Cálculo de pi mediante método Montecarlo

Equipo 10
Jimena Hernández
Melissa Jasso
Rubén Cantú
Omar Gutierrez
Luis Landa

8 de septiembre de 2022

Resumen

En este presente documento se desarrolla el cálculo de los diferentes valores de Pi, existen diferentes maneras de calcularlo solo que en esta ocasión tendrá un enfoque más apto al cálculo de Pi mediante el método Montecarlo, el cuál será descrito primeramente para después utilizarlo mediante Phyton.

1. Introducción

El método Monte Carlo es un método en el que por medio de la estadística y la probabilidad podemos determinar valores o soluciones de ecuaciones que calculados con exactitud son muy complejas, pero que mediante este método resulta sencillo calcular una aproximación al resultado que buscamos.

El método Monte Carlo fue desarrollado en 1944 en Laboratorio Nacional de Los Álamos, como parte de los estudios que condujeron al desarrollo de la bomba atómica. En un principio lo desarrollaron los matemáticos John Von Neumann y Stanislaw Ulam aunque fueron otros matemáticos quienes con su trabajo le dieron una solidez científica, Harris y Herman Kahn[1].

El método de Montecarlo proporciona soluciones aproximadas a una gran variedad de problemas matemáticos posibilitando la realización de experimentos con muestreos de números pseudoaleatorios en una computadora. El método es aplicable a cualquier tipo de problema, ya sea estocástico o determinista. A diferencia de los métodos numéricos que se basan en evaluaciones en N puntos en un espacio M-dimensional para producir una solución aproximada[4].

Por otro lado la programación de el desarrollo del método Montecarlo y la gráfica de Pi se trabajará en Phyton. Python es un lenguaje de programación de alto nivel, orientado a objetos, con una semántica dinámica integrada, principalmente para el desarrollo web y de aplicaciones informáticas.

Es muy atractivo en el campo del Desarrollo Rápido de Aplicaciones (RAD) porque ofrece tipificación dinámica y opciones de encuadernación dinámicas.

Python es relativamente simple, por lo que es fácil de aprender, ya que requiere una sintaxis única que se centra en la legibilidad. Los desarrolladores pueden leer y traducir el código Python mucho más fácilmente que otros lenguajes.

Por tanto, esto reduce el costo de mantenimiento y de desarrollo del programa porque permite que los equipos trabajen en colaboración sin barreras significativas de lenguaje y experimentación[3].

2. Desarrollo

2.1. Metodo Montecarlo

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 aparicion de los ordenadores digitales modernos. Consisten en resolver un problema mediante la invención de juegos de azar cuyo comportamiento simula algún fenomeno real gobernado por una distribución de probabilidad (e.g. un proceso físico) o sirve para realizar un cálculo (e.g. evaluar una integral).

Mas técnicamente, 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 [2]. Veamos un par de ejemplos ilustrativos.

- Gotas de lluvia para estimar Π
  Consideremos un círculo de radio unidad circunscrito por un cuadrado. Suponiendo una lluvia uniforme sobre el cuadrado, podemos hallar el valor de Π a partir de la probabilidad de que las gotas caigan dentro del círculo de la Figura 1.

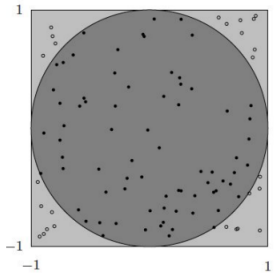

Figura 1: Experimento de las gotas de lluvia para estimar Π.

$$P = \Pi/4$$

  Podemos simular fácilmente este experimento generando aleatoriamente con un ordenador puntos de coordenadas cartesianas (x, y); podemos mejorar nuestra estimación de Π aumentando el numero de puntos generados. Ciertamente el valor de Π puede encontrarse de forma más rápida y precisa mediante otros metodos, pero veremos que el método Monte Carlo es el más eficiente para hallar integrales multidimensionales.

- Las agujas de Buffon para estimar Π
  En 1773 Georges Louis Leclerc, conde de Buffon, propuso el siguiente problema, que el´ mismo resolvió en 1777. Consideremos un papel horizontal, en el que se han trazado rectas paralelas separadas una distancia d, sobre el que se dejan caer aleatoriamente agujas de longitud Ł $< d$ Figura 2

Figura 2: : Experimento de las agujas de Buffon para estimar Π.

¿Cuál es la probabilidad que las agujas crucen una cualquiera de las rectas? La respuesta es:

$$P = 2L/\Pi d$$

Esto nos permite estimar el valor de Π, aunque de nuevo la convergencia es muy lenta. En los dos ejemplos hemos destacado la palabra aleatorio. Sin embargo, un ordenador es determinista, pues solamente es capaz de generar una secuencia programada de numeros ´ pseudoaleatorios.

2.2. Generación de código en Python

En esta sección procederemos a aplicar el método Monte Carlo de gotas de lluvia para obtener el número Pi, esto en base a la ecuación y explicación anteriormente mencionada, para esto se genera un código en Python.

La descripción del código que se mostrara a continuación permite observar la estimación del valor PI en tiempo real por número de repeticiones y por número de replicas, el cambio entre estas se aplica comentando las líneas correspondientes.

2.2.1. Librerias

- multiprocessing - Pool:
  Sirve para utilizar los núcleos físicos de nuestra computadora en la tarea programada y reducir el tiempo de ejecución.
- random - randint:
  Permite generar números enteros aleatorios
- matplotlib.pyplot - plt:
  Permite crear y personalizar los tipos de gráficos más comunes.
- matplotlib.pyplot - *:
  Permite crear e importar la figura
- numpy - np:
  Especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.

```
1  from multiprocessing import Pool
2  from random import randint
3  import matplotlib.pyplot as plt
4  from matplotlib.pyplot import *
5  import numpy as np
```

Figura 3: Librerias utilizadas Π.

2.2.2. Variables

En esta sección se establecen los parámetros necesarios para el dimensionamiento del cuadrado (width y height), así como el radio del círculo. Después se inician en cero el número de puntos dentro y fuera del círculo y se obtiene el radio al cuadrado. Establecemos el número de replicas que tendrá la aproximación de Π, se asigna una variable "nïgual a cero que representara el número de repeticiones y estas se guardaran en un array llamado "nrep", al igual que las aproximaciones de Π en "promediopi". Estas se pueden observar en la figura 4

```
8   width = 10000
9   heigth = width
10  radio = width
11
12  npuntos = 0
13  ndentro = 0
14  radio2 = radio*radio
15  replicas = 100
16  n = 0
17  nrep = []
18  promediopi = []
19  pi2 = []
```

Figura 4: Variables utilizadas.

2.2.3. Aplicación de método y generación de gráfica

Primero establecemos los límites de la gráfica que se mostrará, así como el título en cada eje visto en la figura 5.

```
20  plt.ylim(2,4)
21  plt.xlabel("No. Repeticiones \n [1e6 = 10 replicas]", size = 8)
22  plt.ylabel("π", size = 8)
```

Figura 5: Limites y titulo de ejes

En la figura 6 se realiza la implementación del método Monte Carlo para obtener el número Π, primeramente se genera la implementación de los cuatro núcleos físicos que se tienen en un i7 7th Gen, para después generar dos ciclos for, el primero repite replicas"veces el siguiente for que realiza la generación de puntos aleatorios entre el 0 y ancho. En promediopi.append(pi) almacena pi en un vector, n+=1 actualiza el conteo de número de repeticiones y se almacena en nrep.append(n).

```
24  if __name__ == '__main__':
25          with Pool(4) as p:
26              for j in range(replicas):
27                  for i in range(1,100000):
28                      x = randint(0,width)
29                      y = randint(0,width)
30                      npuntos += 1
31                      if x*x + y*y <= radio2:
32                          ndentro += 1
33                      pi = ndentro * 4 /npuntos
34                      promediopi.append(pi)
35                      n+=1
36                      nrep.append(n)
```

Figura 6: Estimación de Π por Monte Carlo

Por último se crean las líneas de código encargadas de generar la gráfica de la estimación de Π. Este código puede ser ejecutado desde el segundo ciclo for para hacer muestreo de Π por cada generación de número aleatorio y desde el for replicas"para hacer el muestreo en cada replica generada.

En la línea 48 generamos la figura que será la 1, en la línea 49 se genera una línea a lo largo del eje X, donde la altura en la que inicia es el valor de Π para tener una referencia conforme la estimación.

Después en la línea 50 en donde se genera la gráfica se configura como (eje X, eje Y, color de línea)

Línea 51 se agrega un título a la gráfica con un tamaño definido, en la línea 52 se genera el símbolo de Π para colocarlo encima de la línea generada en la línea 49.

Con la línea 53 y 54 se crea la animación de escribir los datos conforme se van obteniendo en la ejecución del programa con una pausa de 0.1s para evitar estresar la computadora y por último en la línea 55 se muestra la animación. Todo lo anterior se presenta en la figura que es mostrada a continuación 7.

```
37                  # Vista por repetición
38                  """
39                  plt.figure(1)
40                  plt.axhline(y=np.pi, c='darkseagreen',linewidth=2,alpha=0.5)
41                  plt.plot(nrep,promediopi,c='mediumorchid')
42                  plt.title('Aproximación π', size = 16)
43                  plt.annotate('π',[0,np.pi],fontsize=25)
44                  plt.draw()
45                  plt.pause(0.1)
46              plt.show() """
47              # Vista por replica
48              plt.figure(1)
49              plt.axhline(y=np.pi, c='darkseagreen',linewidth=2,alpha=0.5)
50              plt.plot(nrep,promediopi,c='mediumorchid')
51              plt.title('Aproximación π', size = 16)
52              plt.annotate('π',[0,np.pi],fontsize=25)
53              plt.draw()
54              plt.pause(0.1)
55          plt.show()
```

Figura 7: Código de generación de gráfica

Nota: Si se desea ver la animación se recomienda ejecutar el programa anexado en la carpeta de GitHub.

2.3. Resultados

Para comprobar el funcionamiento del programa se realizarán dos diferentes pruebas para observar el comportamiento de este al graficar por medio de replicas y repeticiones.

Graficado desde replicas:

- Estimación con 30 replicas
  Al realizar el cálculo de PI con 30 réplicas se obtuvo un resultado de 3.140 muy cercano al valor, por lo que se aumentaran las réplicas para acercarse más al valor.

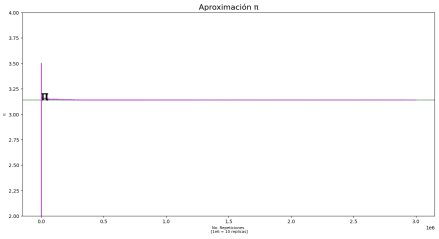

Figura 8: Gráfica de aproximación PI

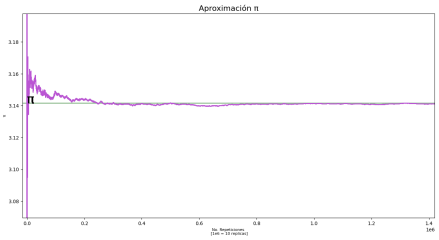

Figura 9: Zoom al comportamiento de PI

- Estimación con 50 replicas
  En las siguientes figuras se presenta otra generación distinta, esta vez desde un número no muy mayor a pi, pero si sobrepasa el límite inferior, conforme se van realizando más replicas se acentúa el valor y se obtiene un valor cada vez más cercano, ahora este es de 3.141570.

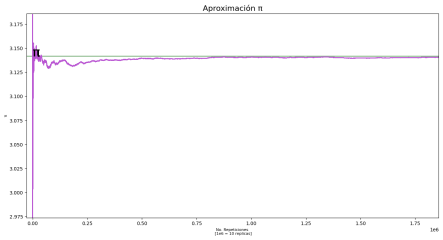

Figura 10: Gráfica de aproximación PI

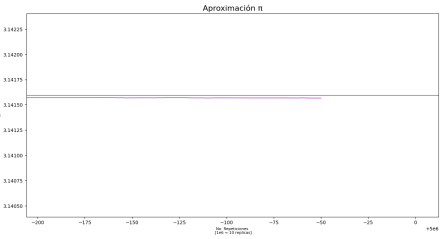

Figura 11: Acercamiento a la referencia de PI

- Estimación con 100 replicas

Graficado desde repeticiones:

- Estimación con 30 replicas
  Esta estimación del número Pi es realizada en base el muestreo de número generado aleatoriamente, pues se observa una mayor oscilación entre los números haciendo que no sea tan exacto y sea más tardado por lo que en base a lo obtenido anteriormente planteamos que es mejor visualizar el comportamiento de PI mediante replicas.

3. Conclusiones

En el uso del método Montecarlo se cae en cuenta de los múltiples valores aproximados que tiene el número pi que resultan ser infinitos. La ventaja de usar el programador phyton es que podemos desarrolar un rango de miles de variables aleatorias como el método lo solicita para tener un número cada vez más cercano a lo que conocemos como 3.1416 todo dependiendo de entre más grande sea el rango de variables destinado.

Además de esto podemos utilizar el programa para gráficar los resultados optenidos y así tener una visualización amplia de las diferentes variaciones por medio de una gráfica.

Referencias

[1] GeoGebra Author. El método montecarlo para estimación de pi, 2016.

[2] J. I. Illana. Métodos monte carlo, Enero 2013.

[3] Grupo Phyton. ¿qué es phyton?, Abril 2020.

[4] P. Watson. Estimación del valor de pi en phyton, Mayo 2021.