Métodos Montecarlo Fing 2022 - Entrega 3

Autor: Carlos M. Martinez, marzo-abril 2022.

Email: carlosm@fing.edu.uy (mailto:carlosm@fing.edu.uy), carlos@cagnazzo.uy (mailto:carlos@cagnazzo.uy)

Ejercicio 6.1

se idealiza una montaña como un cono inscrito en una region 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 × 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 volúmen 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.

In [1]:
```
import random
import math
import tabulate
import time
from IPython.core.display import HTML
random.seed()

import cm2c.fing.mmc.integral as mmci
import cm2c.fing.mmc.utils as mmcutils
reloj_ppal = mmcutils.timeit()
mmci.version()
```

Out[1]:
```
'Integracion MMC v0.1.2 - Carlos Martinez abril 2022'
```

In [2]:
```
# Validacion: integro f(x) = x**2 en (0,1)

import math

r = mmci.integracionMonteCarlo(lambda x: x[0]**2, 1, 10**5)

print("El resultado debe aproximarse a 1/3: ", r[0])
```

```
El resultado debe aproximarse a 1/3:  0.33360438815925963
```

Ahora, para resolver el problema del volumen de la montaña defino una función en R2 que me devuelva la altura de la montaña abstracta:

In [3]:
```
H = 8.0 # altura en km
r = 0.4 # radio de la base en km
n = 10**6 # cantidad de muestras para la parte (a)
delta = 0.05

import math

def Montana(x):
    """
    x es un vector de dos elementos
    devuelve la altura estimada
    """

    # calculo distancia al centro
    d = math.sqrt( (x[0]-0.5)**2 + (x[1]-0.5)**2 )

    if d > 0.4:
        return 0.0
    else:
        return H - (H/r)*d
## end def Montana

(estimZ, estimV, _, _) = mmci.integracionMonteCarlo(Montana, 2, n)

(icn0, icn1) = mmci.intConfianzaAproxNormal(estimZ, estimV, n, delta)

epsilon_est = estimZ-icn0

print(" ")
print("Volumen estimado por MMC {:.5f} km3".format(estimZ))
print("Varianza estimada : {:.5e}".format(estimV))
print("Intervalo de confianza para delta {} : ({:.5f}, {:.5f}) ".format(delta, ic
print("Error estimado: {:.5e}".format(epsilon_est))
```

```
Volumen estimado por MMC 1.34109 km3
Varianza estimada : 3.56573e-06
Intervalo de confianza para delta 0.05 : (1.33739, 1.34479)
Error estimado: 3.70103e-03
```

Parte b:

En base al valor estimado en la parte a, calcular el número de replicaciones necesario para obtener un error absoluto menor a 10^-3 (con nivel de confianza 0.95).

- Paso 1: Realizar un número n' de pruebas preliminares y estimar la varianza.

  En este caso ya tenemos una estimación para n' = 10^6

In [4]:
```
print("Volumen estimado por MMC {:.5f} km3".format(estimZ))
print("Varianza estimada : {:.5e}".format(estimV))
```

```
Volumen estimado por MMC 1.34109 km3
Varianza estimada : 3.56573e-06
```

- Paso 2

  Calcular el tamaño de la muestra requerido según la aproximación normal, el cual se define por:

  ```
  npuntoN(epsilon, delta) = norm.ppf(1-delta/2)**2 * estimV / (epsilon**2)
  ```

In [5]:
```
from scipy.stats import norm

def npuntoN(delta_, epsilon_, estimV_, n_):
    return (norm.ppf(1-delta_/2)**2)*estimV_*n_/(epsilon_**2)

nputoN_est = math.ceil ( npuntoN(0.05, 0.001, estimV, n) )

print(f'{estimV} , {n}')
print(f'{npuntoN_est:,}')
```

```
3.56573492794628e-06 , 1000000
13,697,624
```

- Paso 3

  Repetir las simulaciones con N > npuntoN_estimado, asegurando que las semillas son diferentes (para garantizar la independencia de los experimentos)

In [6]:
```
# tryN = [ 14*10**6, 16*10**6, 20*10**6, 30*10**6]
tryN = [ 14*10**4, 16*10**4, 20*10**4, 30*10**4]

table1 = [ ['N', 'Volumen est. MMC', 'Varianza est.', 'Int. Confianza', 'Error es

reloj = mmcutils.timeit()

for n in tryN:
    (estimZ, estimV, _, _) = mmci.integracionMonteCarloParalelo(Montana, 2, n, 8)
    (icn0, icn1) = mmci.intConfianzaAproxNormal(estimZ, estimV, n, delta)
    epsilon_est = estimZ-icn0
    table1.append([f'{n:,}', f'{estimZ:.5f}', f'{estimV:.5e}', f'({icn0:.5f}, {ic
```

In [7]:
```
tabulate.tabulate(table1, tablefmt='html')
```

Out[7]:

N	Volumen est. MMC	Varianza est.	Int. Confianza	Error est.	Tiempo ejecución
140,000	1.33430	2.54087e-05	(1.32442, 1.34418)	9.87960e-03	0.184
160,000	1.34069	2.23490e-05	(1.33143, 1.34996)	9.26568e-03	0.057
200,000	1.34869	1.79430e-05	(1.34039, 1.35700)	8.30224e-03	0.072
300,000	1.34627	1.19661e-05	(1.33949, 1.35305)	6.77990e-03	0.089

Conclusión La formula de estimación de cantidad de muestras nos da una referencia, en este caso unos 13.6x10^6 muestras. Probando con cantidades de muestras levemente mayores ya logramos bajar el error de 10^-3 que era lo pedido.

Ejercicio 6.2

Problema:

Se desea estimar la integral de la función `F5(X) = x1*x2^2*x3^3*x4^4*x5^5` sobre el hipercubo J^m de dimensión m = 5.

Parte a:

Revisar los códigos preparados para el ejercicio 6.1, elegir uno de ellos como punto de partida.

Sobre esa base, modificarlo para realizar cálculo por Monte Carlo de la integral planteada en el ejercicio 6.2. Realizar 10^6 replicaciones y estimar el valor de ζ. Calcular analíticamente el valor exacto de la integral.

In [8]:
```
# Definimos la función

def F5(x):
    """
    x es un vector en R^5
    """
    return x[0] * (x[1]**2) *  (x[2]**3) * (x[3]**4) * (x[4]**5)
# end def

# Calculo la integral con 10^6 replicas
n = 10**6

(estimZ, estimV, _, _) = mmci.integracionMonteCarlo(F5, 5, n)

HTML(f'<h4>estimZ: {estimZ:.7f} - estimV: {estimV:.5e}</h4>')
```

Out[8]:

estimZ: 0.0014024 - estimV: 9.67742e-11

Cálculo analítico de la integral de la función F5 en J^5:

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 a\, b^2\, c^3\, d^4\, e^5 \, da\, db\, dc\, dd\, de = \frac{1}{720} \approx 0.00138889$$

Computed by Wolfram|Alpha

Parte b:

En base al valor estimado en la parte a, calcular el número de replicaciones necesario para obtener un error menor a 10−4 (con nivel de confianza 0.95).

In [9]:
```
npuntoN_est = math.ceil ( nuntoN(0.05, 0.0001, estimV, n) )

print(f'Estimación de cantidad de muestras: {nuntoN_est:,}')
```

```
Estimación de cantidad de muestras: 37,176
```

Parte c:

Decimos que un intervalo de confianza cubre el valor exacto cuando este último pertenece al intervalo.

Realizar L = 500 experimentos con semillas diferentes, cada uno consistente en estimar por Monte Carlo con el nro. de replicaciones de la parte b el valor de la integral, así como intervalos de confianza de nivel 0.9, 0.95 y 0.99.

Para cada nivel de confianza, calcular el nivel de cobertura empírico (en que porcentaje de los 500 experimentos el intervalo de confianza calculado cubrió el valor exacto).

Discutir los resultados, comparando la cobertura empírica con la especificada.

In [10]:
```
table2 = [ ['L', 'n', '1-delta', '% Experimentos exitosos', 'Tiempo ejecución'] ]
L = 500
delta = 0.10
ZAnalitico = 1.0/720.0
npuntoN_est = math.ceil ( npuntoN(delta, 0.0001, estimV, n) )
reloj = mmcutils.timeit()

def Lexperimentos():
    cobertura = []
    for j in range(0,L):
        random.seed()
        (estimZ_, estimV_, _, _) = mmci.integracionMonteCarloParalelo(F5, 5, npur
        (icn0, icn1) = mmci.intConfianzaAproxNormal(estimZ_, estimV_, nuntoN_est
        epsilon = ZAnalitico-icn0
        if ZAnalitico>=icn0 and ZAnalitico<=icn1:
            cobertura.append(1)
        else:
            cobertura.append(0)
    # end for
    # La cantidad de unos en el array cobertura es la cantidad de experimentos do
    # en el intervalo de confianza
    exitos = cobertura.count(1)

    table2.append( [L, f'{npuntoN_est:,}', f'{1-delta:.2f}', f'{exitos/L*100:.2f}
# end def

Lexperimentos()
print(reloj.lap())
```

```
7.439999899361283e-05
```

In [11]:
```
delta = 0.05
npuntoN_est = math.ceil ( npuntoN(delta, 0.0001, estimV, n) )
Lexperimentos()
```

In [12]:
```
delta = 0.01
npuntoN_est = math.ceil ( npuntoN(delta, 0.0001, estimV, n) )
Lexperimentos()
```

In [13]:
```
tabulate.tabulate(table2, tablefmt='html')
```

Out[13]:

L	n	1-delta	% Experimentos exitosos	Tiempo ejecución
500	26,183	0.90	90.00%	7.983
500	37,176	0.95	93.80%	10.405
500	64,209	0.99	99.00%	16.359

Discusión de los resultados de cobertura

Me resulta interesante ver de que la cantidad de muestras que obtenemos de la formula basada en la normal partiendo de la estimación de la varianza obtenida para 10^6 muestras es significativamente más pequeña que 10^6 (del orden de 3.6 * 10^4).

Cuando realizamos los L=500 experimentos partiendo de estas cantidades de muestras calculadas de esta forma (bastante más pequeñas) obtenemos porcentajes de cobertura que parecen bastante buenos.

Al disminuir el tamaño del intervalo de confianza, naturalmente la cantidad de muestras crece. Es interesante ver también que el porcentaje de experimentos exitosos también crece.

Datos adicionales y referencias

Información acerca del software y hardware utilizados

Software:

- Python 3.8.10 corriendo en Windows WSL2 (Windows Subsystem for Linux)
- Jupyter Notebook

Librerias:

- scipy norm
- pathos multiprocessing (para parelilizar ejecuciones)

Hardware:

- PC Windows 11, con WSL2
- CPU Intel Core i5 10400F (6 cores)
- 16 GB de RAM

In [14]:
```
print(f"%% FIN - tiempo total de ejecución {reloj_ppal.lap():.3f}s")
```

```
%% FIN - tiempo total de ejecución 38.740s
```

Código de las funciones desarrolladas

Adjunto en el archivo *"integral.py.pdf"*.

```python
"""
Montecarlo para integrales.
(c) Carlos M. martinez, marzo-abril 2022
"""

import random
import math
import tabulate
import time
from scipy.stats import norm
import functools
from cm2c.fing.mmc.utils import sortearPuntoRN
from pathos.multiprocessing import ProcessPool as Pool

_VERSION = "Integracion MMC v0.1.2 - Carlos Martinez abril 2022"

def version():
    return _VERSION
# end def

def integracionMonteCarlo(Phi, dim, n):
    """
    Integracion por Montecarlo.
    Phi: funcion a integrar
    n: tamaño de la muestra (cantidad de iteraciones)
    dim: dimensionalidad del problema
    delta: intervalo de confianza

    Resultado: (estimacion valor integral, estimacion varianza)
    """
    S = 0
    T = 0
    for j in range(1, n+1):
        # sortear X({j} con distribución uniforme en R(n)
        Xj = sortearPuntoRN(dim)
        # print(Xj, Phi(Xj))
        if j>1:
            T = T + (1-1/j)*(Phi(Xj)-S/(j-1))**2
        S = S + Phi(Xj)
    # end for
    estimZ = S / n
    estimSigma2 = T / (n-1)
    estimVar = estimSigma2 / n

    return (estimZ, estimVar, S, T)
## end def

## intervalo de confianza aproximación normal
def intConfianzaAproxNormal(estimZ, estimV, n, delta):
    """
    Intervalo de confianza para la integración de Monte Carlo, según el criterio
    de la aproximación normal.

    estimZ : valor estimado de la integraal
    estimV : valor estimado de la varianza
    n : cantidad de iteraciones
    delta : amplitud del intervalo de confianza
```

```
    """

    D = norm.ppf(1-delta/2)*math.sqrt(estimV)

    I0 = estimZ - D

    I1 = estimZ + D

    return (I0, I1)
# end def


# Version paralelizada de Montecarlo
def integracionMonteCarloParalelo(Phi, dim, n, hilos):
    """
        version paralelizada del montecarlo
        N: numero de muestras
        Phi: funcion que implementa el volumen
        hilos: cantidad de hilos en el pool de tareas
    """

    args1 = []
    args2 = []
    args3 = []
    for x in range(0,hilos):
        args3.append( math.ceil(n/hilos) )
        args2.append(dim)
        args1.append(Phi)

    p = Pool(hilos)
    resultados = p.map(integracionMonteCarlo, args1, args2, args3 )
    #print(resultados)

    # unir los resultados para producir el resultado final
    Stotal = 0
    Ntotal = 0
    Ttotal = 0
    for i in range(0, hilos):
        Stotal = Stotal + resultados[i][2]
        Ttotal = Ttotal + resultados[i][3]
        Ntotal = Ntotal + math.ceil(n/hilos)
    #
    VolR = Stotal / Ntotal
    VarVorR = (Stotal/Ntotal)*(1-Stotal/Ntotal)/(Ntotal-1)

    estimZ = Stotal / Ntotal
    estimSigma2 = Ttotal / (Ntotal-1)
    estimVar = estimSigma2 / Ntotal

    return (estimZ, estimVar, Stotal, Ttotal)
# end def integral montecarlo paralelo

if __name__ == "__main__":
    print("Es una biblioteca, no es para correr directamente")
```