Práctica 5: método Monte-Carlo

Raul L.

16 de marzo de 2022

1. Introducción

El método Monte Carlo es idóneo para situaciones en las cuales algún valor o alguna distribución no se conoce y resulta complicado de determinar de manera analítica. Siguiendo los ejemplos de Kurt, paralelicemos algunos casos sencillos en esta práctica. Supongamos que se ocupa conocer el valor de una integral que no se nos antoja resolver para nada, como, por ejemplo

$$\int_3^7 f(x)\,dx$$

para

$$f(x) = \frac{1}{exp(x) + exp(-x)}$$

Por suerte, $2f(x)/\pi$ es una función de distribución valida, ya que

$$\int_{-\infty}^{\infty} \frac{2}{\pi} f(x)\,dx = 1$$

Este hecho nos permite generar números pseudoaleatorios con la distribución g(x) = 2 f(x)/ π,
así estimar

$$\int_3^7 g(x)\,dx$$

y de ahí normalizar el estimado para que sea

$$\int_3^7 f(x)\,dx$$

Se puede comparar con el resultado aproximado de Wolfram Alpha, 0.048834, para llegar a una satisfacción que no estemos completamente mal. Se debe notar que cada ejecución dará un resultado distinto ya que es una muestra pseudoaleatoria [**?**].

2. Objetivo

Estudia estadísticamente la convergencia de la precisión del estimado del integral con en método Monte Carlo, comparando con el valor producido por Wolfram Alpha, en términos del (1) error absoluto, (2) error cuadrado y (3) cantidad de decimales correctos, aumentando el tamaño de muestra [2].

3. Código

Para este código se utilizaron un numero alto de repeticiones repeticiones para cada pedazo, esta cantidad fue con la que se observó que se podía llegar a cuatro decimales de precisión. No se pudo llegar a observar los resultados con más repeticiones por falta de poder de procesamiento.

Código en Python

https://github.com/satuelisa/Simulation/blob/master/MonteCarlo/rng.py

Código creado en Python

https://github.com/Raullr28/Resultados/blob/main/P4/codigo_celdas.py

```
vg = np.vectorize(g)
X = np.arange(-8, 8, 0.05) # ampliar y refinar
Y = vg(X) # mayor eficiencia
correcto= 0.048834111126049310840642 37
print("correcto",correcto)
desde = 2.96
hasta = 7
pdz = (700,5000,10000,80000,300000,1000000)#,5000000)#numero de n para estimar valor
repeticiones = 200
result = {"Estimado": [], "abs": [], "cuad": [], "dec": []}
dec = {"cero": [], "uno": [], "dos": [], "tres": [], "cuatro": [], "cinco": []}
ABS, CUAD = [],[]
for pedazo in pdz:
    print("############## pedazos:",pedazo,"######################")
    absoluto=[]
    cuadrado=[]
    dec_corr=[]
    for i in range(repeticiones):
        generador = GeneralRandom(np.asarray(X), np.asarray(Y))
        V = generador.random(pedazo)[0]
        montecarlo = ((V >= desde) & (V <= hasta))
        integral = sum(montecarlo) / (pedazo)
        estimado=(pi / 2) * integral
        absoluto.append(abs(correcto - estimado))
        cuadrado.append(((correcto - estimado)**2))
        dec_corr.append(decimales(correcto, estimado))#regresa cuantos decimales hubo semejantes
```

Código 1: Representa la automatización para variar el pedo.

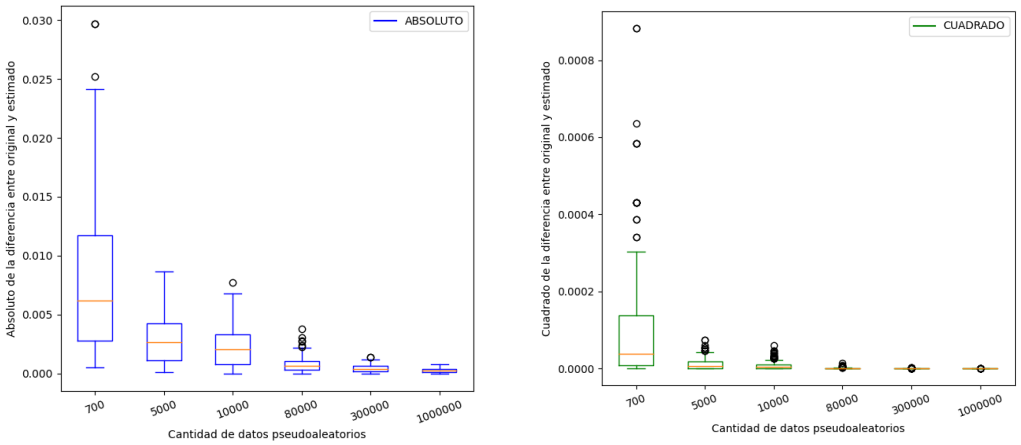
```
def g(x):
    return (2  / (pi * (exp(x) + exp(-x))))

def decimales(real, obtenido):
    contador=-2 #omite el 0 y el punto . del conteo
    real, obtenido= (str(real)), (str(obtenido))# convierte para leer cada valor
    obtenido=obtenido[:(len(real))]# recorte para mismo tamaño
    largo=min([len(real),len(obtenido)])
    for i in range(largo):
        if real[i] == obtenido[i]:
            contador=contador+1
        else:
            break
    return(contador)
```

Código 2: Representación función decimales.

4. Resultados

En las figuras se muestra la probabilidad que existió en cada una de los diferentes partes con un número de repeticiones de 100, esto se realizó para poder dar un resultado estadístico ya que con esto podíamos comparar la probabilidad de encontrar los decimales correctos en cada una de las diferentes partes.



(a) Absoluto

(b) Cuadrado

Figura 1: Gráfica comparativa.

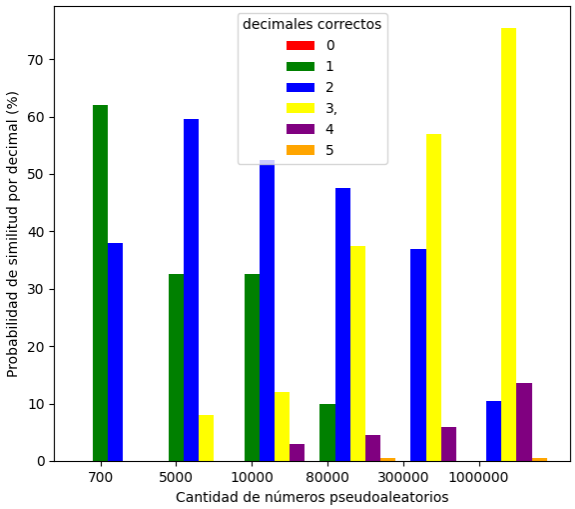
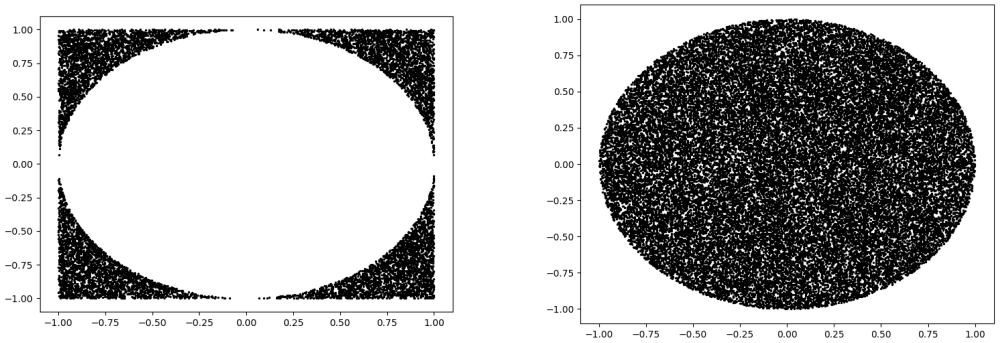


Figura 2: Gráfica estadística de porcentaje.

5. Reto 1

El primer reto es realizar lo mismo para la estimación del valor de Π de Kurt [1].

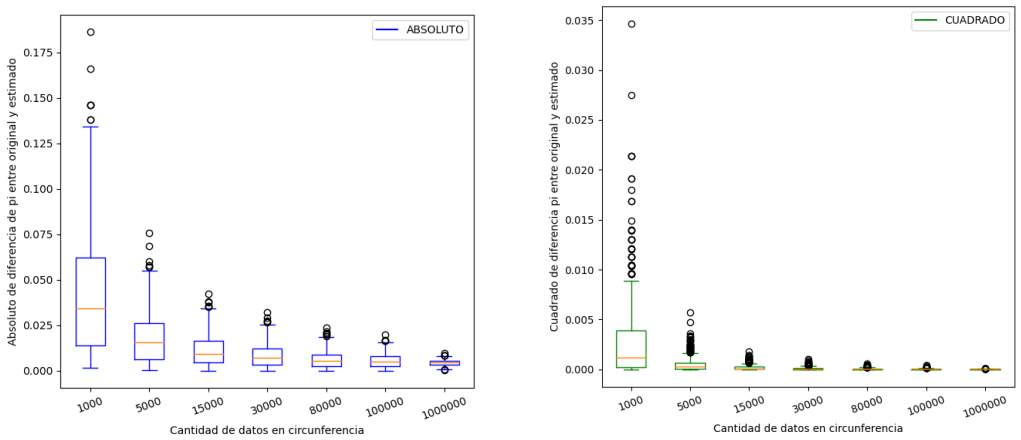


(a) Puntos dentro del circulo

(b) Puntos fuera del circulo

Figura 3: Imagen de crecimiento aleatorio de puntos.

6. Resultados



(a) Absoluto (b) Cuadrado

Figura 4: Gráfica comparativa.

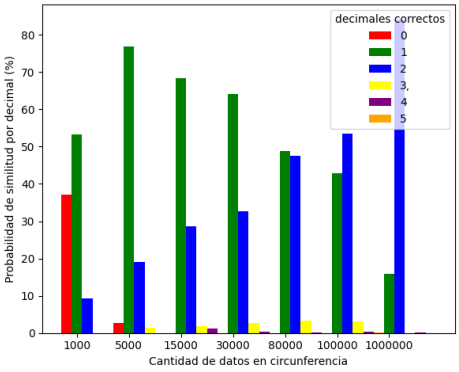


Figura 5: Gráfica estadística de porcentaje.

7. Conclusión

Se demostró que aumentando el numero de repeticiones se podría llegar a tener un porcentaje alto de números decimales correctos.

Referencias

[1] W. Kurt. 6 neat tricks with monte carlo simulations — count bayesie; probably a probability blog. 2015. URL `https://www.countbayesie.com/blog/2015/3/3/6-amazing-trick-with-monte-carlo-simulations`.

[2] E. Schaeffer. Práctica 4: diagramas de voronoi. 2022. URL `https://satuelisa.github.io/simulation/p5.html`.