Math 351 Coding Project: Arctangent to Machine Precision

Objective

Write Python code that computes the arctangent of a list of **one million real numbers** to **machine precision**, using only **elementary operations**.

Constraints

- No external libraries (e.g., `numpy`, `math`, `scipy`, etc.).
- You may use only elementary operations: addition, subtraction, multiplication, division,.
- No precomputed real numbers (including π) but you definitely should precompute things like Taylor series which will be built in the code
- I will generate a list 1,000,000 floating-point numbers from the interval $[-5, 5]$ using the uniform distribution.
- The code must process a list of **1,000,000 floating-point numbers** and return their arctangent values with **machine precision** (i.e., within approximately 15 decimal digits of accuracy).
- The code should include one function that I can run in my python, after I generate the list of 1 million numbers. Prior to running this function, you can define other functions, but these must be called within the body of the main function. I will start a clock, pass the list to the main function, which returns a list of the arctans of the data. Then I stop the clock. I will also check to make the numbers computed are correct.
- The goal is to win, that is, have the fastest code, without making any mistakes.

Hints

- There is considerable strategy here that you can understand by playing around. You don't want to compute the Taylor series directly for every value, but you may want to fix a number of data points and compute the Taylor series for that, and use polynomial interpolation for most of the data. The order of interpolation is up to you. The series you choose depends on where it converges - Don't use the wrong one!

- If you're confident that your number is correct to machine precision, you don't need to test it (testing adds time.) You do have to be careful near zero.
- For values outside the convergence radius of your chosen method, use identities such as:
$$\arctan(x) = \frac{\pi}{2} - \arctan\left(\frac{1}{x}\right) \quad \text{for } x > 0$$
$$\arctan(x) = -\frac{\pi}{2} - \arctan\left(\frac{1}{x}\right) \quad \text{for } x < 0$$
- It's up to you how you get the number π if you end up using it.
- You can Taylor expand about any number, but this requires you to sit down and compute a formula for the expansion.

Evaluation Criteria

- **Correctness**: Are the results accurate to machine precision?
- **Efficiency**: Does the code run in reasonable time on large input?
- **Elegance**: Is the code well-structured and readable?
- **Compliance**: Does the code meet all constraints?

Submission

Submit a single `.py` file containing your function.. Include a brief comment at the top describing your approach and any precomputed data.

You may work in groups of up to three if you choose.

Due Dec 5

Basic Format

Below is a basic structure your code might follow. You are free to organize it differently, as long as it meets the project constraints.

```
def Taylor(x):
   # does stuff
   return(value)


def polynomial(x,coeffs):
   # does stuff
   return(value)

def get_pi():
   # does stuff
   return(value)


def MyArctan(data_list):
   #does stuff
   #runs other functions or whatever
  #processes the arctans of the list
  return(arctan_list)
```

What I will do with your code, is to run everything above, and then

```
data_list =  [random.uniform(-5, 5) for _ in range(10**6)]  # this will be the same for
precise_arctans = [np.arctan(x) for x in data_list]


start = time.time()
output =  MyArctan(data_list):
end = time.time()
end - start

error = [precise_arctans[i] - output[i] for i in range(10**6)]
```