Métodos Montecarlo Fing 2022 - Entrega 7

Autor: Carlos M. Martinez, junio 2022.

Email: carlosm@fing.edu.uy, carlos@cagnazzo.uy

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 v0.1.3 - Carlos Martinez abril-mayo 2022'
```

Ejercicio 14.1

Problema:

Se desea estimar la integral de la función

$$F5(X) = x_1 * x_2^2 * x3^3 * x_4^4 * x_5^5$$

sobre el hipercubo J^m de dimensión $m = 5$.

Valor analítico de la integral de F5 en J^5

El valor de esta integral en particular se puede calcular analíticamente de forma sencilla:

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

In [2]:
```
# defino variable con el valor analítico de la integral

VAnalitico = 1.0 / 720.0

HTML(f"<h4>El valor calculado analíticamente para la integral es: {VAnalitico:.10f}</
```

Out[2]: El valor calculado analíticamente para la integral es: 0.0013888889

(Del ejercicio 6.2, para tener luego para compararar)

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 [3]:
```
# 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, N, lambda x: mmcutils.sortear

(I0, I1) = mmci.intConfianzaAproxNormal(estimZ, estimV, N, 0.05)
anchoIC = I1-I0
errorReal = abs(estimZ-VAnalitico)
errorRealpct = errorReal/VAnalitico*100

HTML(f"<h4>Estimación de la integral simple: {estimZ:.7f}</h4>"+
     f"<h4>Estimación de la varianza : {estimV:.7e}</h4>"+
     f"<h4>Intervalo de confianza : ({I0:.7f}, {I1:.7f})</h4>"+
     f"<h4>Ancho del I. de confianza : ({anchoIC:.7e})</h4>"+
     f"<h4>Error real respecto al VAnalitico ({errorReal:.7e})</h4>"+
     f"<h4>Error real en pct : {errorRealpct:.3f}%</h4>")
```

Out[3]:

Estimación de la integral simple: 0.0013920

Estimación de la varianza : 9.3652532e-11

Intervalo de confianza : (0.0013731, 0.0014110)

Ancho del I. de confianza : (3.7934804e-05)

Error real respecto al VAnalitico (3.1583296e-06)

Error real en pct : 0.227%

Ejercicio 14.1 (grupal)

Dividiendo la dimensión x_5 definimos los siguientes estratos:

- [0, 0.72)
- [0.72, 0.83)
- [0.83, 0.90)
- [0.90, 0.95)
- [0.95, 1.00]

Se pide realizar dos experimentos:

1. Tomando $10^6/5$ iteraciones en cada estrato
2. Tomando una cantidad de iteraciones proporcional a la probablidad de ocurrencia en cada estrato

In [4]:
```
# testo el generador de puntos al azar con rangos
```

```
mmcutils.sortearPuntoRNRangos(2, [(0.2, 0.4), (0.4, 1.0)] )
```

Out[4]:
```
[0.21014085130565618, 0.6973889118021566]
```

Experimento 1

Realizamos los cáluclos tomando $10^6/5$ iteraciones en cada estrato.

In [5]:
```
### Experimento 1: muestreo estratificado pero con la misma cantidad de muestras por

N = 10**6
Rangos = [
 (0.0, 0.72),
 (0.72, 0.83),
 (0.83, 0.90),
 (0.90, 0.95),
 (0.95, 1.00)
]

n_i = [
    int(N/len(Rangos)),
    int(N/len(Rangos)),
    int(N/len(Rangos)),
    int(N/len(Rangos)),
    int(N/len(Rangos))
]

Z=0
V=0
for i, r in enumerate(Rangos):
    (estimZ, estimV, S, _) = mmci.integracionMonteCarloStieltjes(F5, n_i[i],
                                    lambda x: mmcutils.sortearPuntoRNRangos(5, [(0,1)
    p_i = r[1]-r[0]
    print(f"p_i={p_i}, S={S}, estimZ={estimZ}")
    Z = Z + (p_i/n_i[i])*S
    V = V + (p_i**2)*estimV
# end for

# Cálculo del intervalo de confianza al 95% segun aproximación normal

(I0, I1) = mmci.intConfianzaAproxNormal(Z, V, N, 0.05)
anchoIC = I1-I0
errorReal = abs(Z-VAnalitico)
errorRealpct = errorReal/VAnalitico*100


HTML(f"<h4>Estimación de la integral estratificada: {Z:.7f}</h4>"+
     f"<h4>Estimación de la varianza : {V:.7e}</h4>"+
     f"<h4>Intervalo de confianza ({I0:.7f}, {I1:.7f})</h4>"+
     f"<h4>Ancho del I. de confianza ({anchoIC:.7e})</h4>"+
     f"<h4>Error real respecto al VAnalitico ({errorReal:.7e})</h4>"+
     f"<h4>Error real en pct : {errorRealpct:.3f}%</h4>")
```

```
p_i=0.72, S=53.77402121110494, estimZ=0.0002688701060555247
p_i=0.10999999999999999, S=479.379421202924, estimZ=0.00239689710601462
p_i=0.07000000000000006, S=809.9072849763526, estimZ=0.004049536424881763
p_i=0.04999999999999993, S=1127.7331015123066, estimZ=0.005638665507561533
p_i=0.05000000000000044, S=1475.5162184704104, estimZ=0.0073775810923520525
```

Out[5]: Estimación de la integral estratificada: 0.0013915

Estimación de la varianza : 3.5162745e-11

Intervalo de confianza (0.0013799, 0.0014031)

Ancho del I. de confianza (2.3244461e-05)

Error real respecto al VAnalitico (2.6361489e-06)

Error real en pct : 0.190%

¿Cuántas muestras harían falta en un Montecarlo sin estratificar para lograr esta varianza?

In [6]:
```
Nse1 = mmcutils.npuntoN(0.05, 0.00001, estimV, N)
Nse2 = mmcutils.npuntoN(0.05, 0.00001, V, N)

print(f"Muestras necesarias en MC sin estratificar {Nse1:,d}")
print(f"Muestras necesarias en MC estratificado {Nse2:,d}")
```

```
Muestras necesarias en MC sin estratificar 150,011,643
Muestras necesarias en MC estratificado 1,350,763
```

Experimento 2

Realizamos los cálculos tomando una cantidad de iteraciones proporcional a la probablidad de acumulada en cada estrato.

In [7]:
```
## Experimento 2 , tomamos ni proporcional a la probabilidad acumulada en cada estrat

N = 10**6
Rangos = [
 (0.0, 0.72),
 (0.72, 0.83),
 (0.83, 0.90),
 (0.90, 0.95),
 (0.95, 1.00)
]

n_i = []

for r in Rangos:
    nit = int (N * (r[1]-r[0]) )
    n_i.append( nit )

n_i
```

Out[7]: [720000, 109999, 70000, 49999, 50000]

In [8]:
```
## Realizamos calculos con estas muestras por estrato:

Z=0
V=0
for i, r in enumerate(Rangos):
    (estimZ, estimV, S, _) = mmci.integracionMonteCarloStieltjes(F5, n_i[i],
                                        lambda x: mmcutils.sortearPuntoRNRangos(5, [(0,1)
    p_i = r[1]-r[0]
    print(f"p_i={p_i}, n_i={n_i[i]}, S={S}, estimZ={estimZ}")
    Z = Z + (p_i/n_i[i])*S
    V = V + (p_i**2)*estimV
# end for
```

```
# Cálculo del intervalo de confianza al 95% segun aproximación normal

(I0, I1) = mmci.intConfianzaAproxNormal(Z, V, N, 0.05)
anchoIC = I1-I0
errorReal = abs(Z-VAnalitico)
errorRealpct = errorReal/VAnalitico*100

HTML(f"<h4>Estimación de la integral estratificada: {Z:.7f}</h4>"+
     f"<h4>Estimación de la varianza : {V:.7e}</h4>"+
     f"<h4>Intervalo de confianza ({I0:.7f}, {I1:.7f})</h4>"+
     f"<h4>Ancho del I. de confianza ({anchoIC:.7e})</h4>"+
     f"<h4>Error real en pct : {errorRealpct:.3f}%</h4>")
```

```
p_i=0.72, n_i=720000, S=195.3118255047012, estimZ=0.000271266424312085
p_i=0.10999999999999999, n_i=109999, S=259.52760897652985, estimZ=0.00235936334854434
9
p_i=0.07000000000000006, n_i=70000, S=284.3025662756001, estimZ=0.0040614652325085725
p_i=0.04999999999999993, n_i=49999, S=278.30263733032643, estimZ=0.005566164069887926
p_i=0.050000000000000044, n_i=50000, S=367.6737291753708, estimZ=0.007353474583507416
```

Out[8]:

Estimación de la integral estratificada: 0.0013851

Estimación de la varianza : 8.8667928e-11

Intervalo de confianza (0.0013667, 0.0014036)

Ancho del I. de confianza (3.6911472e-05)

Error real en pct : 0.271%

Conclusiones

- Ambos modos de asignar muestras por estrato muestran mejoras en la varianza respecto al estimador no-estratificado, cumpliendo con el Teorema 2 de las diapositivas.
- La asignación de muestras equitativa, es decir asignar la misma cantidad de muestras a cada estrato a pesar de los estratos ser de diferente volumen, es significativamente superior tanto al Montecarlo sin estratificar como a la asignacion de muestras proporcional.
- La asignación de muestras proporcional a la probabilidad de cada estrato muestra una mejora pequeña respecto al Montecarlo sin estratificar, tal como explica el Teorema 2

El Teorema 2 muestra una forma de asignar las muestras para mejorar la varianza **pero** eso no quiere decir que esa forma sea la mejor.

Datos adicionales y referencias

Información acerca del software y hardware utilizados

Software:

- Python 3.8.9 corriendo en macOS 12.4 (Darwin 21.5.0 XNU/ARM64)
- Jupyter Notebook

Librerias:

- scipy norm

- pathos multiprocessing (para paralelizar ejecuciones)

Hardware:

- Macbook Pro 13 (late 2020, macOS 12.4)
- CPU Apple M1 (8 cores)
- 8 GB de RAM

In [9]:
```
print(f"%% FIN - tiempo total de ejecución {reloj_ppal.lap():.3f}s")
```

```
%% FIN - tiempo total de ejecución 5.765s
```

Código de las funciones desarrolladas

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