

# Simulación de Montecarlo

Investigación Operativa, Universidad de San Andrés

Si encuentran algún error en el documento o hay alguna duda, mandenm  un mail a rodriguezf@udesa.edu.ar y lo revisamos.

## 1. ¿Qué es?

La simulaci n de Montecarlo es una t cnica que usa n meros aleatorios para resolver problemas. B asicamente, en lugar de resolver algo de forma anal tica (que a veces es imposible), generamos miles de escenarios aleatorios y vemos qu  pasa en promedio. El proceso es simple:

- Identificamos qu  variables son aleatorias en nuestro problema
- Generamos miles de valores aleatorios para cada variable
- Ejecutamos nuestro modelo para cada combinaci n de valores
- Analizamos los resultados estad sticamente

Por ejemplo, si queremos saber cu nto inventario pedir pero la demanda es incierta, simulamos 10000 escenarios con demandas aleatorias y vemos cu l es la cantidad que nos da mejor resultado en promedio. La clave est  en que cada escenario es una “realizaci n” de lo que podr a pasar. Si simulamos 10000 veces, tenemos 10000 versiones diferentes de la realidad. Al final, promediamos todos los resultados y obtenemos una estimaci n de lo que esperamos que pase.

¿Por qu  funciona? Porque si generamos suficientes escenarios aleatorios, la ley de los grandes n meros nos dice que el promedio de nuestros resultados se va a acercar al valor esperado real del problema.

## 2. Distribuciones que usamos

Las m s comunes son:

- **Uniforme:**  $U(0, 1)$  - Genera valores equiprobables entre 0 y 1.

- **Normal:**  $N(\mu, \sigma^2)$  - Para cosas como demanda, tiempos, etc. Es la más común porque muchas variables en la vida real siguen esta distribución (o se aproximan).
- **Exponencial:**  $Exp(\lambda)$  - Para tiempos entre llegadas. Si los eventos llegan de forma aleatoria e independiente, los tiempos entre llegadas son exponenciales.
- **Poisson:**  $Pois(\lambda)$  - Para contar eventos (clientes que llegan). Si los tiempos entre llegadas son exponenciales, entonces el número de llegadas en un período fijo es Poisson.
- **Binomial:**  $Bin(n, p)$  - Para éxitos/fracasos. Cada ensayo es independiente y tiene la misma probabilidad de éxito.
- **Binomial Negativa:**  $NB(r, p)$  - Para contar fallas antes de obtener  $r$  éxitos. Útil para modelar tiempos hasta el fallo o número de intentos hasta el éxito.

¿Cómo elegimos qué distribución usar? Depende del problema:

- Si no sabemos nada, empezamos con normal
- Si es tiempo entre eventos, exponencial
- Si es conteo de eventos, Poisson
- Si es sí/no, binomial
- Si es número de fallas antes del éxito, binomial negativa

### 3. Ejemplo: ¿Cuánto inventario pedir?

Tenemos que decidir cuántas unidades pedir cada mes. La demanda es incierta: en promedio se venden 100 unidades pero puede variar (desviación estándar de 20). Los costos son:

- Comprar: \$50 por unidad
- Guardar inventario: \$5 por unidad por mes

- Quedarse sin stock: \$100 por unidad que no pudimos vender

El problema es que si pedimos poco, nos quedamos sin stock y perdemos ventas. Si pedimos mucho, nos sobra inventario y pagamos costos de almacenamiento. Necesitamos encontrar el punto óptimo.

La idea es probar diferentes cantidades (60, 65, 70, ..., 140 unidades) y para cada una simular 10000 meses con demandas aleatorias. Calculamos el costo total de cada escenario y promediamos. La cantidad que nos dé el menor costo promedio es la óptima.

### Resolución en Python:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def simulacion_inventario(n_simulaciones=10000):
5     # Parámetros del problema
6     media_demanda = 100
7     desvio_demanda = 20
8     costo_compra = 50
9     costo_almacenamiento = 5
10    costo_stockout = 100
11
12    # Rango de cantidades a evaluar
13    cantidades = range(60, 141, 5)
14    costos_promedio = []
15
16    for Q in cantidades:
17        costos_totales = []
18
19        for _ in range(n_simulaciones):
20            # Generar demanda aleatoria
21            demanda = max(0, np.random.normal(media_demanda,
desvio_demanda))
22
23            # Calcular costos
24            costo_total = Q * costo_compra
25
26            if demanda <= Q:
27                # Hay inventario sobrante
28                inventario_sobrante = Q - demanda
29                costo_total += inventario_sobrante *
costo_almacenamiento
30            else:

```

```

31     # Hay stockout
32     stockout = demanda - Q
33     costo_total += stockout * costo_stockout
34
35     costos_totales.append(costo_total)
36
37     costos_promedio.append(np.mean(costos_totales))
38
39     return cantidades, costos_promedio
40
41 # Ejecutar simulacion
42 cantidades, costos = simulacion_inventario()
43
44 # Encontrar cantidad optima
45 indice_optimo = np.argmin(costos)
46 cantidad_optima = cantidades[indice_optimo]
47 costo_optimo = costos[indice_optimo]
48
49 print(f"Cantidad optima: {cantidad_optima} unidades")
50 print(f"Costo promedio optimo: ${costo_optimo:.2f}")
51
52 # Graficar resultados
53 plt.plot(cantidades, costos, 'b-', linewidth=2)
54 plt.axvline(cantidad_optima, color='r', linestyle='--',
55             label=f'Optimo: {cantidad_optima} unidades')
56 plt.xlabel('Cantidad a pedir (Q)')
57 plt.ylabel('Costo promedio total')
58 plt.title('Optimizacion de Inventario - Simulacion Montecarlo')
59 plt.legend()
60 plt.grid(True, alpha=0.3)
61 plt.show()

```

## 4. Ejemplo: ¿Vale la pena el proyecto?

Queremos evaluar un proyecto de inversión pero hay mucha incertidumbre:

- Inversión inicial: puede ser \$100000 pero puede variar  $\pm \$10000$
- Flujo de caja anual: esperamos \$30000 pero puede variar  $\pm \$5000$
- Tasa de descuento: entre 8% y 12%
- Duración del proyecto: entre 5 y 10 años (más probable 8 años)

El problema es que no sabemos exactamente cuánto va a costar el proyecto, cuánto vamos a ganar cada año, ni cuánto va a durar. Si calculamos el VPN con valores fijos, nos perdemos toda la incertidumbre.

La simulación nos permite generar 10000 escenarios diferentes, cada uno con valores aleatorios para todas las variables. Calculamos el VPN de cada escenario y vemos la distribución de resultados. Así sabemos no solo el VPN esperado, sino también qué tan riesgoso es el proyecto.

### Resolución en Python:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.stats import triang
4
5 def simulacion_proyecto(n_simulaciones=10000):
6     np.random.seed(42)
7
8     # Parametros de las distribuciones
9     inv_media, inv_std = 100000, 10000
10    flujo_media, flujo_std = 30000, 5000
11    tasa_min, tasa_max = 0.08, 0.12
12
13    # Parametros para distribucion triangular (min, moda, max)
14    vida_min, vida_moda, vida_max = 5, 8, 10
15
16    vpn_resultados = []
17
18    for _ in range(n_simulaciones):
19        # Generar valores aleatorios
20        inversion_inicial = np.random.normal(inv_media,
21                                                inv_std)
21        flujo_anual = np.random.normal(flujo_media, flujo_std)
22
22        tasa_descuento = np.random.uniform(tasa_min, tasa_max)
23
23        vida_util = triang.rvs(c=(vida_moda-vida_min)/(
24            vida_max-vida_min),
24                                loc=vida_min, scale=vida_max-
25                                vida_min)
25
26        # Calcular VPN
27        vpn = -inversion_inicial
```

```

28         for anio in range(1, int(vida_util) + 1):
29             vpn += flujo_anual / ((1 + tasa_descuento) **
30             anio)
31
32             vpn_resultados.append(vpn)
33
34     return np.array(vpn_resultados)
35
36 # Ejecutar simulacion
37 vpn_simulaciones = simulacion_proyecto()
38
39 # Calcular estadisticas
40 media_vpn = np.mean(vpn_simulaciones)
41 std_vpn = np.std(vpn_simulaciones)
42 percentil_5 = np.percentile(vpn_simulaciones, 5)
43 percentil_95 = np.percentile(vpn_simulaciones, 95)
44 prob_rentable = np.mean(vpn_simulaciones > 0)
45
46 print(f"VPN promedio: ${media_vpn:.2f}")
47 print(f"Desviacion estandar: ${std_vpn:.2f}")
48 print(f"Percentil 5%: ${percentil_5:.2f}")
49 print(f"Percentil 95%: ${percentil_95:.2f}")
50 print(f"Probabilidad de ser rentable: {prob_rentable:.2%}")
51
52 # Graficar distribucion del VPN
53 plt.figure(figsize=(10, 6))
54 plt.hist(vpn_simulaciones, bins=50, alpha=0.7, color='skyblue',
55          edgecolor='black')
56 plt.axvline(media_vpn, color='red', linestyle='--', linewidth=2,
57             label=f'Media: ${media_vpn:.0f}')
58 plt.axvline(0, color='green', linestyle='-', linewidth=2,
59             label='VPN = 0')
60 plt.xlabel('Valor Presente Neto ($)')
61 plt.ylabel('Frecuencia')
62 plt.title('Distribucion del VPN - Simulacion Montecarlo')
63 plt.legend()
64 plt.grid(True, alpha=0.3)
65 plt.show()

```

## 5. ¿Cómo sabemos que está bien?

Hay que verificar que los resultados se estabilicen. Si simulamos muy pocas veces, el resultado puede ser muy variable. Si simulamos muchas veces, el

resultado se estabiliza.

La idea es hacer una “media móvil”: calculamos el promedio de los primeros 1000 resultados, luego de los primeros 2000, 3000, etc. Si el promedio se estabiliza (no cambia mucho), entonces sabemos que tenemos suficientes simulaciones.

```
1 def verificar_convergencia(resultados, ventana=1000):
2     medias_moviles = []
3     for i in range(ventana, len(resultados)):
4         media_móvil = np.mean(resultados[i-ventana:i])
5         medias_moviles.append(media_móvil)
6
7     plt.plot(range(ventana, len(resultados)), medias_moviles)
8     plt.xlabel('Número de iteraciones')
9     plt.ylabel('Media móvil')
10    plt.title('Convergencia de la Simulación')
11    plt.grid(True, alpha=0.3)
12    plt.show()
13
14    return medias_moviles
```

## 6. Código básico

```
1 # Generar numeros aleatorios
2 import numpy as np
3
4 # Uniforme - valores entre 0 y 1
5 uniforme = np.random.uniform(0, 1, 1000)
6
7 # Normal - media 100, desviacion 15
8 normal = np.random.normal(100, 15, 1000)
9
10 # Exponencial - tasa 2
11 exponencial = np.random.exponential(2, 1000)
12
13 # Poisson - tasa 5
14 poisson = np.random.poisson(5, 1000)
15
16 # Binomial - 10 ensayos, probabilidad 0.3
17 binomial = np.random.binomial(10, 0.3, 1000)
18
```

```

19 # Binomial Negativa - 5 exitos requeridos, probabilidad 0.7
20 binomial_negativa = np.random.negative_binomial(5, 0.7, 1000)
21
22 # Ejemplo simple: calcular el area de un circulo
23 def simulacion_montecarlo(n_iteraciones=10000):
24     resultados = []
25
26     for _ in range(n_iteraciones):
27         # Generar punto aleatorio en cuadrado [-1,1] x [-1,1]
28         x = np.random.uniform(-1, 1)
29         y = np.random.uniform(-1, 1)
30
31         # Verificar si esta dentro del circulo unitario
32         if x**2 + y**2 <= 1:
33             resultados.append(1)    # Dentro del circulo
34         else:
35             resultados.append(0)    # Fuera del circulo
36
37     return np.array(resultados)
38
39 # Ejecutar simulacion
40 resultados = simulacion_montecarlo()
41 # El area del circulo es aproximadamente 4 * (promedio de
42 # resultados)
43 area_estimada = 4 * np.mean(resultados)
44 print(f"Area estimada del circulo: {area_estimada:.4f}")
45 print(f"Area real: {np.pi:.4f}")

```