

Uso del método de Monte Carlo para aproximar el valor de Pi

Equipo 4

P. O. Karen Alexa, I. A. Lizeth Alexandra,
P. P. Maria de los Angeles, T. T. Viviana Nathalie,
M. V Miguel Angel

9 de septiembre de 2022

Resumen

El valor de Pi se sabe que es un valor irracional. Este valor de infinitas cifras decimales ha despertado el interés particular de muchos científicos y personas en todo el mundo. Parecería ser que encontrar nuevas cifras decimales del valor de Pi, se ha vuelto a lo largo del tiempo un desafío más que interesante para muchos matemáticos.

En el presente trabajo se utilizará el método de Monte Carlo para ejecutar simulaciones en Python considerando diferentes rangos y cantidad de replicas, utilizando números pseudoaleatorios generados por computadora. Una vez obtenidos los resultados se graficarán para poder observar el comportamiento de distintas pruebas.

1. Introducción

Nos encargaremos de aplicar el método de Monte Carlo haciendo un programa en Python que nos muestre en una gráfica los distintos valores de pi que serán dados de manera random gracias a una función de Python que nos permite tener números de manera aleatoria, además mostraremos en una gráfica los distintos valores que nos proyecte el programa. Para el desarrollo de este proyecto aplicamos el método de Montecarlo para acercarnos al número pi, para la realización del programa se usaron dos editores de código fuente Visual Studio Code y Spyder todo esto hecho en el lenguaje de Python.

2. Desarrollo

2.1. Metodología

Bajo el nombre de Método Monte Carlo o Simulación Monte Carlo se agrupan una serie de procedimientos que analizan distribuciones de variables aleatorias usando simulación de números aleatorios. En el caso de la estimación de Pi, se usan valores aleatorios para aproximar un valor determinístico [1].

El uso del método de Monte Carlo para aproximar el valor de Pi consiste en dibujar un cuadrado, y dentro de ese cuadrado, dibujar un círculo con diámetro de igual medida que uno de los lados del cuadrado. Luego se dibujan puntos de manera aleatoria sobre la superficie dibujada. Los puntos que están fuera del círculo y los que están dentro, sirven como estimadores de las áreas internas y externas del círculo.

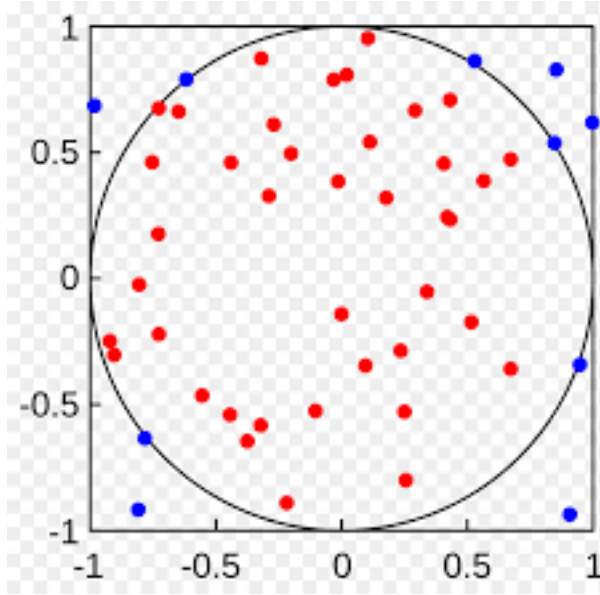


Figura 1: Método Montecarlo.

El área total del cuadrado con lado L es...

$$A_T \equiv L^2. \quad (1)$$

El área total del círculo dentro del cuadrado es...

$$A_C \equiv \pi * \left(\frac{L}{2}\right)^2. \quad (2)$$

La relación de áreas entonces es...

$$\frac{A_C}{A_T} \equiv \frac{\pi}{4}. \quad (3)$$

A partir de una estimación de esta relación, se multiplica por 4, y se obtiene el estimador de Pi. Para la simulación, se usan números pseudoaleatorios con distribución uniforme. A partir de esos números se forman las coordenadas de los puntos que se van a dibujar dentro del cuadrado.

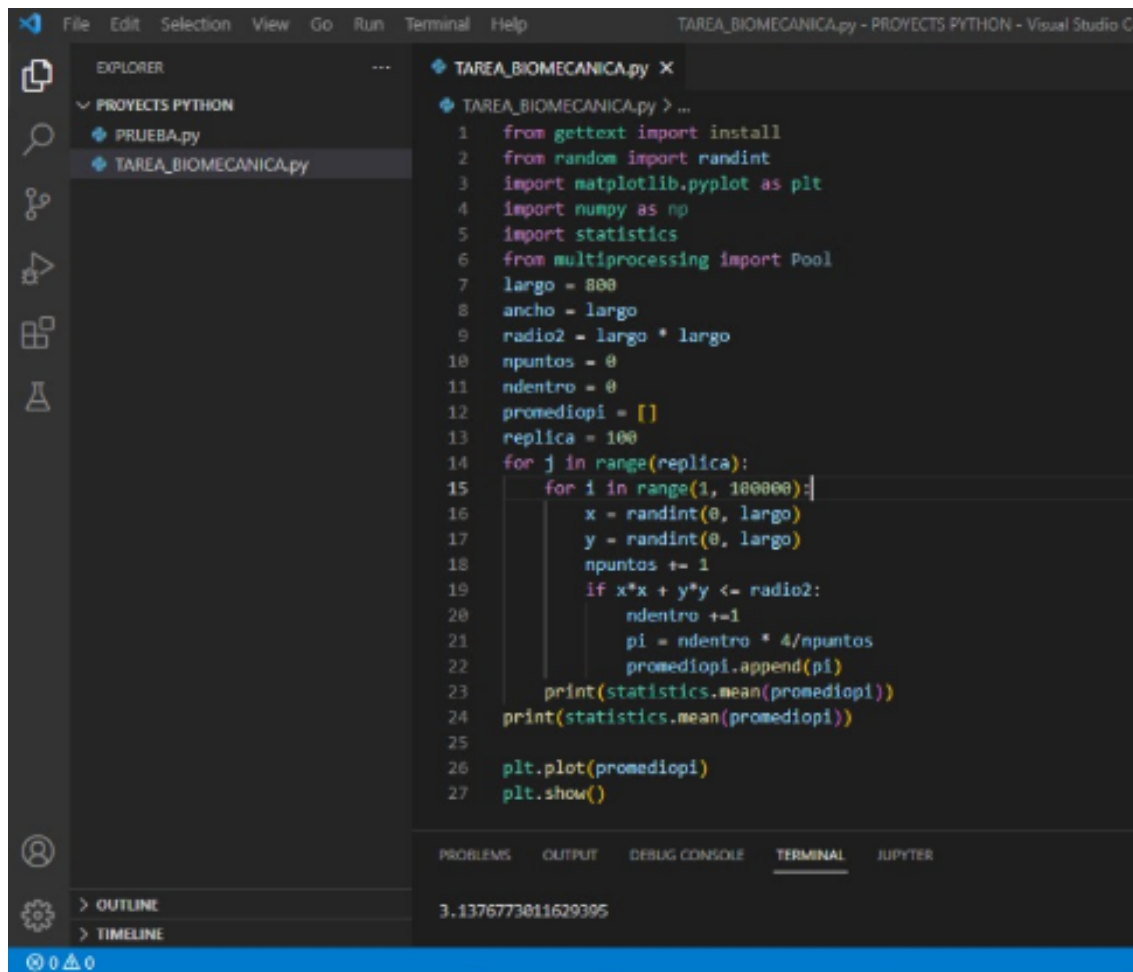
Una vez dibujados una cantidad suficiente de puntos, se estimará Pi mediante la fórmula...

$$\pi \approx 4 * \left(\frac{Puntos_i}{Puntos_t}\right). \quad (4)$$

2.2. Experimentación

Realizamos diferentes pruebas con distintos rangos y largos del círculo, todo esto se modifica en las variables de nuestro programa.

A continuación se mostrará el programa implementado para simular el método de Montecarlo hecho en Python desde Visual Studio.



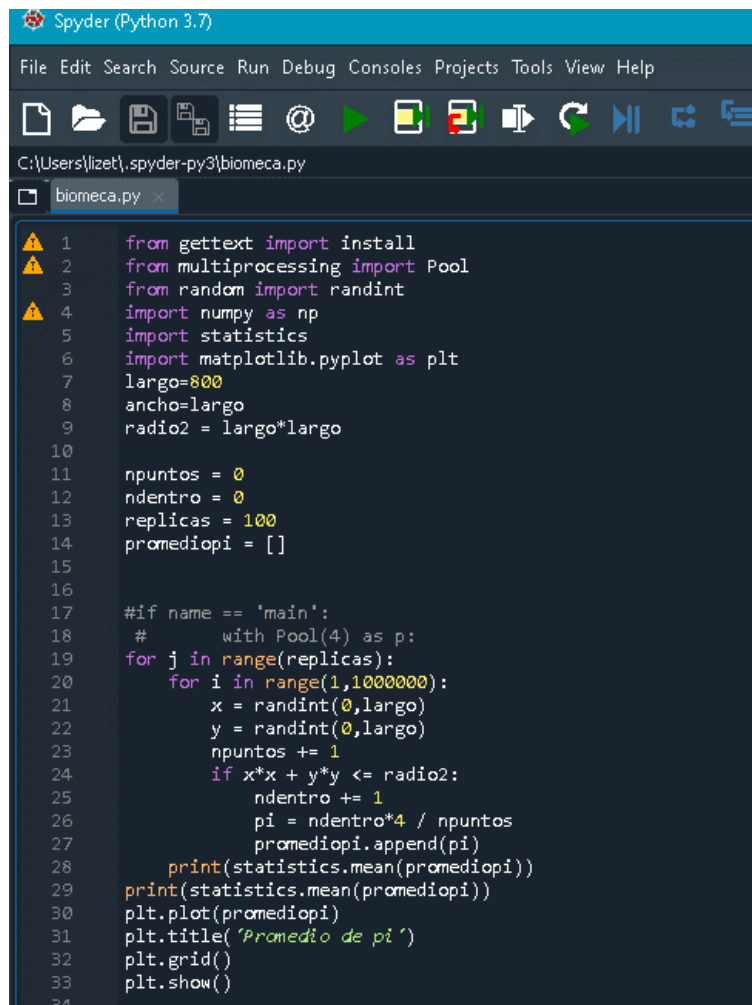
The image shows a screenshot of the Visual Studio Code editor. The Explorer panel on the left shows a project named 'PROYECTS PYTHON' with two files: 'PRUEBA.py' and 'TAREA_BIOMECHANICA.py'. The main editor window displays the code for 'TAREA_BIOMECHANICA.py'. The code is a Python script that implements the Monte Carlo method for estimating the value of pi. It uses random sampling within a square to estimate the area of an inscribed circle. The script includes imports for 'gettext', 'random', 'matplotlib.pyplot', 'numpy', 'statistics', and 'multiprocessing.Pool'. It defines variables for 'largo', 'ancho', 'radio2', 'npuntos', 'ndentro', 'promediopi', and 'replica'. A nested loop structure is used to perform the simulation, and the results are plotted using 'plt.plot' and 'plt.show()'. The terminal at the bottom shows the output of the script, which is the value 3.1376773011629395.

```
1 from gettext import install
2 from random import randint
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import statistics
6 from multiprocessing import Pool
7 largo = 800
8 ancho = largo
9 radio2 = largo * largo
10 npuntos = 0
11 ndentro = 0
12 promediopi = []
13 replica = 100
14 for j in range(replica):
15     for i in range(1, 100000):
16         x = randint(0, largo)
17         y = randint(0, largo)
18         npuntos += 1
19         if x*x + y*y <= radio2:
20             ndentro += 1
21             pi = ndentro * 4/npuntos
22             promediopi.append(pi)
23     print(statistics.mean(promediopi))
24 print(statistics.mean(promediopi))
25
26 plt.plot(promediopi)
27 plt.show()
```

3.1376773011629395

Figura 2: Programa del metodo Montecarlo en Visual Studio.

De igual forma se trabajó en el editor de texto Spyder como se muestra en la figura 3.

The image shows the Spyder Python IDE interface. The title bar reads 'Spyder (Python 3.7)'. The menu bar includes 'File', 'Edit', 'Search', 'Source', 'Run', 'Debug', 'Consoles', 'Projects', 'Tools', 'View', and 'Help'. Below the menu bar is a toolbar with icons for file operations and execution. The file explorer on the left shows the file 'biomeca.py' in the directory 'C:\Users\lizet\spyder-py3\'. The main editor window displays the following Python code:

```
1  from gettext import install
2  from multiprocessing import Pool
3  from random import randint
4  import numpy as np
5  import statistics
6  import matplotlib.pyplot as plt
7  largo=800
8  ancho=largo
9  radio2 = largo*largo
10
11  npuntos = 0
12  ndentro = 0
13  replicas = 100
14  promediopi = []
15
16
17  #if name == 'main':
18  #    with Pool(4) as p:
19  for j in range(replicas):
20     for i in range(1,1000000):
21         x = randint(0,largo)
22         y = randint(0,largo)
23         npuntos += 1
24         if x*x + y*y <= radio2:
25             ndentro += 1
26             pi = ndentro*4 / npuntos
27             promediopi.append(pi)
28     print(statistics.mean(promediopi))
29 print(statistics.mean(promediopi))
30 plt.plot(promediopi)
31 plt.title('Promedio de pi')
32 plt.grid()
33 plt.show()
34
```

Figura 3: Programa del metodo Montecarlo en Spyder.

Para realizar el programa se trabajó con la biblioteca random, la cual contiene una serie de funciones relacionadas con los valores aleatorios[2].

La función randint(a, b) genera un número entero entre a y b, ambos incluidos. a debe ser inferior o igual a b. La función random() genera un número decimal entre 0 y 1 (puede generar 0, pero no 1). La función randrange(a, b, c) genera un número entero entre los valores generados por range(a, b, c). Como ocurre con range(), la función randrange() admite uno, dos o tres argumentos.

2.3. Resultados

2.3.1. Rango de 1 a 300, largo de 300, 100 replicas.

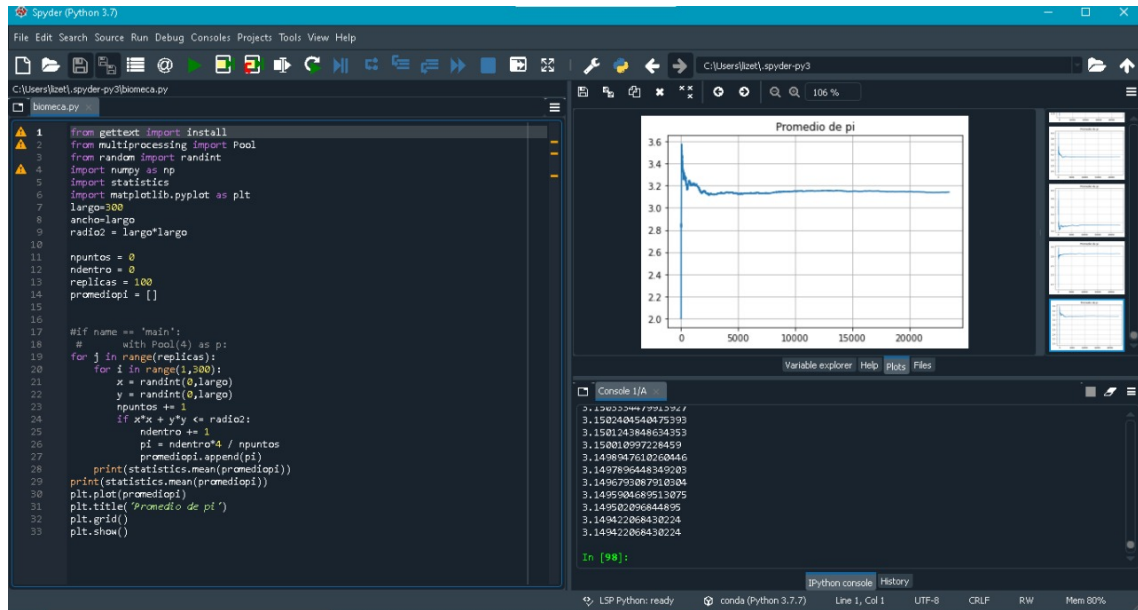


Figura 4: Programa del metodo Montecarlo, primera prueba.

2.3.2. Rango de 1 a 100000, largo de 800, 100 replicas.

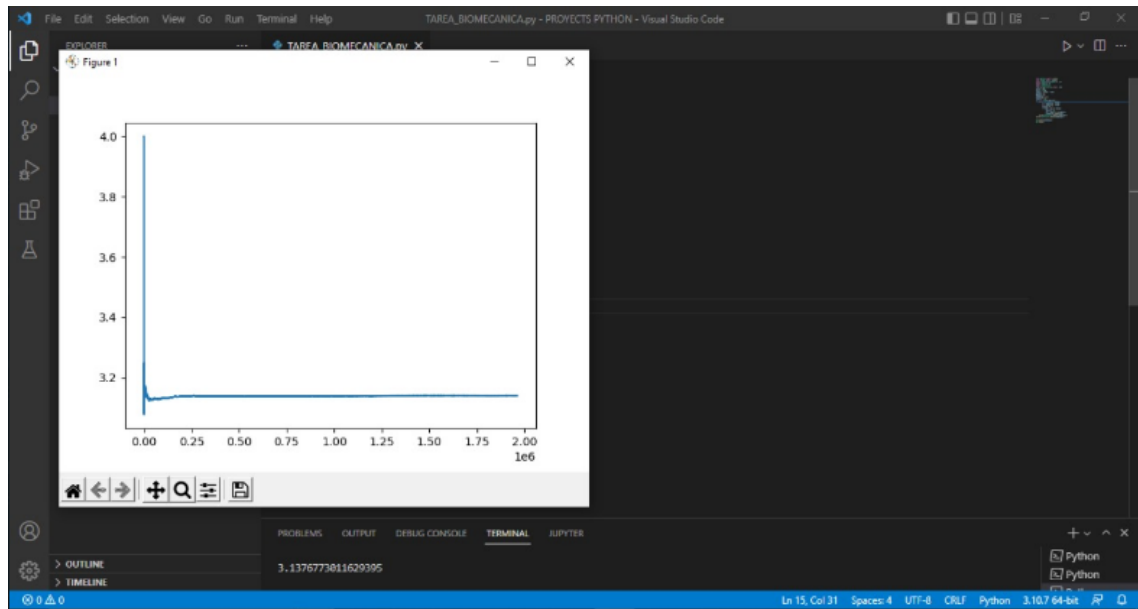


Figura 5: Programa del metodo Montecarlo, segunda prueba.

2.3.3. Rango de 1 a 300, largo de 300, 200 replicas.

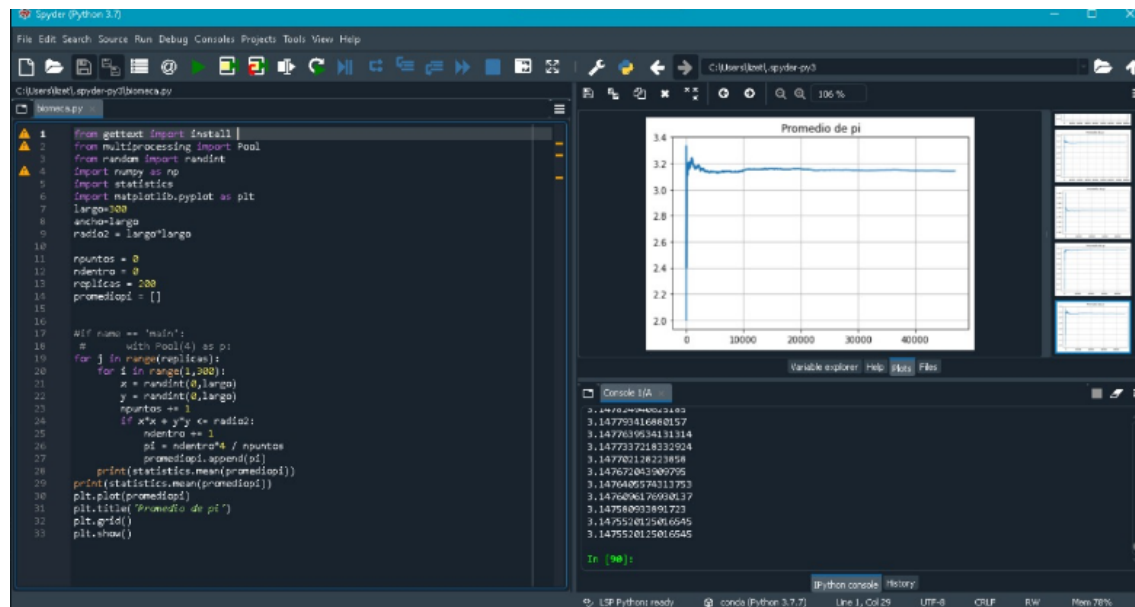


Figura 6: Programa del metodo Montecarlo, tercera prueba.

Se puede observar en las gráficas cómo se estabiliza en el valor más cercano a pi. Se encontraron cambios en las gráficas al modificar ciertas variables como el rango y las replicas.

Pudimos notar que aumentar el número de replicas nos ayudaba a que el valor fuera cada vez más cercano al valor de pi y disminuir el rango nos permitía tener una mejor visualización del comportamiento de la gráfica.

3. Conclusiones

La elaboración de esta actividad fue de gran ayuda para así poder conocer mas acerca del entorno y el uso de Python en Visual Studio, esto utilizando un programa para obtener un valor muy cercano a pi, el cual resultado en una gráfica donde se iban colocando cada uno de estos valores hasta obtener uno muy parecido al que conocemos, de igual forma, al estar haciendo esto conforme se aumentaban el numero de replicas de iba acercando mas a pi y si se disminuía el rango esto nos permitiría que haya una mejor visión en cuanto al comportamiento de la gráfica.

Referencias

- [1] Juan Churruarín Gastón A. Addati, Fernando Celano. Modelos y simulación. www.cema.edu.ar/publicaciones/doc_trabajo.html, Agosto 2016. Accedido en septiembre de 2022.
- [2] Bartolomé Sintés Marco. Valores aleatorios: la biblioteca random, Abril 2019. Accedido en septiembre de 2022.