

Actividad 3: Generar PI por el método de Montecarlo en python

Luis Carlos Gómez Espinoza
Cesar Mauricio Alvarez Olguín
Fátima Montserrat Zarazúa Uribe
Arturo Mariscal Picon
Gabriel López Escobar
Francisco Emiliano Moreno De Alba

7 de septiembre de 2022

Resumen

El método de Montecarlo, no es determinista o estadístico numérico, es usado para aproximaciones matemáticas complejas y difíciles de evaluar con exactitud. El uso del método de Monte Carlo para aproximar el valor de Pi consiste en ilustrar un cuadrado y dentro de ese cuadrado, dibujar un círculo con diámetro de igual medida que uno de los lados del cuadrado. Posteriormente se dibujan puntos de manera aleatoria (x,y) sobre la superficie diseñada. Finalmente se calcula la proporción de puntos numéricos que se encuentran dentro del círculo y el número total de puntos generados, a partir de dichos datos, se calcula el posible valor de pi. La estimación se hace gracias a un script de python.

1. Introducción

El método Montecarlo puede implementarse para estimar probabilidades que serían muy complicadas de calcular de forma teórica, o para corroborar que ocurrirá lo que se espera en determinado experimento.

Se explorará el método de Montecarlo para el cálculo de pi. Se dará una explicación acerca del método de Montecarlo así como su objetivo principal para la simulación presentada. Este cálculo será analizado más detenidamente haciendo uso como ejemplo de el experimento de la aguja de Buffon, planteada por el naturalista francés Georges-Louis Leclerc de Buffon. Para hacer demostración del método se hará una simulación en Python, un programa que nos aporta muchas ventajas que se mostrarán así como sus distintas aplicaciones; gracias a este y haciendo demostración de nuestro programa se obtendrán los resultados y se mostrarán las gráficas de los cálculos realizados.

2. Desarrollo

2.1. ¿Qué es el método Montecarlo?

La simulación de Montecarlo, o método de Montecarlo, le debe el nombre al famoso casino del principado de Mónaco. La ruleta es el juego de casino más famoso y también el ejemplo más sencillo de mecanismo que permite generar números aleatorios.

La clave de este método está en entender el término ‘simulación’. Realizar una simulación consiste en repetir, o duplicar, las características y comportamientos de un sistema real. Así pues, el objetivo principal de la simulación de Montecarlo es intentar imitar el comportamiento de variables reales para, en la medida de lo posible, analizar o predecir cómo van a evolucionar.

A través de la simulación, se pueden resolver desde problemas muy sencillos, hasta problemas muy complejos. Algunos problemas pueden solucionarse con papel y bolígrafo. Sin embargo, la mayoría requieren el uso de progra-

mas informáticos como Excel, R Studio o Matlab. Sin estos programas, resolver determinados problemas llevaría muchísimo tiempo.

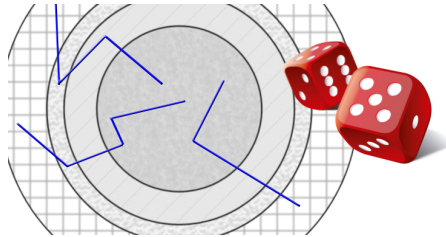


Figura 1: El método Montecarlo debe su nombre al famoso casino del principado de Mónaco

----- Cálculo de Pi por Montecarlo -----

Este método está cerca del experimento de aguja de Buffon, planteada por el naturalista francés Georges-Louis Leclerc de Buffon.

Consideremos al círculo unitario inscrito en el cuadrado de lado 2 centrado en el origen. Dado que el cociente de sus áreas es $\pi/4$, el valor de π puede aproximarse usando Montecarlo de acuerdo al siguiente método:

- 1.- Dibujar un círculo unitario, y al cuadrado de lado 2 que lo inscribe.
- 2.- Lanzar un número "n" de puntos aleatorios uniformes dentro del cuadrado.
- 3.- Contar el número de puntos dentro del círculo, i.e. puntos cuya distancia al origen es menor que 1.
- 4.- El cociente de los puntos dentro del círculo dividido entre "n" es un estimado de, $\pi/4$. Multiplicar el resultado por 4 para estimar π .

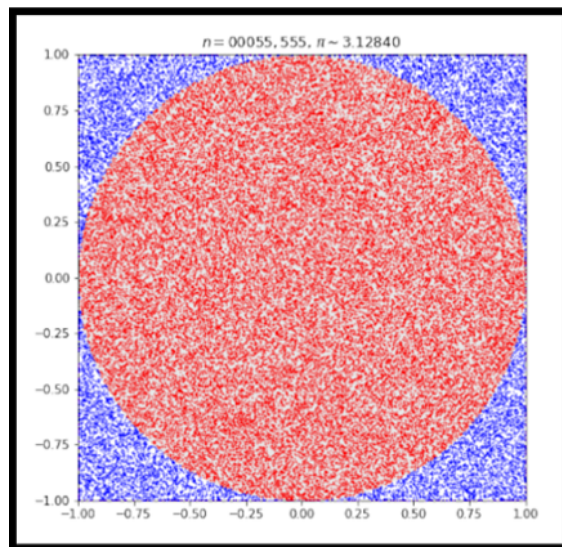


Figura 2: Estimación de Pi por el método de Montecarlo en Python con un número de puntos "n" de 10 a 1,000,000

NOTA: En este cálculo se tienen que hacer dos consideraciones importantes:

- Si los puntos no están uniformemente distribuidos, el método es inválido.
- La aproximación será pobre si solo se lanzan unos pocos puntos. En promedio, la aproximación mejora conforme se aumenta el número de puntos.

2.2. ¿Qué es python?

Python es un lenguaje de programación interpretado cuya principal filosofía es que sea legible por cualquier persona con conocimientos básicos de programación. Además, posee una serie de características que lo hacen muy particular y que, sin duda, le aportan muchas ventajas y están en la raíz de su uso tan extendido:

1. Se trata de un lenguaje open source o de código abierto, por lo que no hay que pagar ninguna licencia para utilizarlo.
2. Su carácter gratuito hace que continuamente se estén desarrollando nuevas librerías y aplicaciones.
3. Combina propiedades de diferentes paradigmas de programación, lo que permite que sea muy flexible y fácil de aprender.
4. Sus aplicaciones no se limitan a un área en concreto.
5. Es apto para todas las plataformas.

El principal obstáculo que se puede encontrar en Python, es que se trata de un lenguaje interpretado, es decir, que no se compila, sino que se interpreta en tiempo de ejecución. Como consecuencia, es más lento que Java o C/C++. Sin embargo, gracias a avances como la computación en la nube, en la actualidad disponemos de una gran capacidad de cómputo a un coste muy asequible.

Su objetivo es la automatización de procesos para ahorrar tanto complicaciones como tiempo, los dos pilares en cualquier tarea profesional. Dichos procesos se reducirán en pocas líneas de código las cuales se han de insertar en una gran variedad de plataformas y sistemas operativos.

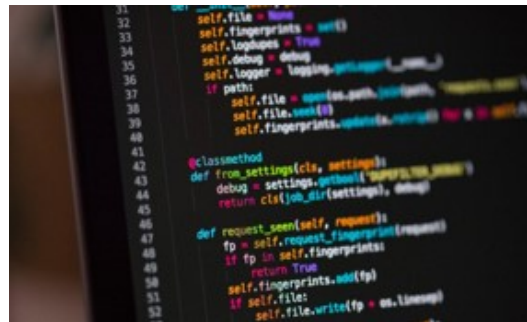


Figura 3: Representación de un código implementado en el área de trabajo de Python

Este lenguaje de programación, se utiliza en prácticamente todas las industrias y campos científicos que pueda imaginarse, como:

- Ciencia de los datos.
- Aprendizaje automático.
- Desarrollo web.
- Educación en Ciencias de la Computación.
- Visión por ordenador y procesamiento de imágenes.
- Desarrollo de juegos.
- Biología y Bioinformática.

Otras áreas como robótica, vehículos autónomos, negocios, meteorología y desarrollo de interfaces gráficas de usuario también se benefician del uso de Python.

2.3. ¿Qué es Pi?

El número pi es un número decimal infinito no periódico famoso por aparecer en muchas fórmulas matemáticas en los campos de la geometría, teoría de los números, probabilidad, análisis matemático y en aplicaciones de física.

En otras palabras, el número pi es un número decimal que no puede expresarse en forma de fracción de enteros (número irracional).

El número pi se representa mediante el siguiente símbolo:



Figura 4: Simbología del número pi

Los primeros dígitos del número pi son:

3,141592653589793115997963468544185161590576171875...

La mayoría de calculadoras tienen una tecla especial para este número y con tan solo activarla nos ahorramos escribir los decimales de pi. Los problemas complejos se resuelven directamente mediante computación.

2.4. Generación y pasos del código en python

1. Se abre la interfaz de Visual Studio Code y se crea un nuevo archivo.

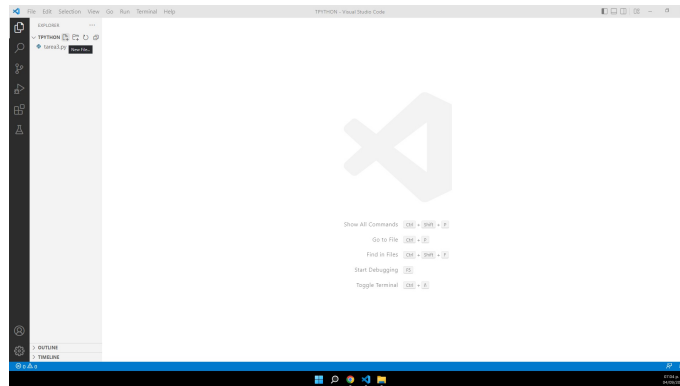


Figura 5: Pantalla inicial de Visual Studio Code en dónde se realiza el código de python.

2.5. Encabezado del programa

2. Se establecen las librerías a utilizar en el código en donde:

- from multiprocessing import Pool: sirve para utilizar los núcleos reales que tienen nuestra computadora en la ejecución del programa.
- from random import randint: se utiliza esta librería para generar los números random que se utilizarán para pi.
- import statistics: se utiliza para el cálculo de valores estadísticos en el campo de los números reales.
- import time: esta librería se utiliza para tomar el tiempo en que se ejecuta el programa.

-import numpy as np: esta librería permite la generación y manejo de datos extremadamente rápido, la cual almacena y opera con datos de manera eficiente.

-import matplotlib.pyplot as plt: esta librería permite generar gráficas, se utilizará para graficar los datos de pi con respecto al tiempo.

```
tarea2.py x
tarea3.py > ...
1 # TAREA NO. 3 (CÁLCULO Y GRAFICO DE PI)
2 # BIOMECHANICA JUEVES N4-N6, EQUIPO 6
3 #Luis Carlos Gómez Espinoza
4 #Cesar Mauricio Alvarez Olguin
5 #Fátima Montserrat Zarazúa Uribe
6 #Arturo Mariscal Picon
7 #Gabriel López Escobar
8 #Francisco Emiliano Moreno De Alba
9 from multiprocessing import Pool
10 from random import randint
11 import statistics
12 import time
13 import numpy as np
14 import matplotlib.pyplot as plt
15
```

Figura 6: Librerías de python utilizadas en el código.

3. Se establecen las dimensiones del cuadrado tales como el ancho y alto, también se genera el radio del círculo que está dentro del cuadrado.

```
15
16 width = 10000
17 height = width
18 radio = width
19
```

Figura 7: Dimensiones declaradas para el ancho, alto y radio de las figuras del método Montecarlo.

4. Se establece la variable para los puntos dentro y fuera del círculo, así como el radio al cuadrado y la cantidad de réplicas que se generarán durante la ejecución del código.

5. Se establece la variable de promedio para pi.

```
19
20 npuntos = 0
21 ndentro = 0
22 radio2 = radio*radio
23 replicas = 250
24 promediopi = []
25
```

Figura 8: Declaración de las variables usadas para el código del método Montecarlo.

2.6. Cuerpo del programa

6. Se genera un condicional if en donde se utilizan los núcleos reales del equipo que se está usando, en este caso son 6 para un i5 10th.

7. Se establece la variable inicio, con la cual se empezará a tomar el tiempo de ejecución del programa.

8. Se abre un ciclo for en donde el rango de este son las réplicas generadas en la ejecución del programa.

9. Se establece el rango de las réplicas que va de 1 a 100000.

10. Se establece la variable t0 para tomar el tiempo de inicio cuando se genera cada dato promediado de pi de las réplicas.

11. Con $x = \text{randint}(0, \text{width})$ se genera un dato aleatorio que este entre 0 y el ancho.

12. Se genera un dato aleatorio que este entre 0 y el ancho con $y = \text{randint}(0, \text{width})$.

13. Se hace un incremento en el número de puntos afuera del círculo cada vez que un punto está fuera de este con `npuntos += 1`.
14. Se genera una condición con `if x*x + y*y` menor igual que `radio2` en donde la suma de los puntos `x*x + y*y` es menor igual que el radio al cuadrado, quiere decir que ese punto está dentro del círculo.
15. Se hace un incremento en el número de puntos dentro del círculo con `ndentro += 1` cada vez que un punto está dentro de este.
16. Se calcula pi con el número de puntos dentro del círculo multiplicado por 4 y dividido entre los puntos que están afuera con `pi = ndentro * 4 / npuntos`.
17. Se obtiene un promedio de los valores de pi obtenidos con `promediopi.append(pi)`.
18. Se toma el tiempo en que se termina de generar ese dato de pi promediado con `t1 = time.time()`.
19. Se realiza la resta del tiempo en que se genera el dato de pi promediado, esto por medio de una resta del tiempo final menos el tiempo inicial con `tf = t1 - t0`.
20. Se obtiene el valor del tiempo en microsegundos con `tm = tf * 1000000`, en dónde 1000000 es la cantidad de microsegundos que hay en un 1 segundo, esto se debe a que el tiempo es muy corto y sí se toma en segundos será una cantidad mínima para graficar.

```

25
26 if __name__ == '__main__':
27     with Pool(6) as p:
28         inicio = time.time()
29         for j in range(replicas):
30             for i in range(1,100000):
31                 t_0 = time.time()
32                 x = randint(0,width)
33                 y = randint(0,width)
34                 npuntos += 1
35                 if x*x + y*y <= radio2:
36                     ndentro += 1
37                 pi = ndentro * 4 / npuntos
38                 promediopi.append(pi)
39                 t_1 = time.time()
40                 t_f = t_1 - t_0
41                 t_m = t_f * 1000000
42                 print(statistics.mean(promediopi))
43                 print("tiempo: {}".format(t_m))
44             final = time.time()
45             delta = final - inicio
46             minutos = delta/60
47             print("tiempo: {}".format(delta))
48             print("tiempo: {}".format(minutos))
49

```

Figura 9: Cuerpo del programa realizado en Visual Studio Code con python.

21. Se imprime el promedio de pi en la consola con `print(statistics.mean(promediopi))`.
22. Se imprime el tiempo que se tarda en generar el dato de pi con `print("tiempo: µs.".format(tm))`.
23. Se toma el tiempo en que se termina de ejecutar el programa para generar los datos de pi con `final = time.time()`.
24. Se realiza la resta del tiempo en que empezó el código y el tiempo en que finalizó con `delta = final - inicio`, delta sirve para obtener el tiempo que tardó en ejecutarse el programa, el tiempo se registra en segundos.
25. Se realiza la conversión de segundos a minutos, creando la variable minutos con `minutos = delta/60`, dónde se divide delta entre la cantidad de segundos que hay en un minuto.
26. Se imprime y muestra el tiempo medido en segundos que tardó en realizarse el programa en la consola con `print("tiempo: s.".format(delta))`.
27. Se imprime y muestra el tiempo medido en minutos que tardó en realizarse el programa en la consola con `print("tiempo: s.".format(minutos))`.

2.7. Final del programa

28. Se genera un vector que se utiliza para determinar los límites de la gráfica con $v=[0,1000000,0,3.8]$, en dónde el rango del eje x para el tiempo de 0 a 1000000 microsegundos, asimismo, el rango del eje y va de 0 a 3.8 para los valores de pi.

29. Se genera la gráfica de los valores de pi con respecto al tiempo con `plt.plot(promediopi,"b--")`, se utiliza b-- para establecer el estilo de la gráfica.

30. Se establece el nombre del eje x de las grafica con `plt.xlabel('Tiempo en microsegundos')`.

31. Se establece el nombre del eje y de la gráfica con `plt.ylabel('Valores de Pi')`.

32. Se establece el título de la gráfica con `plt.title('Tarea 3 Equipo 6 Biomecánica Jueves N3 - N6')`.

33. Se establece el vector para el rango de los ejes de la gráfica con `plt.axis(v)`.

34. Se genera una cuadrícula en la gráfica con `plt.grid()`.

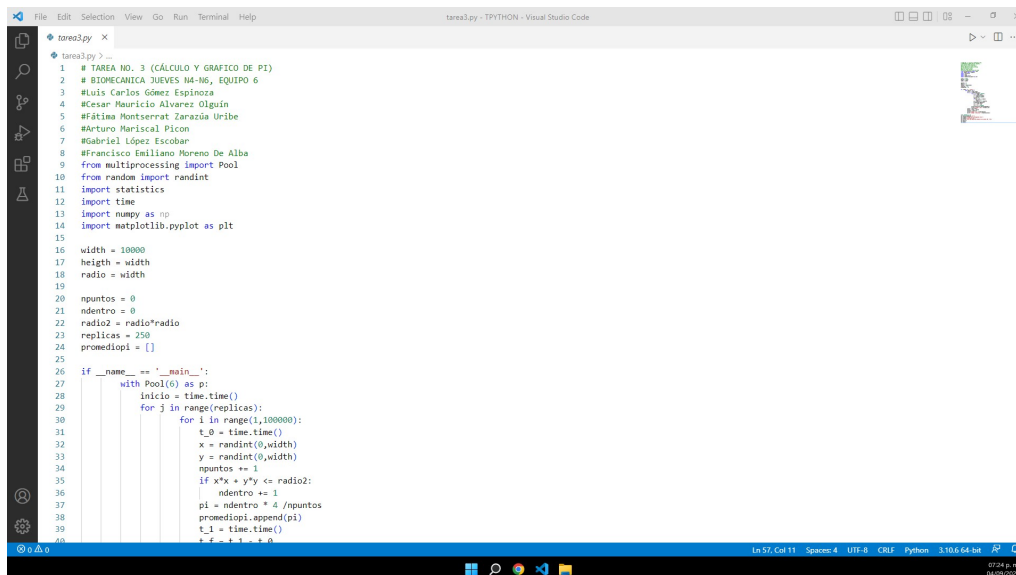
35. Se muestra una ventana con la gráfica con `plt.show()`.

```
49
50 v=[0,1000000,0,3.8]
51 plt.plot(promediopi,"b--")
52 plt.xlabel('Tiempo en microsegundos (μs)')
53 plt.ylabel('Valores de pi')
54 plt.title('Tarea #3 Equipo #6 Biomecánica Jueves N3 - N6')
55 plt.axis(v)
56 plt.grid()
57 plt.show()
```

Figura 10: Final del programa realizado en Visual Studio Code con python.

2.8. Código en python, ejecución del programa y gráfica obtenida de valores de pi con respecto al tiempo.

-----Pantallas completas del programa realizado en Visual Studio Code con python-----



```
tarea3.py > ...
1 # TAREA NO. 3 (CÁLCULO Y GRÁFICO DE PI)
2 # BIOMECAÁNICA JUEVES N4-N6, EQUIPO 6
3 #Luis Carlos Gómez Espinoza
4 #Cesar Mauricio Alvarez Olguín
5 #Fátima Montserrat Zarazúa Uribe
6 #Arturo Hariscal Picon
7 #Gabriel López Escobar
8 #Francisco Emiliano Moreno De Alba
9 from multiprocessing import Pool
10 from random import randint
11 import statistics
12 import time
13 import numpy as np
14 import matplotlib.pyplot as plt
15
16 width = 10000
17 height = width
18 radio = width
19
20 npuntos = 0
21 ndentro = 0
22 radio2 = radio*radio
23 replicas = 250
24 promediopi = []
25
26 if __name__ == '__main__':
27     with Pool(6) as p:
28         inicio = time.time()
29         for j in range(replicas):
30             for i in range(1,100000):
31                 t_0 = time.time()
32                 x = randint(0,width)
33                 y = randint(0,width)
34                 npuntos += 1
35                 if x*x + y*y <= radio2:
36                     ndentro += 1
37                 pi = ndentro * 4 / npuntos
38                 promediopi.append(pi)
39             t_1 = time.time()
40             t = t_1 - t_0
```

Figura 11: Código del programa realizado en Visual Studio Code con python.

```

20 npuntos = 0
21 ndentro = 0
22 radio2 = radio*radio
23 replicas = 250
24 promediopi = []
25
26 if __name__ == '__main__':
27     with Pool(6) as p:
28         inicio = time.time()
29         for j in range(replicas):
30             for i in range(1,100000):
31                 t_0 = time.time()
32                 x = randint(0,width)
33                 y = randint(0,width)
34                 npuntos += 1
35                 if x*x + y*y <= radio2:
36                     ndentro += 1
37                 pi = ndentro * 4 / npuntos
38                 promediopi.append(pi)
39                 t_1 = time.time()
40                 t_f = t_1 - t_0
41                 t_m = t_f * 1000000
42                 print(statistics.mean(promediopi))
43                 print("tiempo: {}us.".format(t_m))
44             final = time.time()
45             delta = final - inicio
46             minutos = delta/60
47             print("tiempo: {}s.".format(delta))
48             print("tiempo: {}minutos.".format(minutos))
49
50 v=(0,1000000,0,3.8)
51 plt.plot(promediopi,"b--")
52 plt.xlabel("tiempo en microsegundos (us)")
53 plt.ylabel("Valores de pi")
54 plt.title('Tarea #3 Equipo #6 Biomecánica Jueves N3 - N6')
55 plt.axis(v)
56 plt.grid()
57 plt.show()

```

Figura 12: Código del programa realizado en Visual Studio Code con python.

- - - - -Ejecución y gráficas del programa realizado en Visual Studio Code con python- - - - -

Se muestra la ejecución del programa y la gráfica obtenida en Visual Studio Code con python, en dónde se observan los distintos valores de PI obtenidos con respecto al tiempo que se generó cada uno, además, se muestra un acercamiento a ciertos valores de la gráfica al azar, ya que sin hacer esto no es posible observarlos con claridad debido a que son una gran cantidad en un espacio pequeño.

```

tiempo: 0.0us.
3.141073165625767
tiempo: 0.0us.
3.1410737951704077
tiempo: 0.0us.
3.141074436358654
tiempo: 0.0us.
3.141075072096591
tiempo: 0.0us.
tiempo: 1866.9874281883244.
tiempo: 31.116457136472068minutos.

```

Figura 13: Ejecución del código del programa realizado en Visual Studio Code con python.

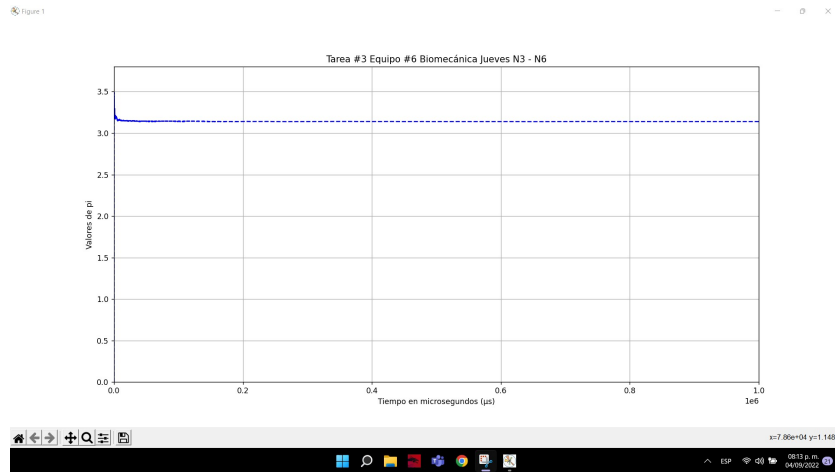


Figura 14: Gráfica obtenida de valores de PI con respecto al tiempo.

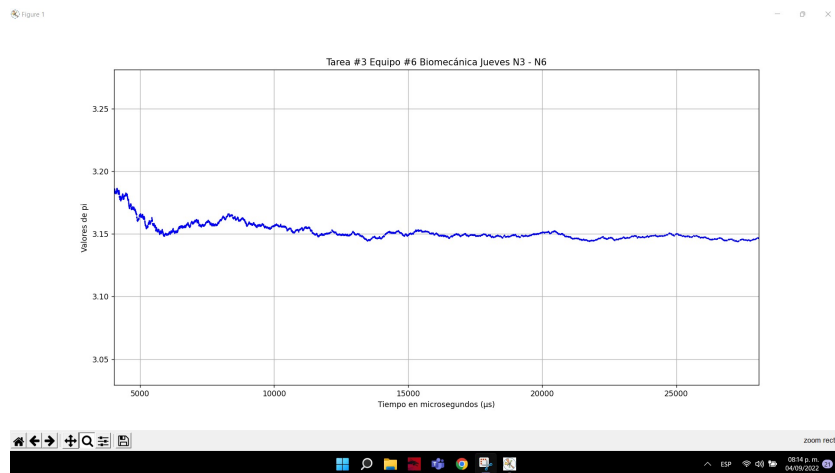


Figura 15: Acercamineto a los valores de pi respecto al tiempo en la gráfica obtenida.

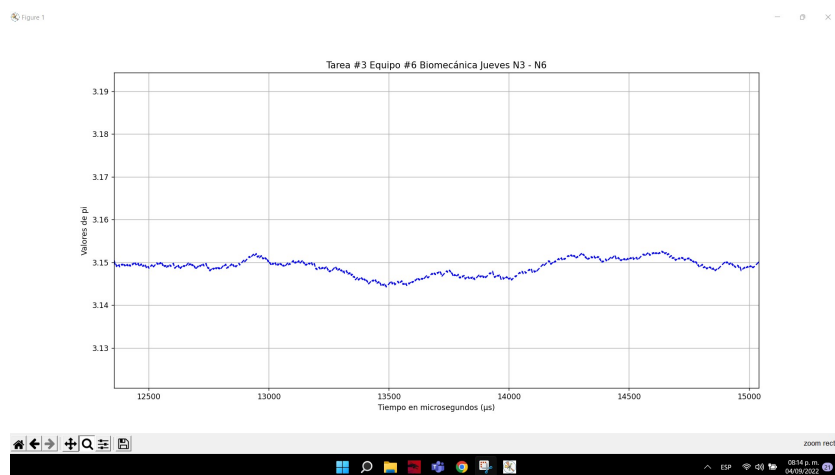


Figura 16: Acercamineto aumentado a los valores de pi respecto al tiempo en la gráfica obtenida.

3. Conclusiones

Se muestra la ejecución del programa y la gráfica obtenida en Visual Studio Code con python, en dónde se observan los distintos valores de PI obtenidos con respecto al tiempo que se generó cada uno, además, se muestra un acercamiento a ciertos valores de la gráfica al azar, ya que sin hacer esto no es posible observarlos con claridad debido a que son una gran cantidad en un espacio pequeño.

Siempre hemos visto el valor de pi para calcular el área de un círculo, algunas veces hemos visto de donde se obtiene, sin embargo el calcularlo por medio de un programa en python es algo nuevo y muy interesante como por diferentes caminos podemos llegar a mas resultados, creo que más allá de saber el valor de pi por medio de valores que se generan, es una oportunidad para poder calcular cosas diferentes en el futuro además de conocer mas herramientas para obtener algún resultado.