Assignment 6

Dhruv Prasad

October 27, 2024

Introduction

This report describes the implementation and optimization of the trapezoidal rule in Cython, comparing performance and accuracy with pure python and numpy.

Cython Implementation

The Cython function, `cy_trapz(f, a, b, n)` uses:

- `cdef` for static typing of variables,
- `cpdef` to specify the return type of the function while keeping it as a python object so it can be called in other modules
- `@cython.cdivision(True)` to skip checking division by 0.

```
%%cython --annotate
# So that we can see the C code generated corresponding to each line of co

import cython
@cython.cdivision(True) # Decorator to skip devide by zero error catching
cpdef double cy_trapz(f, double a, double b, int n):
    cdef double h = (b - a)/n
    cdef double area = 0.5*(f(a)+f(b))
    cdef double x=a+h
    for i in range(n-1):
        area += f(x)
        x+=h
    return area*h
```

Performance Analysis

Integration Results

Implementation	Calculated Value	Analytical Value	Absolute Error	Execution Time
`py_trapz`	0.33333333499994316	0.333...	1.67e-9	928 µs
`cy_trapz`	0.33333333499994316	0.333...	1.67e-9	710 µs
`np_trapz`	0.33333333500000006	0.333...	1.67e-9	1.26 ms

Table 1: Integration results for $f(x) = x^2$ over [0, 1].

Implementation	Calculated Value	Analytical Value	Absolute Error	Execution Time
`py_trapz`	1.9999999835503668	2.0	1.64e-8	1.31 ms
`cy_trapz`	1.9999999835503668	2.0	1.64e-8	1.2 ms
`np_trapz`	1.999999983550659	2.0	1.64e-8	1.78 ms

Table 2: Integration results for $f(x) = \sin(x)$ over [0, π].

Implementation	Calculated Value	Analytical Value	Absolute Error	Execution Time
`py_trapz`	1.718281829890868	1.718281828459045	1.43e-9	1.26 ms
`cy_trapz`	1.718281829890868	1.718281828459045	1.43e-9	1.15 ms
`np_trapz`	1.7182818298909468	1.718281828459045	1.43e-9	1.72 ms

Table 3: Integration results for $f(x) = e^x$ over [0, 1].

Implementation	Calculated Value	Analytical Value	Absolute Error	Execution Time
`py_trapz`	0.6931471811849667	0.6931471805599453	6.25e-10	945 µs
`cy_trapz`	0.6931471811849667	0.6931471805599453	6.25e-10	876 µs
`np_trapz`	0.6931471811849453	0.6931471805599453	6.25e-10	1.38 ms

Table 4: Integration results for $f(x) = \frac{1}{x}$ over [1, 2].

Findings and Observations

Accuracy

Accuracy remained similar across implementations.

Performance

As we can see, the cython optimizations helped to increase the speed slightly.

Interestingly, numpy performed the worst. This is because numpy is optimized for applying functions over large number of inputs in arrays. By removing the np.vectorize line and directly applying numpy functions, we get far lower execution times.

This does make the other two implementations slower, as numpy functions are not optimized and slow when we apply them on scalar values, as we can see below.

Integration Results

Implementation	Calculated Value	Analytical Value	Execution Time
`py_trapz`	0.33333333499994316	0.3333333333333333	7.16 ms
`cy_trapz`	0.33333333499994316	0.3333333333333333	7.65 ms
`np_trapz`	0.33333333500000006	0.3333333333333333	47.8 µs

Table 5: Integration results for $f(x) = x^2$ over [0, 1].

Implementation	Calculated Value	Analytical Value	Execution Time
`py_trapz`	1.9999999835503668	2.0	7.37 ms
`cy_trapz`	1.9999999835503668	2.0	8.06 ms
`np_trapz`	1.999999983550659	2.0	154 µs

Table 6: Integration results for $f(x) = \sin(x)$ over [0, π].

Implementation	Calculated Value	Analytical Value	Execution Time
`py_trapz`	1.718281829890868	1.718281828459045	7.61 ms
`cy_trapz`	1.718281829890868	1.718281828459045	7.67 ms
`np_trapz`	1.7182818298909468	1.718281828459045	112 µs

Table 7: Integration results for $f(x) = e^x$ over [0, 1].

Implementation	Calculated Value	Analytical Value	Execution Time
`py_trapz`	0.6931221811849666	0.6931471805599453	7.12 ms
`cy_trapz`	0.6931471811849667	0.6931471805599453	7.93 ms
`np_trapz`	0.6931471811849453	0.6931471805599453	51.1 µs

Table 8: Integration results for $f(x) = \frac{1}{x}$ over [1, 2].

Findings and Observations

Accuracy

Accuracy remained similar across implementations.

Performance

Numpy is now by far the fastest.