Nombre: Juan Manuel Varela

Entrega Métodos de Monte Carlo

Unidad 4 – Sesión 11 - Ejercicio 11.1

Descripción del problema:

Para generar un punto aleatorio (X_1, X_2) en un círculo de centro (0, 0) y radio 1, es posible hacerlo de la forma siguiente (derivación disponible en las páginas 234 y 235 del libro de referencia del curso, "Monte Carlo: concepts, algorithms and applications", Fishman 1996):

• se genera un valor aleatorio r, de distribución $F_r(x) = x^2$ para $0 \le x \le 1$, y 0 para cualquier otro x;

• se generan dos v.a. independientes Z_1 y Z_2 de distribución normal (0, 1);

• se calcula $X_1 = rZ_1/\sqrt{{Z_1}^2 + {Z_2}^2}$ y $X_2 = rZ_2/\sqrt{{Z_1}^2 + {Z_2}^2}$.

Utilizar esta propiedad para volver a resolver el Ejercicio 6.1 parte a, pero generando únicamente valores de puntos dentro del círculo de base de la montaña:

Problema: se idealiza una montaña como un cono inscrito en una región cuadrada de lado 1 km. La base de la montaña es circular, con centro en (0.5, 0.5) y radio r = 0.4km, y la altura es H = 8km. La altura de cada punto (x, y) de la montaña está dada por la función: $f(x,y) = H - H/r \times \sqrt{(x - 0.5)^2 + (y - 0.5)^2}$, en la zona definida por el círculo, y 0 fuera del círculo. El volumen total de la montaña (en km cúbicos) puede verse como la integral de la función altura en la región.

• Parte a: escribir un programa para calcular el volumen por Monte Carlo. Realizar 10^6 replicaciones y estimar el valor de ζ y el error cometido (con nivel de confianza 0.95), utilizando como criterio la aproximación normal.

Comparar la precisión obtenida con la alcanzada en el ejercicio 6.1.

Sugerencia: tener en cuenta que al estar generando puntos dentro del círculo, estamos calculando una integral de Lebesgue-Stieltjes, por lo que es necesario ajustar el integrando de manera que quede explícita la integral en la forma $\int k(z)dF(z)$, con z un vector. En particular, por ser un sorteo uniforme dentro del círculo, la densidad de probabilidad en el círculo es 1/(´área del círculo), y 0 afuera del mismo.

Solución a aplicar en Python:

1. Importar random
2. Importar math
3. Importar time
4. Importar norm de scipy.stats
5. Definir semilla
6. Leer número de replicaciones
7. Leer δ, tal que 1- δ sea el nivel de confianza deseado
8. Definir función: k(x,y): """Evalúa la función $f(x,y) = H - H/r \times \sqrt{(x - 0.5)^2 + (y - 0.5)^2}$ en el punto (x,y)"""
   8.1. centro=(0.5, 0.5)
   8.2. radio=0.4
   8.3. H=8

8.4. Distancia= distancia de (x,y) al centro
8.5. Altura=H-H/radio*distancia
8.6. Salida: Altura

9. Definir función: generar_punto(): """Genera un punto aleatorio dentro del círculo de centro (0.5,0.5) y radio 0.4, con distribución uniforme"""
   9.1. Sortear valor aleatorio r, de distribución $F_r(x) = x^2$ para $0 \leq x \leq 1$, y 0 para cualquier otro x
   9.2. Sortear Z_1 y Z_2 con distribución normal (0,1)
   9.3. Calcular X_1 y X_2 ajustando para que el centro sea (0.5,0.5) y el radio 0.4
   9.4. Salida: [X_1, X_2]
10. Definir función: integración_m_l_s(función, dF, n, nivel): """Estima por monte carlo el valor de la integral la de Lebesgue-Stieltjes de la función utilizando un tamaño de muestra n y halla un intervalo de confianza del nivel deseado"""
    10.1. S=0, T=0 #Inicialización
    10.2. For i entre 1 y n:
       10.2.1. Sortear $Punto_{(i)}$ con distribución uniforme en el círculo de centro (0.5,0.5) y radio 0.4
       10.2.2. If i>1 then $T=T+(1-1/i)*(funcion(Punto_{(i)})/dF-S/(i-1))^2$
       10.2.3. $S = S + funcion(Punto_{i(i)})/dF$ #Acumular en S y T*
    10.3. Integral= S/n #Estimador puntual de la integral
    10.4. Varianza_función= T/(n-1) #Estimador puntual de la varianza de función(Punto)
    10.5. Varianza_integral= Varianza_función/n #Estimador puntual de la varianza de la integral
    10.6. Cálculo [I_1(S,n,1-nivel), I_2(S,n,1-nivel)] #Intervalo de confianza del nivel ingresado para la integral
    10.7. Salida: Integral, error, Varianza_función
11. Inicio = tiempo de inicio
12. Estimación, error_est, varianza_fi= integración_monte_carlo(k, 1/área del círculo, n, 1- δ)
13. Tiempo_ejecución=Tiempo final-inicio #Cálculo del tiempo de ejecución
14. Escribir: “Estimador: {Estimación}, Error: {error_est}, Tiempo (segundos): {Tiempo_ejecución}

Resultados:

Plataforma de cómputo: PC de escritorio, procesador Intel I5-12400, RAM: 32 Gb, Windows 10

Semilla: 6
$n=10^6$
$\delta=0.05$
$\bar{\zeta}$= 1,340586 km^3
ε= 0,001858 km^3
Tiempo de ejecución: 2,420248 segundos

Entonces el intervalo de confianza es (1,338728km^3; 1,342444km^3)

Si se compara con el ejercicio 6.1.a, cuyo resultado utilizando la misma semilla, número de replicaciones y nivel de confianza fue el intervalo (1,338694km^3;1,346098km^3), se tiene que el intervalo obtenido tiene un largo de aproximadamente la mitad, por lo que se podría decir que este método da estimaciones más cercanas al volumen real. Esta mejoría no se obtiene de

manera gratuita, ya que en este caso el tiempo de ejecución fue un 81% mayor al del ejercicio 6.1.a.

Anexos:

Log de ejecuciones:

```
runcell(0, 'C:/…/ejercicio11.1.py')
Ingrese el número de replicaciones, n: 1000000
Ingrese delta, tal que el nivel de confianza sea 1-delta: 0.05
Estimador:  1.340586      Error:  0.001858          Tiempo(segundos):
2.420248
```

Código fuente:

```
import random
import math
import time
from scipy.stats import norm

random.seed(6)


n=int(input('Ingrese el número de replicaciones, n: '))
delta=float(input('Ingrese delta, tal que el nivel de confianza sea 1-delta:
'))

def k(x,y):
    centro=[0.5, 0.5]
    radio=0.4
    H=8
    distancia=math.sqrt((x-centro[0])**2+(y-centro[1])**2)
    altura=H-H/radio*distancia
    return altura

def generar_punto():
    r = math.sqrt(random.random())
    Z1 = random.gauss(0,1)
    Z2 = random.gauss(0,1)
    X1 = 0.5 + r * Z1 / math.sqrt(Z1**2 + Z2**2) * 0.4
    X2 = 0.5 + r * Z2 / math.sqrt(Z1**2 + Z2**2) * 0.4
    return [X1, X2]

def integracion_m_l_s(funcion, dF, n, nivel):
    S=0
    T=0
    for i in range(1,n+1):
        punto= generar_punto()
        if i>1:
            T=T+(1-1/i)*(funcion(punto[0],punto[1])/dF-S/(i-1))**2
        S=S+funcion(punto[0],punto[1])/dF
    integral= S/n
    var_funcion= T/(n-1)
    var_integral= var_funcion/n
    I1=integral-norm.ppf(nivel+(1-nivel)/2)*math.sqrt(var_integral)
    I2=integral+norm.ppf(nivel+(1-nivel)/2)*math.sqrt(var_integral)
    error=(I2-I1)/2
```

```
    return integral, error, var_funcion

inicio= time.time()
est, err, var_fi_n = integracion_m_l_s(k, 1/(math.pi * 0.4**2), n, 1-delta)
tiempo_ejecucion= time.time()-inicio
print(f'Estimador:{est:10f}\
      \tError:{err:10f}\
      \tTiempo(segundos):{tiempo_ejecucion:10f}')
```