

# Obtención de $\pi$ por método Montecarlo en Python

Eimie Carolina Pereda Sánchez  
Gloria Rosalía Domínguez Azueta  
Luis Lauro García Hernández  
Jorge Luis Ávila Hernández

12 de septiembre de 2022

## Resumen

En este documento se presenta una explicación breve de  $\pi$  y la manera de obtenerlo mediante el método Montecarlo usando el programa python, partiendo de los números aleatorios. El método Montecarlo tiene muchas funciones y se puede usar para muchos ejemplos, también esto tiene sus ciertas ventajas y desventajas, pero al final los resultados fueron muy próximos al valor de  $\pi$  y la metodología resulto bastante intuitiva.

**Palabras clave:**  $\pi$ , método Montecarlo, python, codificación, números aleatorios.

## 1. Introducción

Pi ( $\pi$ ) es la relación entre las longitudes de una circunferencia y su diámetro, en geometría euclidiana. Es un número irracional y una de las constantes matemáticas más importantes. Se emplea frecuentemente en matemáticas, física e ingeniería. El valorérico de  $\pi$ , truncado a sus primeras cifras, es el siguiente:

$$\pi \approx 3,1415926535897932384...$$

Figura 1: Valor de  $\pi$

El valor de  $\pi$  se ha obtenido con diversas aproximaciones a lo largo de la historia, siendo una de las constantes matemáticas que más aparece en las ecuaciones de la física, junto con el número  $e$ . Por ello, tal vez sea la constante que más pasiones desata entre los matemáticos profesionales y aficionados. La relación entre la circunferencia y su diámetro no es constante en geometrías no euclídeas. En este reporte, para la obtención de  $\pi$  se utiliza el lenguaje de programación "Python", en el cual se programa utilizando el método Montecarlo a partir de números aleatorios. Como primera sección se presenta el estado del arte como parte del desarrollo, posteriormente en la sección de experimentación se expone la programación del código y los resultados por medio de gráficas. Finalmente, se concluye respecto al reporte y se incluyen las referencias utilizadas y el anexo del código de programación[1].

## 2. Desarrollo

El término Monte Carlo se aplica a un conjunto de métodos matemáticos que se empezaron a usar en los 1940s para el desarrollo de armas nucleares en Los Alamos, favorecidos por la aparición de los ordenadores digitales modernos. Consisten en resolver un problema mediante la invención de juegos de azar cuyo comportamiento simula algún fenómeno real gobernado por una distribución de probabilidad o sirve para realizar un cálculo. Más técnicamente, un Monte Carlo es un proceso estocástico numérico, es decir, una secuencia de estados cuya evolución viene determinada por sucesos aleatorios. Recordemos que un suceso aleatorio es un conjunto de resultados que se producen con cierta probabilidad. Veamos un par de ejemplos ilustrativos[2].

### Ejemplo 1: Gotas de lluvia para estimar $\pi$

Consideremos un círculo de radio unidad circunscrito por un cuadrado. Suponiendo una lluvia uniforme sobre el cuadrado, podemos hallar el valor de  $\pi$  a partir de la probabilidad de que las gotas caigan dentro del círculo (figura 2):

$$P = \frac{\text{área del círculo}}{\text{área del cuadrado}} = \frac{\int_{-1}^1 dx \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} dy}{\int_{-1}^1 dx \int_{-1}^1 dy} = \frac{2 \int_{-1}^1 dx \sqrt{1-x^2}}{2 \cdot 2} = \frac{\pi}{4}.$$

Figura 2: Relación de áreas del círculo y cuadrado

Es decir,  $\pi = 4P$ . Nótese que: - Podemos simular fácilmente este experimento generando aleatoriamente con un ordenador puntos de coordenadas cartesianas (x, y);

- Podemos mejorar nuestra estimación de  $\pi$  aumentando el número de puntos generados (ejercicio 1);

- Tenemos un método para hallar la integral que aparece en la ecuación (1.1). Ciertamente, el valor de  $\pi$  puede encontrarse de forma más rápida y precisa mediante otros métodos, pero veremos que el método Monte Carlo es el más eficiente para hallar integrales multidimensionales.

### 2.1. Ventajas y Desventajas

#### Ventajas:

- Es un método directo y flexible. Existe un amplio abanico de programas y lenguajes destinados a simular.
- Cuando el modelo matemático es demasiado complicado, la simulación permite obtener una aproximación.
- La simulación nos permite formular condiciones extremas con riesgos nulos.
- La simulación no interfiere con el mundo real. Permite experimentar.

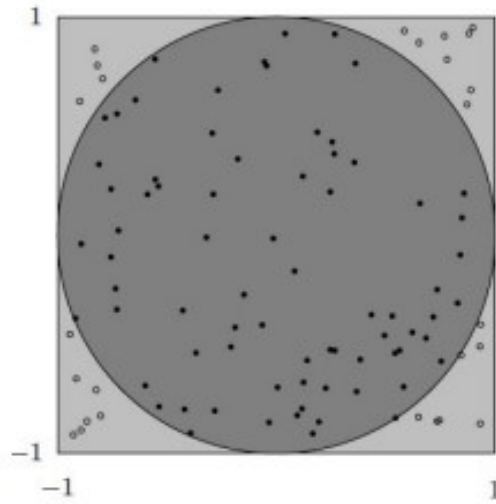


Figura 3: Experimento de las gotas de lluvia para estimar



Figura 4: Experimento de las agujas de Buffon para estimar  $\pi$

- Permite estudiar la interacción entre las diferentes variables del problema. Mediante la simulación podemos “influir en el tiempo” de los procesos.
- La simulación permite resolver problemas que no tienen solución analítica.

#### **Desventajas:**

- Una buena simulación puede resultar muy complicada, gran número de variables.
- La simulación no genera soluciones óptimas globales.
- No proporciona la decisión a tomar, sino que resuelve el problema mediante aproximación para unas condiciones iniciales.
- Cada simulación es única, interviene el azar.

## 2.2. Generación de números pseudo-aleatorios

La generación de una buena secuencia de números aleatorios es la base probabilística del método de Monte Carlo. Cada número aleatorio debe ser totalmente independiente de los otros números de la secuencia. Además, dos generadores aleatorios independientes deben proporcionar estadísticamente el mismo valor promedio de salida.

Esto hace que los generadores tengan que cumplir una serie de características:

1. Buena distribución; se entiende que los números obtenidos estén uniformemente distribuidos en el intervalo en el que se obtienen  $[0, 1]$ . Si tomamos un subintervalo cualquiera, la fracción de números aleatorios que aparece respecto del total tiene que ser la misma para todo subintervalo de la misma amplitud.
2. Al ser generados mediante un algoritmo, siempre tienen un ciclo más o menos largo. En el caso de simulaciones en que se usa una gran cantidad de números aleatorios, es importante que estos no se repitan para que evitar las correlaciones.
3. Es importante que se pueda reproducir la sucesión de números usados. Si se repite la simulación en las mismas condiciones, el resultado ha de ser el mismo.

## 3. Descripción del algoritmo

El algoritmo de Simulación Monte Carlo fundamentado en la generación de números aleatorios, se basa en las distribuciones acumuladas de frecuencias:

- Determinar la o las variables aleatorias y sus funciones de distribución de probabilidad. - Generar un número aleatorio uniformemente distribuido en  $(0,1)$ .
- Determinar el valor de la variable aleatoria para el número aleatorio generado de acuerdo con las pdf que tengamos.
- Iterar los dos pasos anteriores tantas veces como muestras necesitamos.
- Calcular la media, desviación estándar, error y realizar el histograma.
- Analizar resultados para distintos tamaños de muestra.

Otra opción para trabajar con el método Monte Carlo, cuando la variable aleatoria no es directamente el resultado de la simulación o tenemos relaciones entre variables, es la siguiente:

- Diseñar el modelo lógico de decisión.
- Especificar las distribuciones de probabilidad para las variables aleatorias relevantes.
- Incluir posibles dependencias entre variables.
- Muestrear valores de las variables aleatorias.

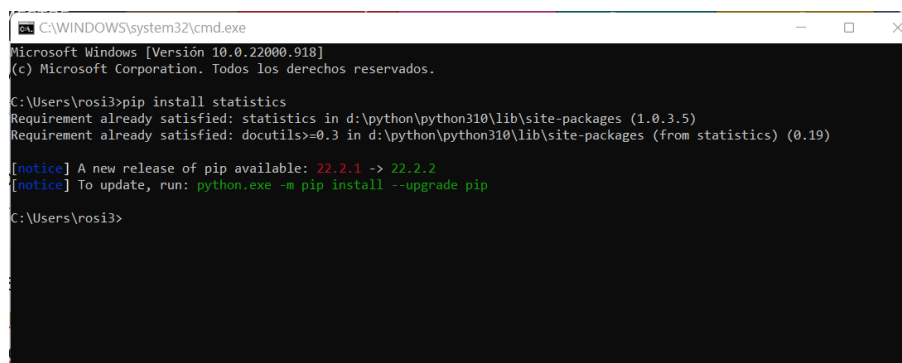
- Calcular el resultado del modelo según los valores del muestreo (iteración) y registrar el resultado.
- Repetir el proceso hasta tener una muestra estadísticamente representativa.
- Obtener la distribución de frecuencias del resultado de las iteraciones.
- Calcular media, desviación.

Las principales características para tener en cuenta para la implementación o utilización del algoritmo son:

- El sistema debe ser descrito por una o más funciones de distribución de probabilidad.
- El proceso de generación de los números aleatorios es importante para evitar que se produzca correlación entre los valores muestrales.
- Establecer límites y reglas de muestreo para las pdf; conocemos qué valores pueden adoptar las variables.
- Definir cuándo un valor aleatorio tiene o no sentido para el modelo a simular.
- Estimar con qué error trabajamos, cuánto error podemos aceptar para que una corrida sea válida.
- Paralelización y vectorización: En aplicaciones con muchas variables se estudia trabajar con varios procesadores paralelos para realizar la simulación[3].

## 4. Experimentación

En el proceso de elaboración de la programación para la obtención de  $\pi$  se descargaron librerías necesarias para que el programa ejecutará la programación sin errores, algunas de las librerías descargadas fueron: statistics y matplotlib (Figura 5 y 7) con ayuda del comando en el CMD: "pip installz seguido el nombre de la librería.



```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.22000.918]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\rosi3>pip install statistics
Requirement already satisfied: statistics in d:\python\python310\lib\site-packages (1.0.3.5)
Requirement already satisfied: docutils>=0.3 in d:\python\python310\lib\site-packages (from statistics) (0.19)
[notice] A new release of pip available: 22.2.1 -> 22.2.2
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\rosi3>

```

Figura 5: Instalación de librería statistics ("pip install statistics")

Una vez instaladas las librerías se prosiguió a escribir el código en Visual Studio Code, en un archivo previamente creado de formato .py.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.22000.918]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\rosi3>pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.5.3-cp310-cp310-win_amd64.whl (7.2 MB)
----- 7.2/7.2 MB 2.0 MB/s eta 0:00:00
Collecting packaging>=20.0
  Downloading packaging-21.3-py3-none-any.whl (40 kB)
----- 40.8/40.8 kB ? eta 0:00:00
Collecting python-dateutil>=2.7
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
----- 247.7/247.7 kB 2.5 MB/s eta 0:00:00
Collecting pillow>=6.2.0
  Downloading Pillow-9.2.0-cp310-cp310-win_amd64.whl (3.3 MB)
----- 3.3/3.3 MB 2.1 MB/s eta 0:00:00
Collecting cycler>=0.10
  Downloading cycler-0.11.0-py3-none-any.whl (6.4 kB)
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.4.4-cp310-cp310-win_amd64.whl (55 kB)
----- 55.3/55.3 kB ? eta 0:00:00
Collecting pyparsing>=2.2.1
  Downloading pyparsing-3.0.9-py3-none-any.whl (98 kB)
----- 98.3/98.3 kB ? eta 0:00:00
Requirement already satisfied: numpy>=1.17 in d:\python\python310\lib\site-packages (from matplotlib) (1.23.3)
Collecting fonttools>=4.22.0
  Downloading fonttools-4.37.1-py3-none-any.whl (957 kB)
----- 957.2/957.2 kB 2.2 MB/s eta 0:00:00
Collecting six>=1.5
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
```

Figura 6: Instalación de librería Matplotlib ("pip install matplotlib")

```
D:\> Users > rosi3 > Downloads > pi.py > ...
1  #EQUIPO 8
2  #Eimie Carolina Pereda Sánchez
3  #Gloria Rosalía Domínguez Azueta
4  #Luis Lauro García Hernández
5  #Jorge Luis Ávila Hernández
6  from multiprocessing import Pool
7  import matplotlib.pyplot as plt
8  from random import randint
9  import statistics
10 width = 10000
11 height = width
12 radio = width
13
14 npuntos = 0
15 ndentro = 0
16 radio2 = radio*radio
17 replicas = 1000
18 promediopi = []
19 listareplicas = []
20 listapromedios = []
21
22 if __name__ == '__main__':
23     with Pool(4) as p:
24         for j in range(replicas):
25             for i in range(1,100000):
26                 x = randint(0,width)
27                 y = randint(0,width)
28                 npuntos += 1
29                 if x*x + y*y <= radio2:
30                     ndentro += 1
31                     pi = ndentro * 4 / npuntos
32                     promediopi.append(pi)
33                     print(statistics.mean(promediopi))
34                     listareplicas.append(j)
35                     listapromedios.append(statistics.mean(promediopi))
36                     #print(statistics.mean(promediopi))
37             plt.plot(listareplicas,listapromedios)
38             plt.xlabel('Réplicas')
39             plt.ylabel('Valores promedio de pi')
40             plt.title('EQ#8:Aproximación de pi por método Montecarlo')
41             plt.show()
```

Figura 7: Código programado en Python para obtención aproximada de  $\pi$  por método Montecarlo.

## 5. Resultados

Los resultados obtenidos de la ejecución del código se muestran en las Figuras 8 y 9.

En las gráficas anteriores podemos observar que mientras más replicas de áreas sean obtenidas, mayor es la aproximación de  $\pi$ .

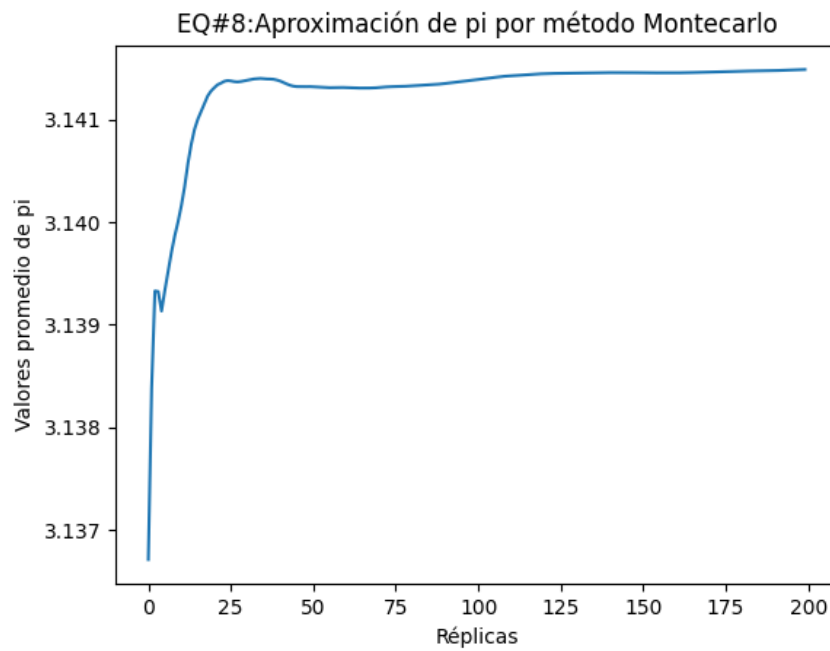


Figura 8: Gráfica de la aproximación de obtención de  $\pi$  a 200 réplicas.

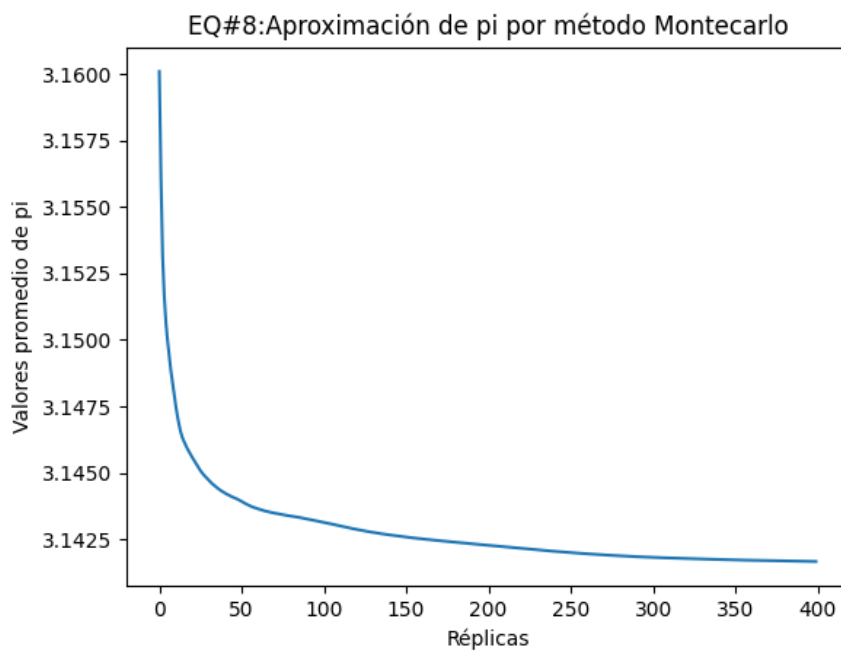


Figura 9: Gráfica de la aproximación de obtención de  $\pi$  a 400 réplicas.

## 6. Conclusiones

Este trabajo utiliza el método Montecarlo para la obtención de pi mediante el uso de software Python, la programación del código nos ayudó a conocer el entorno de programación Python, el cual nos servirá para futuros trabajos y es bueno empezar por algo un poco básico para así ir ganando experiencia e irnos familiarizando con el programa Python. Hablando del método Montecarlo se puede aplicar en muchas partes y es bastante intuitivo y flexible, esto hace fácil el poder obtener pi aunque este no sea muy preciso, pero si aproximado, gracias a los números que se arrojan aleatoriamente, aun así consideramos que es un método muy bueno para la obtención de pi.

## Referencias

- [1] Federación de enseñanza de CC.OO. Números famosos: El número pi, Mayo 2009.
- [2] José Ignacio Illana. Departamento de física teórica y del cosmos universidad de granada, Enero 2013.
- [3] Carlos Zapata. El método de monte carlo y el programa de computo mcnp, Julio 2015.

## Anexo

Código de programación utilizado en Python para la obtención aproximada de  $\pi$ :

```
1  #EQUIPO 8
2  from multiprocessing import Pool
3  import matplotlib.pyplot as plt
4  from random import randint
5  import statistics
6  width = 10000
7  height = width
8  radio = width
9
10 npuntos = 0
11 ndentro = 0
12 radio2 = radio*radio
13 replicas = 400
14 promediopi = []
15 listareplicas = []
16 listapromedios = []
17
18 if __name__ == '__main__':
19     with Pool(4) as p:
20         for j in range(replicas):
21             for i in range(1,100000):
22                 x = randint(0,width)
```



```

23         y = randint(0,width)
24         npuntos += 1
25         if x*x + y*y <= radio2:
26             ndentro += 1
27         pi = ndentro * 4 /npuntos
28         promediopi.append(pi)
29         print(statistics.mean(promediopi))
30         listareplicas.append(j)
31         listapromedios.append(statistics.mean(promediopi))
32     #print(statistics.mean(promediopi))
33     plt.plot(listareplicas,listapromedios)
34     plt.xlabel('R plicas')
35     plt.ylabel('Valores promedio de pi')
36     plt.title('EQ#8:Aproximaci n de pi por m todo Montecarlo')
37     plt.show()

```