1 Submission Instructions

The project must be submitted through BBLearn as a Jupyter notebook. Create a new notebook in Google Colab. Please name your notebook as follows:

Project 1 - Your Name - Your CSU ID

For example, Alice Mathperson with ID 1234567 would name her notebook:

Project 1 - Alice Mathperson - 1234567

You are allowed to make multiple submissions in BBLearn. However, only the final submission will be graded. In your final submission, make sure of the following:

- The notebook must run without errors. Before submitting the notebook, check that all cells execute correctly, *in the order they appear in the notebook*.
- There is one exception to the rule above. It is often useful to test code we know to produce errors, so that we know how our code reacts to mistakes made by the user. In this case, provide an explanation of the code being tested.
- Add text cells to explain what you are doing in each step, and state your conclusions. Your project should be readable as a mathematical paper.
- Your code should not only be correct, but well structured and clear. Use meaningful variable names and add comments where you feel necessary.

2 Project Statement

In this project we will explore a modern algorithm to approximate π based on the *arithmetic-geometric mean* iteration. This algorithm has been used to compute π to a precision of millions of decimal places.

The algorithm is defined in terms of three sequences defined recursively as follows:

1. Choose two real numbers a_0 and b_0 such that $a_0 > b_0 > 0$, and let $c_0 = \sqrt{a_0^2 - b_0^2}$.
2. For $n \geq 1$, let:

$$a_n = \frac{1}{2}(a_{n-1} + b_{n-1}, \quad b_n = (a_{n-1}b_{n-1})^{\frac{1}{2}}, \quad c_n = \frac{1}{2}(a_{n-1} - b_{n-1})$$

Then, the sequences (a_n) and (b_n) converge to a common value, called the *arithmetic-geometric mean* of a_0 and b_0:

$$\text{agm}(a_0, b_0) = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n.$$

Then, we have the following remarkable formula for π:

$$\pi = \frac{4\left(\text{agm}\left(1, 2^{-1/2}\right)\right)^2}{1 - \sum_{j=1}^{\infty} 2^{j+1} c_j^2}$$

2.1 Part 1

Write code that uses the formula above to compute an approximation for π. To obtain the approximation, we *truncate* the formula for π. More precisely, we have the following estimates:

$$\frac{4(a_n)^2}{1-\sum_{j=1}^{n} 2^{j+1}c_j^2} > \pi > \frac{4(b_n)^2}{1-\sum_{j=1}^{n} 2^{j+1}c_j^2}$$

One possible way to organize the computation is the following:

```
# Initialization
a, b = ... # Initialize a, b
c = ...    # Initialize c
pwr2 = ... # Initialize powers of 2
acc = ...  # Initialize accumulator for the sum
maxn = ... # Maximum number of iterations
tol = ...  # Required tolerance for approximation
for n in range(maxn):
    # update a, b, c, pwr, acc
    # Compute lower and upper bounds for pi
    if upper_bound - lower_bound < tol:
        break
# Print results, error messages, etc.
```

Note: This is just a suggestion of how to set up the code. Feel free to modify it if you find a better solution!

In this version of the code, use either the built-in Python mathematical functions or `numpy` (here it does not really make a difference, since we are not using any vectorization).

To check correctness of the code, compare the result with the approximation for π given by Python (either `math.pi` or `numpy.pi`).

2.2 Part 2

Let's now compute a higher precision of π. To achieve this, we can use the `decimal` module, which is part of the Python standard library. The `decimal` module provides support to arbitrary precision computations with *decimal* numbers (as opposed to the *binary* representation that is the default).

Do the following:

- Read the documentation of the `decimal` module, available at the link:

  https://docs.python.org/3/library/decimal.html

- Modify your code to approximate π so that it uses the `decimal` module.

- Use your code to compute an approximation to π that is correct to 1000 significant digits.

- Find a high-precision approximation for π and use it to check your result.
- What is the highest precision you can get with your code, in a reasonable time?