Nombre: Juan Manuel Varela

Entrega 2: Métodos de Monte Carlo

Unidad 2 – Sesión 03 - Ejercicio 3.1

Descripción del problema:

Se desea estimar el volumen de una región R de $[0, 1]^6$ definida por todos los puntos de la hiperesfera de centro (0.45, 0.5, 0.6, 0.6, 0.5, 0.45) y radio 0.35, que además cumplan las restricciones siguientes: $3x_1 + 7x_4 \leq 5$; $x_3 + x_4 \leq 1$; $x_1 - x_2 - x_5 + x_6 \geq 0$.

• Parte a: implementar un programa que reciba como parámetro la cantidad de replicaciones n a realizar, y emplee Monte Carlo para calcular (e imprimir) la estimación del volumen de R, y la desviación estándar de este estimador. Incluir código para calcular el tiempo de cálculo empleado por el programa. Utilizar el programa con $n = 10^4$ y luego con $n = 10^6$ para estimar el volumen de R. Discutir si los dos valores obtenidos parecen consistentes.

• Parte b: como forma de validar el programa, eliminar las restricciones adicionales de desigualdad, y comparar el volumen calculado por Monte Carlo con $n = 10^6$ con el valor exacto el volumen de una hiperesfera de dimensión 6, $\frac{\pi^3 r^6}{6}$
Discutir también la relación de este valor con el obtenido en la parte a.

Solución a aplicar en Python:

Parte a:

1. Importar random #Esta biblioteca incluye funciones para generar números seudoaleatorios
2. Importar numpy #Esta biblioteca incluye funciones para calcular suma, raiz cuadrada
3. Importar time #Esta biblioteca incluye funciones que permiten calcular el tiempo de ejecución
4. Definir semilla
5. Definir función: random_point(): """Genera un punto seudoaleatorio de dimensión 6 dentro del hipercubo $[0,1]^6$"""
   5.1. Point= array vacío
   5.2. Para i entre 0 y 5:
      5.2.1. Generar un número seudoaleatorio entre 0 y 1 y agregarlo a Point
   5.3. Salida: Point
6. Definir función: monte_carlo(n): """Ejecuta la función 'random_point' n veces y calcula la estimación del volumen de la región y la varianza del estimador"""
   6.1. Suma= 0 #Inicialización
   6.2. Centro=(0.45, 0.5, 0.6, 0.6, 0.5, 0.45) #Definición del centro de la hiperesfera R
   6.3. For i entre 0 y n-1:
      6.3.1. Punto= random_point()
      6.3.2. Distancia = distancia a Centro
      6.3.3. If (Distancia ≤ 0.35) and (cumple el resto de condiciones):
         6.3.3.1. S=S+1 #Acumulo
   6.4. λ=S/n #Fracción de puntos que cae dentro de la región definida por las condiciones
   6.5. V= λ*(1- λ)/(n-1) #Cálculo de la varianza del estimador
   6.6. σ= raíz(V) #Cálculo de la desviación estándar del estimador
   6.7. Salida: λ, σ
7. Leer n

8. Inicio = tiempo de inicio
9. Volumen, Desv_Volumen = monte_carlo(n)
10. Tiempo_ejecución=Tiempo final-Inicio #Cálculo del tiempo de ejecución
11. Escribir “Volumen estimado: ”Volumen, “Desviación estándar: ”Desv_Volumen, “Tiempo de ejecución: “Tiempo_ejecución” s”

Parte b:

1. Importar random #Esta biblioteca incluye funciones para generar números seudoaleatorios
2. Importar numpy #Esta biblioteca incluye funciones para calcular suma, raiz cuadrada
3. Importar time #Esta biblioteca incluye funciones que permiten calcular el tiempo de ejecución
4. Definir semilla
5. Definir función: random_point(): """Genera un punto seudoaleatorio de dimensión 6 dentro del hipercubo $[0,1]^6$"""
   5.1. Point= array vacío
   5.2. Para i entre 0 y 5:
      5.2.1. Generar un número seudoaleatorio entre 0 y 1 y agregarlo a Point
   5.3. Salida: Point
6. Definir función: monte_carlo(n): """Ejecuta la función ‘random_point’ n veces y calcula la estimación del volumen de la región y la varianza del estimador"""
   6.1. Suma= 0 #Inicialización
   6.2. Centro=(0.45, 0.5, 0.6, 0.6, 0.5, 0.45) #Definición del centro de la hiperesfera *R*
   6.3. For i entre 0 y n-1:
      6.3.1. Punto= random_point()
      6.3.2. Distancia = distancia a Centro
      6.3.3. If (Distancia ≤ 0.35):
         6.3.3.1. S=S+1 #Acumulo
   6.4. λ=S/n #Fracción de puntos que cae dentro de la hiperesfera
   6.5. V= λ*(1- λ)/(n-1) #Cálculo de la varianza del estimador
   6.6. σ= raíz(V) #Cálculo de la desviación estándar del estimador
   6.7. Salida: λ, σ
7. Leer n
8. Inicio = tiempo de inicio
9. Volumen, Desv_Volumen = monte_carlo(n)
10. Tiempo_ejecución=Tiempo final-Inicio #Cálculo del tiempo de ejecución
11. Volumen_real= cálculo de volumen real con fórmula matemática
12. Escribir “Volumen estimado: ”Volumen, “Desviación estándar: ”Desv_Volumen, “Tiempo de ejecución: “Tiempo_ejecución” s”
13. Escribir “Volumen real: “Volumen_real

Resultados:
Plataforma de cómputo: PC de escritorio, procesador Intel I5-12400, 32 Gb RAM, Windows 10
Semilla: 10 para todas las ejecuciones

Parte a:

n	Volumen est.	Desviación estándar	t (s)
10^4	0,0002	0,00014141428428549925	0,044847965240478516
10^6	0,000295	0,00001717303904127833	4,1671364307403564

Si bien los valores son bastante diferentes (el volumen estimado para $n=10^6$ es un 47,5% mayor al estimado para $n=10^4$), se debe tener en cuenta que la desviación estándar del estimador aún es muy grande en el caso de $n=10^4$ (está en el mismo orden que el volumen estimado), por lo que se podría concluir que ese número de replicaciones no es suficiente para dar una estimación que se acerque a la realidad. Al aumentar el número de replicaciones a 10^6, la desviación estándar disminuye un orden, por lo que se podría suponer que este valor de la estimación del volumen se acerca mucho más al volumen real.

Parte b:

n	Volumen est.	Desviación estándar	t (s)
10^6	0,009533	0,00009717063009537065	4,710233449935913

Volumen real: 0,009499628763439042

Los valores de volumen obtenidos a partir de la simulación y de la fórmula matemática son muy próximos entre sí, por lo tanto, se concluye que el programa devuelve los resultados deseados.
Además, este valor es coherente respecto al obtenido en la parte a, ya que, al quitarle restricciones, el volumen obtenido debería ser igual o mayor.

Unidad 2 – Sesión 04 - Ejercicio 4.1

Descripción del problema:

1. Comparar y discutir la dependencia de los criterios de peor caso n_C, n_N, n_H frente a los parámetros ε y δ.
2. Calcular n_C, n_N, n_H para ε = 0.01, δ = 0.001, 0.01, 0.05.

Solución a aplicar en Python:

Parte 2:

1. Importar math #Esta biblioteca incluye funciones para calcular el entero inmediatamente mayor a un número real y logaritmo natural
2. Importar norm de scipy.stats #Incluye funciones para calcular valores de la distribución normal
3. Leer delta #Definición de intervalo de confianza nivel (1-delta)
4. Leer épsilon #Definición de tamaño de intervalo de confianza
5. Calcular n_C
6. Calcular n_N

7. Calcular n_H
8. Escribir "nC=" n_C, "nN=" n_N, "nH=" n_H

Resultados:

1. Analizando las fórmulas para el cálculo del tamaño de muestra necesario según los distintos métodos:

   Desigualdad de Chebychev:

   $$n_C(\epsilon, \delta) = \lceil 1/(4\delta\epsilon^2) \rceil$$

   Teorema Central del Límite:

   $$n_N(\epsilon, \delta) = \lceil \left(\Phi^{-1}(1-\delta/2)/2\epsilon\right)^2 \rceil$$

   Teorema de Hoeffding:

   $$n_H(\epsilon, \delta) = \lceil 2ln(2/\delta)/(4\epsilon^2) \rceil$$

   Se puede ver que en todos los casos *n* disminuye como $1/\varepsilon^2$, por lo que para un δ fijo, se cumplirá que, para cualquier ε escogido, $n_C > n_H > n_N$
   También se aprecia que n_C disminuye como 1/δ, n_N como $\phi^{-1}(1-\delta/2)$, mientras que n_H lo hace como ln(2/δ). Por esta razón, ocurre que para un ε fijo, dependiendo del valor de δ podrá darse que $n_C > n_H > n_N$ o $n_H > n_C > n_N$.
   Además, se puede concluir que si se quiere bajar el *n* necesario, es más favorable aumentar el tamaño del intervalo de confianza que disminuir el nivel de confianza ($1/\varepsilon^2$ se acerca a 0 más rápido que cualquiera de las otras funciones)

2.

ε	**δ**	**n_C**	**n_N**	**n_H**
0,01	0,001	2500000	27069	38005
0,01	0,01	250000	16588	26492
0,01	0,05	50000	9604	18445

   Para los parámetros utilizados, el tamaño de muestra necesario es menor cuando se calcula empleando el Teorema Central del Límite.

Anexos:

Log de ejecuciones Ej 3.1 parte a:

```
runcell(0, 'G:/…/Ej 3.1a.py')
Ingrese la cantidad de replicaciones a realizar, n:
10000
Volumen estimado: 0.0002 , Desviación estándar: 0.00014141428428549925 ,
Tiempo de ejecución: 0.044847965240478516 s

runcell(0, 'G:/…/Ej 3.1a.py')
Ingrese la cantidad de replicaciones a realizar, n:
1000000
Volumen estimado: 0.000295 , Desviación estándar: 1.7173039041278334e-05 ,
Tiempo de ejecución: 4.1671364307403564 s
```

Log de ejecuciones Ej 3.1 parte b:

```
runcell(0, 'G:/…/Ej 3.1b.py')
Ingrese la cantidad de replicaciones a realizar, n:
1000000
Volumen estimado: 0.009533 , Desviación estándar: 9.717063009537065e-05 ,
Tiempo de ejecución: 4.710233449935913 s
Volumen real: 0.009499628763439042
```

Log de ejecuciones Ej 4.1:

```
runcell(0, 'G:/…/Ej 4.1.py')
Ingrese delta:
0.001
Ingrese epsilon:
0.01
nC=2500000 , nN=27069 , nH=38005

runcell(0, 'G:/…/Ej 4.1.py')
Ingrese delta:
0.01
Ingrese epsilon:
0.01
nC=250000 , nN=16588 , nH=26492

runcell(0, 'G:/…/Ej 4.1.py')
Ingrese delta:
0.05
Ingrese epsilon:
0.01
nC=50000 , nN=9604 , nH=18445
```

Código fuente Ej 3.1 parte a:

```
import random #Esta biblioteca incluye funciones para generar números
seudoaleatorios
import numpy as np #Esta biblioteca incluye funciones para calcular suma, raiz
cuadrada
import time #Esta biblioteca incluye funciones que permiten calcular el tiempo
de ejecución

random.seed(10) #Inicialización de la secuencia de números seudoaleatorios
```

```
def random_point():
    """Genera un punto seudoaleatorio de dimensión 6"""
    point=[] #Creación de lista vacía
    #Se agregan las coordenadas del punto una a una
    for i in range(6):
        point.append(random.random())
    point_arr=np.array(point) #1-D array con las coordenadas del punto
    return point_arr

def monte_carlo(n):
    S=0 #Inicialización
    centro_R=np.array([0.45, 0.5, 0.6, 0.6, 0.5, 0.45]) #Definición del centro
de la hiperesfera
    for i in range(n):
        point=random_point()
        distance=np.sqrt(np.sum((point-centro_R)**2)) #Cálculo de la distancia
entre el punto generado y el centro
        #Se verifica que el punto esté dentro de la hiperesfera y que cumpla
las demás condiciones
        if distance<=0.35 and (3*point[0]+7*point[3])<=5 and
(point[2]+point[3])<=1 and (point[0]-point[1]-point[4]+point[5])>=0:
            S=S+1 #Acumulación de casos verdaderos
    lambda_hat=S/n #Fracción de puntos que están dentro de la región
    V_lambda_hat=(lambda_hat)*(1-lambda_hat)/(n-1)
    Desv_lambda_hat=np.sqrt(V_lambda_hat)
    return lambda_hat, Desv_lambda_hat

entrada=input('Ingrese la cantidad de replicaciones a realizar, n:\n')

inicio=time.time()
Volumen, Desv_Volumen = monte_carlo(int(entrada))
tiempo=time.time()-inicio #Cálculo del tiempo de ejecución

print('Volumen estimado: '+str(Volumen), ', Desviación estándar:
'+str(Desv_Volumen), ', Tiempo de ejecución: '+str(tiempo)+' s')
```

Código fuente Ej 3.1 parte b:

```
import random #Esta biblioteca incluye funciones para generar números
seudoaleatorios
import numpy as np #Esta biblioteca incluye funciones para calcular suma, raiz
cuadrada
import time #Esta biblioteca incluye funciones que permiten calcular el tiempo
de ejecución

random.seed(10) #Inicialización de la secuencia de números seudoaleatorios

def random_point():
    """Generates a random point of dimension 6"""
    point=[] #Creación de lista vacía
    #Se agregan las coordenadas del punto una a una
    for i in range(6):
        point.append(random.random())
    point_arr=np.array(point) #1-D array con las coordenadas del punto
    return point_arr

def monte_carlo(n):
    S=0 #Inicialización
    for i in range(n):
```

```
        centro_R=np.array([0.45, 0.5, 0.6, 0.6, 0.5, 0.45]) #Definición del
centro de la hiperesfera
        point=random_point()
        distance=np.sqrt(np.sum((point-centro_R)**2)) #Cálculo de la distancia
entre el punto generado y el centro
        if distance<=0.35: #Se verifica que el punto esté dentro de la
hiperesfera
            S=S+1 #Acumulación de casos verdaderos
    lambda_hat=S/n #Fracción de puntos que están dentro de la región
    V_lambda_hat=(lambda_hat)*(1-lambda_hat)/(n-1)
    Desv_lambda_hat=np.sqrt(V_lambda_hat)
    return lambda_hat, Desv_lambda_hat

entrada=input('Ingrese la cantidad de replicaciones a realizar, n:\n')

inicio=time.time()
Volumen, Desv_Volumen = monte_carlo(int(entrada))
tiempo=time.time()-inicio #Cálculo del tiempo de ejecución

Volumen_real= np.pi**3*0.35**6/6

print('Volumen estimado: '+str(Volumen), ', Desviación estándar:
'+str(Desv_Volumen), ', Tiempo de ejecución: '+str(tiempo)+' s')
print('Volumen real: '+str(Volumen_real))
```

Código fuente Ej 4.1:

```
import math #Esta biblioteca incluye funciones para calcular el entero
inmediatamente mayor a un número real y logaritmo natural
from scipy.stats import norm #Incluye funciones para calcular valores de la
distribución normal

delta=float(input('Ingrese delta:\n')) #Definición de intervalo de confianza
nivel (1-delta)
epsilon=float(input('Ingrese epsilon:\n')) #Definición de tamaño de intervalo
de confianza

nC=math.ceil(1/(4*delta*epsilon**2))
nN=math.ceil((norm.ppf(1-delta/2)/(2*epsilon))**2)
nH=math.ceil(2*math.log(2/delta)/(4*epsilon**2))

print('nC='+str(nC),', nN='+str(nN),', nH='+str(nH))
```