Putting it All Together:
Running C++ Code in Python and Getting a Speed Boost Along the Way!

Justin Hart

June 2025

1 Introduction

One of the biggest complaints about Python is that it's slow! However, the great thinkers at Python Incorporated have thought of this, and have given you a solution. Be it Numpy, Scipy, Pandas, one that nobody uses, there are plenty of math libraries that use C, C++, or Fortran(!) as a backend and allow you to run your slow calculations in the speedy distributions you've come to love over the course of this course!

In this project, you will get to exercise your function writing and software engineering skills as you do two things, first implement 20 mathematical functions in C++ and then learn how to link it to Python, creating a package you could use in your own line of work! Additionally, we will provide you with some of these methods written solely in Python, and you'll get to benchmark the speed differences!

In the benchmark file, you will be able to call [your_name].func() to run your calculations, for example:

```
import time

start = time.perf_counter()

for i in range(1000):   #run 1000 times to show difference on scale
        justinPy.sin(3.14159265)

end = time.perf_counter()
```

2 Student Expectations

2.1 Topics Covered

1. Libraries/Linking

2. Errors
3. Functions
4. Data Types
5. Testing

2.2 Deliverables

Working Python Module which runs C++ code and a test.py file.

2.3 Rubric

See rubric.png.

3 Items Provided to Student

1. List of Mathematical Functions to implement, and the equations behind them. See equations.pdf
2. List of test cases to implement. See test_cases.pdf
3. Separate python module which implements all mathematical functions purely in Python
4. Read the *Friendly* Manual

   https://docs.python.org/3/tutorial/modules.html

   https://docs.python.org/3/extending/extending.html

   https://www.gnu.org/software/libc/manual/html_node/Mathematics.html

4 TA Tools

The TA will be provided a ta_test.py file which runs the code, and checks that the output values of each of the 20 mathematical functions are the same as expected, in a set error range. The example below is a minimal product and can have far better output for the TA to grade on.

Additionally, the test cases should check error messages that the student should implement, such as division by 0.

```
import student_code as sc
import numpy as np
import math

try:
    student_value = [sc.sin(np.pi), sc.sin(np.pi/6), sc.sin(0)]
```

```
    expected_value = [1, 0.5, 0]
    epsilon = 0.0001

    for i, value in enumerate(student_value):
        if math.abs(value - expected value[i]) > epsilon:
                raise Exception(f"Test failed on expected value of {expected_
        else:
                pass

    print('Test case sc.sin succeeded.')
except Exception as e:
    print('Test case sc.sin failed.')
```

Finally, the TA must evaluate the test.py file, which should have standardized output by the student and will have 6 test cases. The TA will also have their own test.py file to compare against to see if outputs are realistic.

5 Inspiration

Credit to Matt Fowler for creating a GitHub project demonstrating the speedup from C++ code, and the Lehigh University High Performance Computing documentation for highlighting this example.