Práctica 5
Método Monte-Carlo

Edson Edgardo Samaniego Pantoja
Materia: Simulación computacional
Fecha 17 de marzo de 2021

1. Introducción

El método Monte-Carlo proporciona soluciones aproximadas a una gran variedad de problemas matemáticos haciendo posible la realización de experimentos con muestreos de números pseudoaleatorios en una computadora es idóneo para situaciones en las cuales algún valor o alguna distribución no se conoce y resulta complicado de determinar de manera analítica.

2. Metodología

Se requiere aproximar el resultado de la siguiente integral:

$\int_3^7 \frac{dx}{\exp(x)+\exp(-x)}$

Se generan números pseudoaleatorios con distribución $g(x) = \frac{2f(x)}{\pi}$, donde $f(x) = \frac{1}{\exp(x)+\exp(-x)}$. Para la integral ya que se requiere el resultado para compararlo con lo que se obtiene, se hace uso de la página WolframAlpha [3], donde se ingresa la integral con sus límites y da un resultado de `0.048834`. Para la práctica es utilizado un método de distribución de probabilidad que da solo valores positivos y todos estos juntos da uno, por decir representan un universo completo. Es dado un código en el repositorio de Schaeffer [2] donde se puede realizar el gráfico de distribución de probabilidad como el de la figura 1, en el cual nosotros obtendremos la proporción dentro del intervalo de tres a siete, dentro del gráfico se genera un histograma de números pseudoaleatorios que siguen la función teórica analítica de $g(x)$.

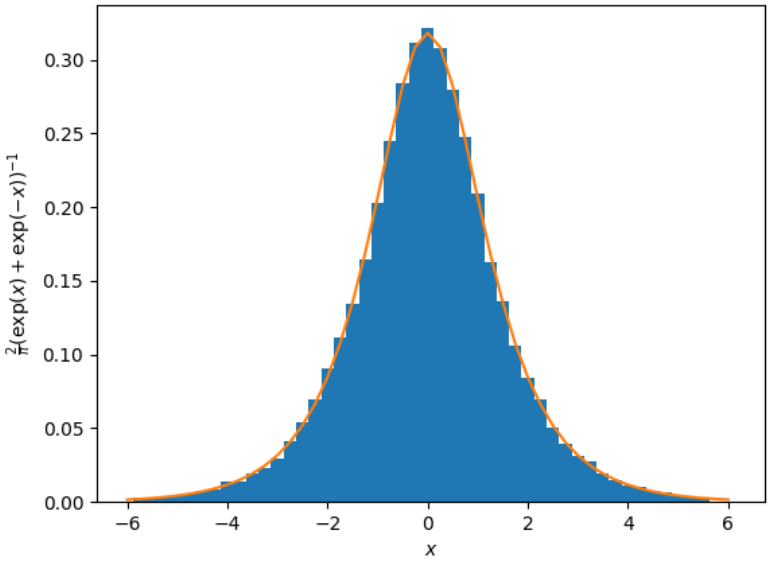

Figura 1: Gráfica de distribución de probabilidad e histograma de $g(x)$ [1].

Al tener la generación pseudoaleatoria se puede estimar la integral en el programa, utilizando como apoyo el código del repositorio de Schaeffer [2] el cual toma una muestra especificada dentro del intervalo de tres a siete, a partir de esto se saca un porcentaje de la cantidad que hay en el intervalo y la muestra tomada se sabrá el porcentaje que hay en esa fracción de valores con esto se renormaliza para recuperar el integral mediante funciones especificadas y nos retorna un valor parecido al obtenido en WolframAlpha.

3. Objetivo

El objetivo de la practica es que con el código explicado anteriormente se pueda comparar el resultado de WolframAlpha de `0.048834` contra el resultado que arroja el programa. Más concreto y especifico se hará un análisis donde se comparará a partir de dos, tres, cuatro y cinco decimales realizando varias replicas para asegurar un porcentaje de eficiencia en cada decimal de precisión y en cada prueba con cada decimal se varia el número de muestras ya que esto afecta que tanta eficiencia hay conforme los decimales aumentan. Lo importante es determinar el tamaño de muestra y representar el resultado como una sola gráfica con el número de decimales correctos contra el tamaño de muestra para una tasa de éxito que se elija.

4. Simulación

En el programa lo principal que se realiza es el cambio de decimales que para este caso se examinaran dos, tres y cuatro decimales, así que entrarán a un ciclo `for` que hará esta función, dentro del ciclo se asigna el valor de `n` en condiciones `if` de cada decimal y estas tienen su propio `n` que son el numero de muestras. El código completo puede ser consultado en el repositorio de Samaniego [1].

```
desde = 2.96
hasta = 7
replicas = 100

for decimales in (2, 3, 4):
    if decimales == 2:
        n=81000
    if decimales == 3:
        n=530000
    if decimales == 4:
        n=880000
```

Una vez que entra al primer ciclo con los primeros dos decimales y se le asigna el número de muestras, procede a otro ciclo `for` que es el generador pseudoaleatorio explicado anteriormente el cual da un resultado aproximado y variante dentro de la variable `numero`. Lo siguiente es como se comparan los decimales del valor recibido y el valor fijo nombrados `valor1` y `valor2` respectivamente, el método fue convertir los valores a `string` para de esta manera lograr separar cada dígito sólo que esta en formato de cadena así que después de tener los dos valores se procede a un ciclo que toma valor de la cantidad de decimales a leer y dentro los dos valores son convertidos a valores numéricos enteros (no cadena) para compararlos dato por dato y almacenarlo para después tomar una decisión donde si detecta un valor que es distinto dentro de los decimales comparados, este sumara un entero a la variable `similar` y al final tener la suma de cuantas veces no coincidió el número pseudoaleatorio con el valor fijo.

```
    for C in range(replicas):
        generador = GeneralRandom(np.asarray(X), np.asarray(Y))
        V = generador.random(n)[0]
        mc = ((V >= desde) & (V <= hasta))
        numero=((pi / 2) * integral)
        valor1= 0.048834
        valor2= numero
        for x in str(valor1):
            digito1.append(x)
        for y in str(valor2):
            digito2.append(y)
        for z in range(decimales):
            A=(int(digito1[z+2])==int(digito2[z+2]))
            comparacion.append(A)
        if all(s==True for s in comparacion)== False:
            similar.append(1)
```

Una vez acumulado cuantas replicas difirieron se procede a sacar el porcentaje de efectividad utilizando una regla de tres donde a las `replicas` realizadas se le esta restando la acumulación de diferencias por cien y dividido entre el total de `replicas`.

```
        cuantos=(len(similar))
        porcentaje = (((replicas-cuantos)*100)/(replicas))
        print(porcentaje,'%')
```

5. Ajustes de programa

Este apartado consta de cambios en la programación base donde se produce la integral y sus límites, debido a qué con dichos rangos de tres a siete se hacen pruebas con un número de muestras alto de hasta quinientos mil y se muestra que el resultado no pasa la condición de más de dos decimales coincidentes a el valor fijo de `0.048834`, por lo que se opta por ampliar los límites para un mayor intervalo de muestras quedando de 2.96 a siete, así es como hubo una mejora y un mayor porcentaje de coincidencia en tres decimales. Otro cambio realizado en el programa o más bien una decisión que se tomo es no utilizar cinco decimales debido a que el número de muestras era excesivamente alto y la computadora tardaría mucho tiempo en procesar las cien replicas y el porcentaje de efectividad sería muy bajo.

6. Resultados

Las pruebas se realizaron en repetidas ocasiones para poder obtener los resultados mas estables a una tasa de éxito fija de cada decimal por ejemplo para dos decimales se fijo la tasa a `90%` con una cantidad de ochenta y un mil muestras que se puede observar en el cuadro 1.

Cuadro 1: Registro de muestras por cada decimal de precisión, número de replicas realizadas en la prueba y el porcentaje de efectividad obtenido del total de replicas.

Decimales	Muestras	Replicas	Porcentaje de efectividad
2	81000	100	90 %
3	530000	100	80 %
4	880000	100	10 %

Para mejor representación de los datos tabulados se realiza un gráfico de puntos de cada decimal de precisión y la cantidad de muestras realizadas para su tasa de efectividad 2 y adjunto a esto una línea que muestra el cambio o incremento de muestras conforme se busca más precisión por decimal, se puede observar que el cambio de dos a tres decimales implicó un aumento de muestras de 449000 aunque el porcentaje sólo se estableció de `80%` debido a la tardanza de la obtención de datos, y para los cuatro decimales el porcentaje fue mas bajo de `10%` debido a que las muestras fueron muy altas 880000 obteniendo una diferencia de muestras de 350000 entre tres y cuatro decimales.

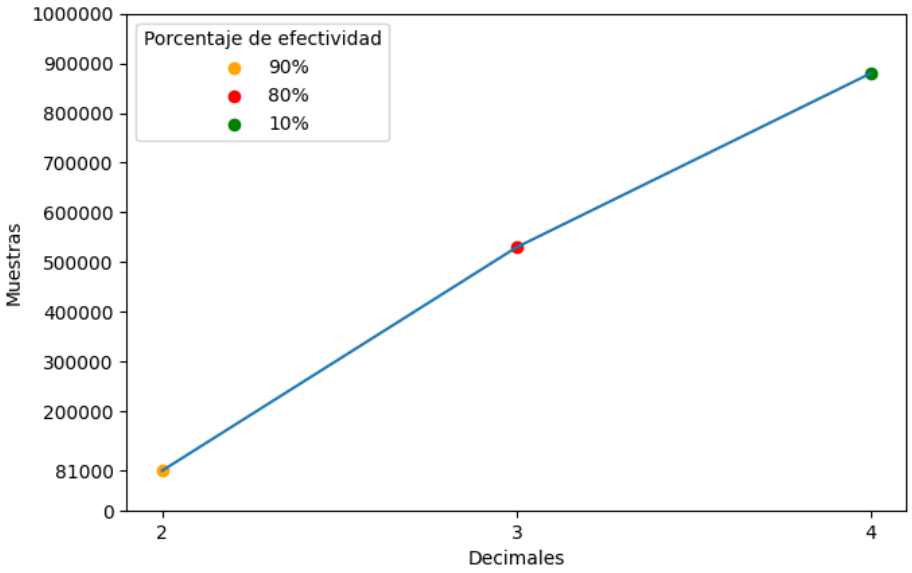

Figura 2: Gráfica de cantidad de muestras por decimal de precisión y tasa de éxito.

Por último se representa en un diagrama caja-bigote 3 de cada decimal como fueron sus replicas en lo que respecta a su distribución y se puede observar que conforme aumentaban los decimales de precisión, el número ya no era tan disperso ya que se acercaba más al valor obtenido en WolframAlpha pero vemos que en promedio la mayor cantidad de datos se mantuvo en el rango de `0.048` y `0.049`.

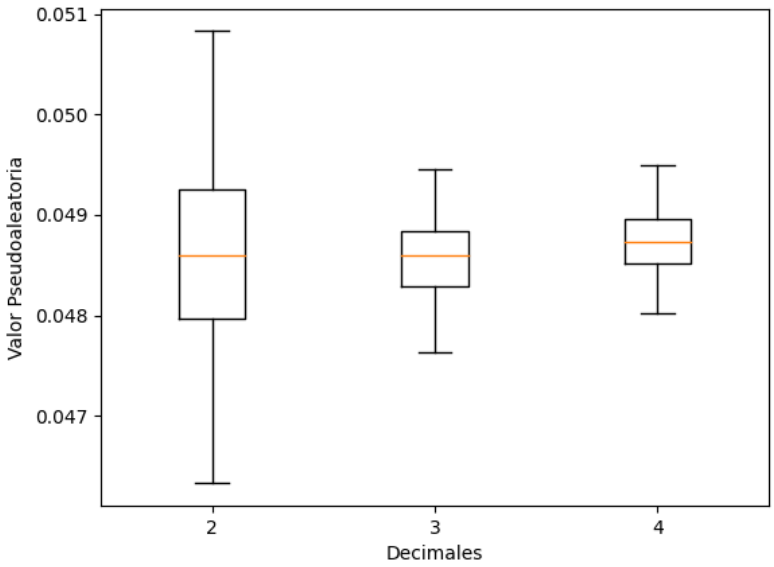

Figura 3: Caja-bigote por cada decimal y su distribución de replicas de valor pseudoaleatorio.

Referencias

[1] Samaniego E. Práctica 5, 2021. URL https://github.com/edson-samaniego/simulation-2021/tree/main/Pr%C3%A1ctica-5.

[2] Schaeffer E. Monte-carlo, 2021. URL https://github.com/satuelisa/Simulation/blob/master/MonteCarlo/integral.py.

[3] Wolfram S. Wolfram alpha, 2021. URL https://www.wolframalpha.com/input/?i=integrate+%282%2Fpi%29+*+%281%2F%28exp%28x%29%2Bexp%28-x%29%29%29+from+-infty+to+infty.