in coding Problem 4.

   Create a plot of cosine on the interval $[-\pi, \pi]$ and plot the Taylor-series approximations for $n = 1, 3, 14$. You should **not** be writing these sums out by hand. Instead, use what we have talked about in class for calculating sums (for loop).

Each plot should be on the same axis in the same figure. In order to receive full credit, your plot must have the following features:

(i) The plot of $\cos(x)$ should be a solid black line of width 2.

(ii) The plot of the $n = 1$ Taylor approximation should be a blue dashed line of width 2.

(iii) The plot of the $n = 3$ Taylor approximation should be a red dot-dashed line of width 2.

(iv) The plot of the $n = 14$ Taylor approximation should be a magenta dotted line of width 2.

(v) The x-axis should be labeled "x-values".

(vi) The y-axis should be labeled "$\cos(x)$ approximations".

(vii) There should be a title with the text "$\cos(x)$ and its Taylor approximations."

(viii) There must be a legend with 4 entries (one for each plot). The legend should be in a place that does not block the plots (or at least minimizes the amount blocked).

(ix) The font on your axes and title should be large enough to read without zooming in: a font size of about 10 is good.

An example of what the figure should look like (without all of the required plots) is below.