# Math 5490 – Homework 1: Due Jan 25 by 11:59pm

## Instructions:

- Complete the problems below, and submit your solutions and Python code by uploading them to the Google form for homework 1: https://forms.gle/uUyjvLKuoX8i9j3G8
- Submit all your Python code in a single .py file using the function templates given in each problem. I will import your functions from this file and test your code.
- If you use LaTeX to write up your solutions, upload them as a pdf file. Students who use LaTeX to write up their solutions will receive bonus points on the homework assignment (equivalent to 1/3 of a letter grade bump).
- If you choose to handwrite your solutions and scan them, please either use a real scanner, or use a smartphone app that allows scanning with you smartphone camera. It is not acceptable to submit photos of your solutions, as these can be hard to read.

## Problems:

1. Check that you are registered for the Piazza site https://piazza.com/umn/spring2023/math5490 and make a post to the whole class. Tell us something about yourself.

2. Write a Python function that computes the square root of a positive number using the Babylonian method. The Babylonian method to compute $\sqrt{S}$ for $S > 0$ constructs the sequence $x_n$ by setting $x_0 = S$ and iterating

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right).$$

   Your code can take as input a tolerance parameter $\varepsilon > 0$, and should iterate until $|x_n^2 - S| \leq \varepsilon$, and then return $x_n$. Test your square root function to make sure it works.

   When submitting your code, use the function template

```
def babylonian_sqrt(S,eps=1e-6):
   return x
```

   Your function should use only basic Python programming. In particular, do not use any packages, like Numpy, Scipy, etc.

3. Prove that the iteration in the Babylonian method above converges quadratically to the square root of $S$. In particular, show that the error $\varepsilon_n = \frac{x_n}{\sqrt{S}} - 1$ satisfies

$$\varepsilon_{n+1} = \frac{\varepsilon_n^2}{2(\varepsilon_n + 1)}.$$

   From this, show that $\varepsilon_n \geq 0$ for $n \geq 1$, and so

$$\varepsilon_{n+1} \leq \frac{1}{2}\min\{\varepsilon_n^2, \varepsilon_n\}.$$

4. Write a Python function that computes the cross product of two 3-dimensional vectors $\mathbf{x} = (x_1, x_2, x_3)$ and $\mathbf{y} = (y_1, y_2, y_3)$. The cross product $\mathbf{z} = \mathbf{x} \times \mathbf{y}$ is given by

$$\mathbf{z} = (x_2y_3 - x_3y_2, x_3y_1 - x_1y_3, x_1y_2 - x_2y_1).$$

When submitting your code, use the function template

```
def cross_product(x,y):
   return z
```

Your function should use only basic Python programming. In particular, do not use any packages, like Numpy, Scipy, etc. The inputs `x` and `y` should be Python lists containing the elements of each vector, and your code should return a Python list `z` containing the cross product $\mathbf{z} = \mathbf{x} \times \mathbf{y}$.

5. Write a Python function that computes the largest magnitude eigenvalue and corresponding eigenvector of a square matrix with the power iteration. The power iteration is

$$\mathbf{x}_{n+1} = \frac{A\mathbf{x}_n}{\|A\mathbf{x}_n\|}.$$

For a diagonalizable matrix, the iteration converges to the eigenvector of $A$ with largest magnitude eigenvalue. The eigenvalue is

$$\lambda = \lim_{n\to\infty} \mathbf{x}_n^T A\mathbf{x}_n.$$

Compare your function to the true eigenvector and eigenvalue for small matrices where you can compute it by hand, to check that your function works. For a stopping condition, compute the residual vector

$$\mathbf{r}_n = A\mathbf{x}_n - (\mathbf{x}_n^T A\mathbf{x}_n)\mathbf{x}_n$$

and run the iterations until $\|\mathbf{r}_n\| \leq \varepsilon$, where $\varepsilon > 0$ is a given tolerance parameter.

In this exercise you may use Numpy. When submitting your code, use the function template

```
def power_method(A,eps=1e-6):
   return l, x
```

The input `A` should be a square NumPy array containing the elements of the matrix `A`, and the returns are the eigenvalue `l`, and a numpy array `x` containing the eigenvector. Try to write your code with only one loop, over the iterations in the power method, and make use of NumPy matrix vector multiplication and NumPy functions for the norm.