Estimation of π using Numerical Integration method in Python: A comparative analysis of the different implementations

Ángel Adrián Campos Borges
Data Engineering,Universidad Politécnica de Yucatán
Km. 4.5. Carretera Mérida- Tetiz Trabaje Catastral 448. CP97357 Ucú, Yucatán.

2109021@upy.edu.mx

***Abstract* – This report presents an estimate of the value of π using the Numerical Integration method in Python. Three different versions of the method are implemented, each with different Python functions; pi_riemann(), pi_riemann_parallel(), pi_riemann_mpi() and a comparative analysis of their performance is performed.**

***Keywords* – High-Performance Computing (HPC), Data Science, Engineering, Numerical Integration method, Python, Python Processing, Computational Resources, Parallel Processing, Scalability, Efficiency.**

I. State of the Art

Numerical integration. In mathematics, the study of integrals is fundamental since it solves many problems in daily life. This branch is studied in Infinitesimal Calculus as a defined integral function f(x) in an interval [a, b] as the result of a process limit of a finite sum. The Monte Carlo method has been widely used to estimate the value of π. One of the simplest ways to do this is using the circle inscribed in a square method. In this method, a square of side 1 is generated and a circle is inscribed inside it. The value of π can be estimated as the ratio of the area of the circle to the area of the square.

Numerical integration has a wide range of applications in various fields, from science and engineering to economics and finance. Here are some examples:

1. Calculation of areas and volumes: Area under a curve: Numerical integration is used to approximate the area enclosed between a curve and the x-axis. This is useful in fields such as physics to calculate the work done by a force or in economics to determine the total demand for a good.

2. Calculation of centers of mass and moments: Centers of mass: Numerical integration is used to calculate the center of mass of irregular objects or continuous mass distributions. This is relevant in engineering for the design of structures and mechanisms.

3. Solving differential equations: Ordinary differential equations (ODEs): Numerical integration is a fundamental method for solving ODEs, which describe the behavior of dynamic systems in time. It is used in areas such as physics, engineering, biology and economics.

4. Calculation of probabilities: Probability integrals: Numerical integration is applied to calculate probabilities of random events. This is useful in statistics and in areas such as risk analysis and decision making.

5. Evaluation of financial models: Option Pricing: Numerical integration is used to calculate the price of financial options, such as call and put options, which are complex financial instruments.

6. Simulation of physical systems: Particle Motion: Numerical integration is used to simulate the motion of particles in force fields, such as in molecular physics or fluid dynamics.

7. Calculation of satellite trajectories: Space orbits: Numerical integration is used to calculate the trajectories of satellites and spacecraft in space, considering the influence of gravity and other factors.

8. Signal analysis: Image Processing: Numerical integration is applied in image processing to perform tasks such as convolution and image filtering.

9. Tomography reconstruction: Medical Imaging: Numerical integration is used in CT scans to reconstruct three-dimensional images from two-dimensional projections.

10. Calculation of fractals: Fractal geometry: Numerical integration is used to calculate the areas and dimensions of

fractals, which are mathematical structures with complex geometric properties.

But the application we will see is the calculation of areas under a curve to determine the value of PI (π).

II. INTRODUCTION; ESTIMATION OF π USING THE NUMERICAL INTEGRATION METHOD

The number pi (π) can be calculated using Riemann sums, a mathematical technique for approximating the area under a curve. The process involves dividing the interval of integration into smaller subintervals and adding the areas of rectangles or geometric figures that fit those subintervals.

Estimation of π using Numerical Integration method in Python: General steps:

- Define the function: The function to integrate is f(x) = 1/√(1-x^2), which represents the shape of a quarter circle with radius 1. The integration interval is [0, 1] , since it covers a full quarter circle.
-
- Select the type of Riemann sum: There are different types of Riemann sums, such as the sum of left and right rectangles, or trapezoids. For this case, we will use the sum of left rectangles.
-
- Divide the integration interval: We divide the interval into n subintervals of equal width. The width of each subinterval will be Δx = (b - a) / n = (1 - 0) / n = 1/n.
-
- Calculate the areas of rectangles: For each subinterval, we calculate the area of the rectangle with base Δx and height f(a_i), where a_i is the left end of subinterval i.
-
- Adding the areas of the rectangles: The approximation of the area under the curve (and therefore, the value of π) is obtained by adding the areas of all the rectangles: Approximation of π ≈ Σ[f(a_i) * Δx].
-
- Increase the number of subintervals: As we increase the number of subintervals (i.e., n), the approximation of the value of π becomes more precise.

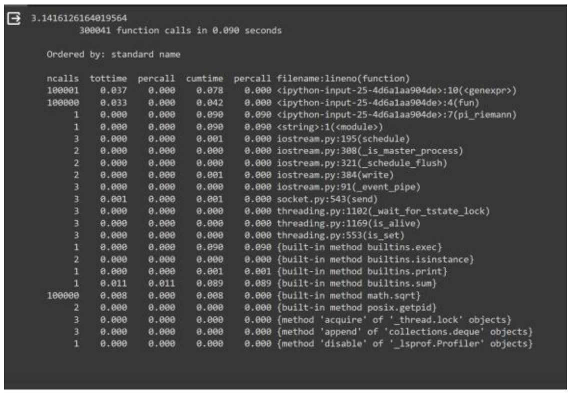
III. SOURCE CODE

Below is the python code, called "**PI via Numerical Integration.ipynb**", which contains the approximation to π by numerical integration method in different versions, unparallelized vs parallelized:

Pi approximation by numerical integration unparallelized:

```python
#Aproximacion de PI sin parallelizacion
from math import sqrt
import cProfile

def fun(x):
  return sqrt(1-x**2)

def pi_riemann(n):
  dx = 1/n
  sr = 0
  sr = sum(fun(i * dx) * dx for i in
range(n))
  print(4*sr)

#pi_riemann(100000)

# Run the profiling
cProfile.run('pi_riemann(100000)')
```

Below is the profiling of the code, which contains the approximation to π by numerical integration unparallelized:



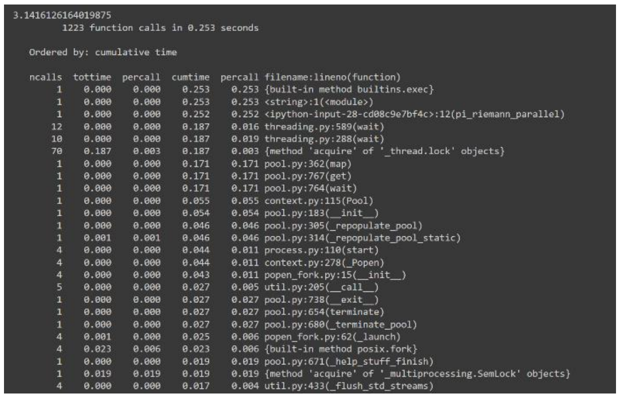
```
3.1416126164019564
         300041 function calls in 0.090 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   100001    0.037    0.000    0.078    0.000 <ipython-input-25-4d6a1aa904de>:10(<genexpr>)
   100000    0.033    0.000    0.042    0.000 <ipython-input-25-4d6a1aa904de>:4(fun)
        1    0.000    0.000    0.090    0.090 <ipython-input-25-4d6a1aa904de>:7(pi_riemann)
        1    0.000    0.000    0.090    0.090 <string>:1(<module>)
        3    0.000    0.000    0.001    0.000 iostream.py:195(schedule)
        2    0.000    0.000    0.000    0.000 iostream.py:308(_is_master_process)
        2    0.000    0.000    0.000    0.000 iostream.py:321(_schedule_flush)
        2    0.000    0.000    0.001    0.000 iostream.py:384(write)
        3    0.000    0.000    0.000    0.000 iostream.py:91(_event_pipe)
        3    0.001    0.000    0.001    0.000 socket.py:543(send)
        3    0.000    0.000    0.000    0.000 threading.py:1102(_wait_for_tstate_lock)
        3    0.000    0.000    0.000    0.000 threading.py:1169(is_alive)
        3    0.000    0.000    0.000    0.000 threading.py:553(is_set)
        1    0.000    0.000    0.090    0.090 {built-in method builtins.exec}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.isinstance}
        1    0.000    0.000    0.001    0.001 {built-in method builtins.print}
        1    0.011    0.011    0.089    0.089 {built-in method builtins.sum}
   100000    0.008    0.000    0.008    0.000 {built-in method math.sqrt}
        2    0.000    0.000    0.000    0.000 {built-in method posix.getpid}
        3    0.000    0.000    0.000    0.000 {method 'acquire' of '_thread.lock' objects}
        3    0.000    0.000    0.000    0.000 {method 'append' of 'collections.deque' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Pi approximation by numerical integration parallelized:

```python
#Aproximacion de PI CON parallelizacion
from math import sqrt
import multiprocessing
import cProfile
```

```
def fun(x):
    return sqrt(1 - x**2)

def partial_sum(args):
    #xi = i * dx
    n, start, end, dx = args
    return sum(fun(i * dx) * dx for i
in range(start, end))

def pi_riemann_parallel(n,
num_processes=4):
  pool = multiprocessing.Pool()
  dx = 1 / n
  ranges = [(n, int(i * n /
num_processes), int((i + 1) * n /
num_processes), dx) for i in
range(num_processes)]

  results = pool.map(partial_sum,
range)
  total_sum = sum(results)
  print(4 * total_sum)

# Ejemplo de uso:
#pi_riemann_parallel(100_000)

# Run the profiling
#cProfile.run('pi_riemann_parallel(10
0000)')
cProfile.run('pi_riemann_parallel(100
000)', sort='cumtime')
```

Below is the profiling of the function ***pi_riemann_parallerl()***, which contains the approximation to π by numerical integration parallelized as calculation method:



```
3.1416126164019875
         1223 function calls in 0.253 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.253    0.253 {built-in method builtins.exec}
        1    0.000    0.000    0.253    0.253 <string>:1(<module>)
        1    0.000    0.000    0.252    0.252 <ipython-input-28-cd08c9e7bf4c>:12(pi_riemann_parallel)
       12    0.000    0.000    0.187    0.016 threading.py:589(wait)
       10    0.000    0.000    0.187    0.019 threading.py:288(wait)
       70    0.187    0.003    0.187    0.003 {method 'acquire' of '_thread.lock' objects}
        1    0.000    0.000    0.171    0.171 pool.py:362(map)
        1    0.000    0.000    0.171    0.171 pool.py:767(get)
        1    0.000    0.000    0.171    0.171 pool.py:764(wait)
        1    0.000    0.000    0.055    0.055 context.py:115(Pool)
        1    0.000    0.000    0.054    0.054 pool.py:183(__init__)
        1    0.000    0.000    0.046    0.046 pool.py:305(_repopulate_pool)
        1    0.001    0.001    0.046    0.046 pool.py:314(_repopulate_pool_static)
        4    0.000    0.000    0.044    0.011 process.py:110(start)
        4    0.000    0.000    0.044    0.011 context.py:278(_Popen)
        4    0.000    0.000    0.043    0.011 popen_fork.py:15(__init__)
        5    0.000    0.000    0.027    0.005 util.py:205(__call__)
        1    0.000    0.000    0.027    0.027 pool.py:738(__exit__)
        1    0.000    0.000    0.027    0.027 pool.py:654(terminate)
        1    0.000    0.000    0.027    0.027 pool.py:680(_terminate_pool)
        4    0.001    0.000    0.025    0.006 popen_fork.py:62(_launch)
        4    0.023    0.006    0.023    0.006 {built-in method posix.fork}
        1    0.000    0.000    0.019    0.019 pool.py:671(_help_stuff_finish)
        1    0.019    0.019    0.019    0.019 {method 'acquire' of '_multiprocessing.SemLock' objects}
        4    0.000    0.000    0.017    0.004 util.py:433(_flush_std_streams)
```

There are more results of using cProfile in the function, but how there are too many of them, we choose some of them as a sample.

There is a 3rd version of the approximation to pi by numerical integration parallelized with mpi4py library in python. But this version of the code couldn't be tested, because it didn't show any results.

The code use mpi4py library from python.

```
###Usando mpi4py
# Escribir el código en un archivo
Python
codigo_mpi = """
from math import sqrt
from mpi4py import MPI
import cProfile

def fun(x):
    return sqrt(1 - x**2)

def pi_riemann_mpi(n):
    comm = MPI.COMM_WORLD #Comunicador
    rank = comm.Get_rank()  #ID
    size = comm.Get_size()  #No.Procesos

    dx = 1.0 / n
    local_n = n // size
    local_start = rank * local_n
    local_end = (rank + 1) * local_n if
rank != size - 1 else n

    local_sum = sum(fun(i * dx) * dx for
i in range(local_start, local_end))
```

```
    total_sum =
comm.reduce(local_sum, op=MPI.SUM,
root=0)

    if rank == 0:
        pi = 4 * total_sum
        print(f"Approximation of PI
is {pi}")

if __name__ == "__main__":

    n=1000
    pi_riemann_mpi(n)
    cProfile.run('pi_riemann(1000)')

"""

# Escribir el código en un archivo en
el entorno de Colab
with open("mpi_example2.py", "w") as
file:
    file.write(codigo_mpi)

# Ejecutar el programa MPI
# Ejecutar el programa MPI
permitiendo la sobresuscripción de
procesos
!mpiexec --allow-run-as-root --
oversubscribe -n 2 python
mpi_example.py
```

Since the test does not display results on the screen, it cannot display the cProfiling result either.

VI. Conclusion

This report presents an estimate of the value of π using the numerical integration method in Python. Three different versions of the method were implemented, each wrapping a particular function of three, but only 2 were tested, while the last one didn't show anything in the screen.

Version 1: Basic implementation using a **for** loop and the **math.sqrt()** function to calculate the root of the numbers. We call this ***pi_riemann().***

Version 2: Parallel implementation using the multiprocessing library to distribute computation across multiple cores on each run. We call this ***pi_riemann_parallel().***

Version 3: Parallel implementation using the mpi4py library to distribute the computation across multiple processes. We call this ***pi_riemann_mpi()***.

Counterintuitively, **version 2**, ***pi_riemann_parallel()***, which uses multiprocessing, is the slowest. This may be due to a "small" number of iterations ,meanwhile with 10 million it should be the better version.

Version 3, ***pi_riemman_mpi()***, which uses multiprocessing with mpi4py, this should be the fastest of the 3. But it couldn't be tested, so there is not prove.

Version 1, ***pi_riemann().***, although the simplest, has a faster execution time to version 2, but while the number of iterations keep "low", but with 10 million this version should be slow.

References

[1] EcuRed. (s. f.). Integración numérica - EcuRed. https://www.ecured.cu/Integraci%C3%B3n_num%C3%A9rica#:~:text=Integraci%C3%B3n%20num%C3%A9rica.%20En%20las%20matem%C3%A1ticas%20el%20estudio%20de,un%20proceso%20de%20l%C3%ADmite%20de%20una%20suma%20finita.

[2] Repaso de sumas de Riemann (artículo) | Khan Academy. (s. f.). Khan Academy. https://es.khanacademy.org/math/ap-calculus-ab/ab-integration-new/ab-6-2/a/riemann-sums-review

[3] colaboradores de Wikipedia. (2023b, diciembre 19). Integración numérica. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/Integraci%C3%B3n_num%C3%A9rica

[4] https://www.perlego.com/book/527071/python-high-performance-second-edition-pdf