

# Métodos Montecarlo Fing 2022 - Entrega 3

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

**Email:** carlosm@fing.edu.uy, 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  $r = 0.4\text{km}$ , y la altura es  $H = 8\text{km}$ . 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  $\text{km}^3$ ) 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  $\zeta$  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
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)
```

El resultado debe aproximarse a 1/3: (0.3336498738021155, 8.888123888347356e-07)

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, icn0, icn1))
print("Error estimado: {:.5e}".format(epsilon_est))

```

```

Volumen estimado por MMC 1.33906 km3
Varianza estimada : 3.56224e-06
Intervalo de confianza para delta 0.05 : (1.33536, 1.34276)
Error estimado: 3.69922e-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 1.33906 km3
Varianza estimada : 3.56224e-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)

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

```

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

```
3.5622396974563113e-06 , 1000000
13,684,198
```

- 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 est.', 'T
reloj = mmcutils.timeit()

for n in tryN:
    (estimZ, estimV) = mmci.integracionMonteCarlo(Montana, 2, n)
    (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}, {icn1:.5f})', f'{epsilon_est:.5f}'])
```

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

```
Out[7]:
```

N	Volumen est. MMC	Varianza est.	Int. Confianza	Error est.	Tiempo ejecución
140,000	1.34031	2.55975e-05	(1.33040, 1.35023)	9.91623e-03	0.206
160,000	1.33312	2.21624e-05	(1.32389, 1.34235)	9.22692e-03	0.232
200,000	1.34076	1.78397e-05	(1.33248, 1.34904)	8.27831e-03	0.290
300,000	1.33861	1.18475e-05	(1.33186, 1.34536)	6.74623e-03	0.439

**Conclusión** La formula de estimación de cantidad de muestras nos da una referencia, en este caso unos  $13.6 \times 10^6$  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  $F5(X) = x_1 \cdot x_2^2 \cdot x_3^3 \cdot x_4^4 \cdot x_5^5$  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  $\zeta$ . 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: 0.0013785 - estimV: 9.04433e-11

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

 Cálculo analítico de la integral de F5

### 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 ( npuntoN(0.05, 0.0001, estimV, n) )

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

```

Estimación de cantidad de muestras: 34,744

### 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).

```

In [10]: table2 = [ ['L', 'n', '1-delta', '% Experimentos exitosos', 'Tiempo ejecucion'] ]
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):
        (estimZ_, estimV_) = mmci.integracionMonteCarlo(F5, 5, npuntoN_est)
        (icn0, icn1) = mmci.intConfianzaAproxNormal(estimZ_, estimV_, npuntoN_est, delta
        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 donde el
    # en el intervalo de confianza
    exitos = cobertura.count(1)

```

```
table2.append( [L, f'{npuntoN_est:,}', 1-delta, f'{exitos/L*100:.3f}', f'{reloj.lap(
# end def
```

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

```
0.000101000000145030208
```

```
In [11]: delta = 0.05
npuntoN_est = math.ceil ( npuntoN(0.05, 0.0001, 9.62463e-11, 10**6) )
Lexperimentos()
print(npuntoN_est)
```

```
36973
```

```
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 ejecucion
500	24,470	0.9	92.200	24.578
500	36,973	0.95	95.200	36.957
500	60,009	0.99	99.000	58.909

## Datos adicionales y referencias

### Información acerca del software y hardware utilizados

#### Software:

- Python 3.9
- Jupyter Notebook

#### Librerías:

- scipy norm
- pathos multiprocessing

#### Hardware:

- Macbook Pro 2021, M1 Pro de 8 cores
- MacOS Monterrey 12.3.1
- 16 GB de RAM

```
In [ ]:
```

## Código de las funciones desarrolladas

TBW\* antes de la entrega