"P5" Método Monte-Carlo

NESTOR

Marzo 2022

1. Objetivo

El objetivo de esta actividad es el estudiar como es el comportamiento en la situación que se combinan en los conceptos estadísticos como lo es el muestreo aleatorio, con la generación de números aleatorios y la automatización de los cálculos. Es un procedimiento matemático que consiste en la generación numérica de series mediante un muestreo aleatorio de las distribuciones de probabilidad [1].

2. Desarrollo

El desarrollo de la actividad está basado en el código implementado por E. Schaeffer [3]. Basando el desarrollo, en el código [3] se establecen las instrucciones para la ejecución del programa. Se encuentra todas las instrucciones en el repositorio de Nestor en GitHub.

Código 1: Ejecución de Parámetros

```
desde = 2
hasta = 5
pedazo = 50000
cuantos = [500, 5000, 50000]
```

En una receta y ejecutando con la función, se genera una cantidad de números aleatorios con una distribución de acuerdo a la integral normalizada. Esto se define un conjunto de funciones que se utilizan para generar o manipular números aleatorios a través del módulo aleatorio. Las funciones del módulo aleatorio se basan en una función generadora de números pseudoaleatorios random, que genera un número flotante aleatorio entre 0,0 y 1,0 [1]

Código 2: Instrucciones para Generador Random

```
from GeneralRandom import GeneralRandom
generador = GeneralRandom(np.asarray(X), np.asarray(Y))

def parte(replica):
    V = generador.random(pedazo)[0]
    return ((V >= desde) & (V <= hasta)).sum()

def compare_strings(a, b):
    a = str(a)
    b = str(b)

    if a is None or b is None:
        return 0

    size = min(len(a), len(b))
    count = 0

    for i in range(size):
        if a[i] == b[i]:
            count += 1
        else:
            break
    return count
```

¿Qué es el multiprocesamiento de importación en Python? multiprocesamiento es un paquete que admite procesos de generación mediante una API similar al módulo de subprocesamiento. El paquete de multiprocesamiento

ofrece simultaneidad tanto local como remota, eludiendo efectivamente el bloqueo global del intérprete mediante el uso de subprocesos en lugar de subprocesos [1] .

Código 3: Instrucción para calcular iteraciones

```
import multiprocessing
if __name__ == "__main__":
  with multiprocessing.Pool() as pool:
    for c in cuantos:
    p = c * pedazo
    montecarlo = pool.map(parte, range(c))
    integral = sum(montecarlo) / p
```

Se proporciona una serie de funciones y clases útiles para gestionar los subprocesos y las comunicaciones entre ellos. Las iteraciones del análisis y se calculan los valores de los errores para poder comparar las estimaciones de la integral definida dependiendo de la cantidad de iteraciones para representar los resultados.

Código 4: Resultados de iteraciones

```
resultados = {'Iteraciones': puntos,
                      'Error Absoluto': ae,
                      'Error Cuadrado': se,
                      'Decimales Correctos': dec}
        df = pd.DataFrame(resultados)
        sns.barplot(data=df, x='Iteraciones',
                    y='Error Absoluto',
                    dodge=False)
        plt.savefig('AbsErr.png')

        plt.show()
        sns.barplot(data=df, x='Iteraciones',
                    y='Error Cuadrado',
                    dodge=False)
        plt.savefig('SqErr.png')
        plt.show()
        sns.barplot(data=df, x='Iteraciones',
                    y='Decimales Correctos',
                    dodge=False)
        plt.savefig('Decimals.png')
        plt.show()
        print(df)
```

3. Resultados

En los diagramas se representa como es el comportamiento de las variaciones e iteraciones cuando se da valores al momento de estimar un resultado.

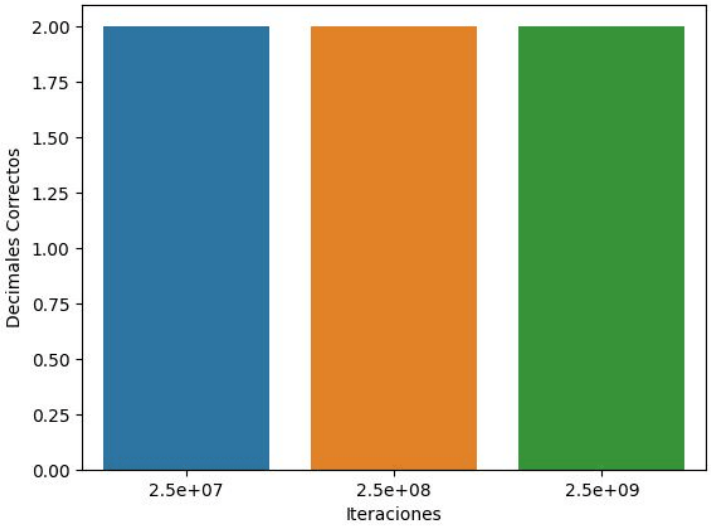


Figura 1: Diagrama decimal.

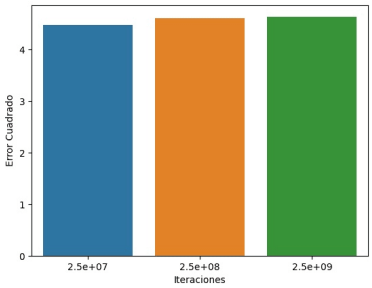
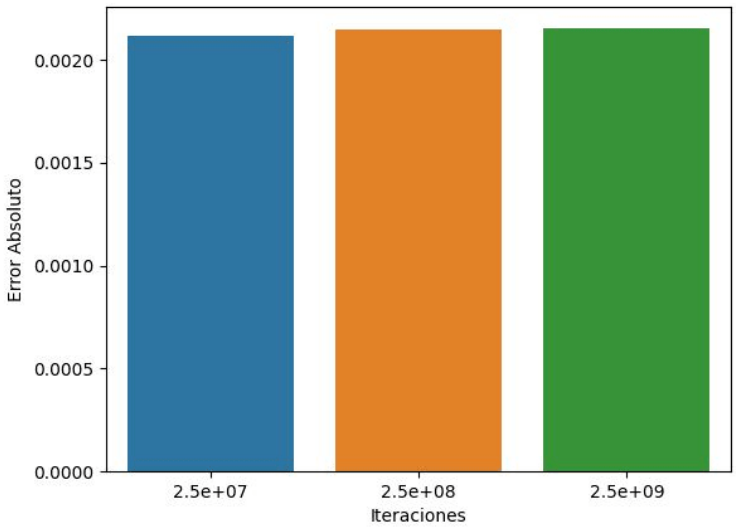


Figura 2: Diagrama Absoluto.



4. Conclusiones

Se concluye que las iteraciones representa una ligera variación va aumentando con respecto a la cantidad de iteraciones. Sin embargo cuando varía esto puede cambiar los valores para ser exactos debido a la ejecución.

[3] [2] [1]

Referencias

[1] J. Hunter. Lines, bars and markers. *Repositorio, GitHub*, 2021.

[2] J.FeroxDeitas. Github,pasos$_a$*leatorios.py.URL*.

E. Schaeffer. Github,montecarlo. URL https://github.com/satuelisa/Simulation/tree/master/MonteCarlo.