Simulación de Montecarlo

Investigación Operativa

Universidad de SanAndrés

¿Qué es la simulación de Montecarlo?

Técnica que usa números aleatorios para resolver problemas

En lugar de resolver analíticamente:

- Generamos miles de escenarios aleatorios
- Ejecutamos nuestro modelo para cada escenario
- Analizamos los resultados estadísticamente

¿Por qué funciona? La ley de los grandes números nos dice que el promedio se acerca al valor esperado real.

Proceso de simulación

Pasos:

1. Identificar variables aleatorias y parámetros del modelo
2. Generar valores aleatorios para cada variable (muchas repeticiones)
3. Ejecutar el modelo para cada escenario simulado
4. Analizar los resultados (promedios, percentiles, intervalos)

Distribuciones que usamos

- **Uniforme** $U(0,1)$: valores equiprobables entre 0 y 1
- **Normal** $N(\mu, \sigma^2)$: demanda, tiempos; muy común por aproximación
- **Exponencial** $Exp(\lambda)$: tiempos entre llegadas aleatorias independientes
- **Poisson** $Pois(\lambda)$: conteo de eventos en un período fijo
- **Binomial** $Bin(n, p)$: sí/no con ensayos independientes
- **Binomial Negativa** $NB(r, p)$: número de fallas antes de r éxitos.

Análisis de sensibilidad

¿Qué es? Una vez que encontramos la solución óptima de nuestro problema, es importante entender qué tan sensible es esta solución a cambios en los parámetros del modelo. El análisis de sensibilidad consiste en variar sistemáticamente uno o más parámetros del modelo y observar cómo cambia la solución óptima.

Esto nos permite:

- Identificar qué parámetros tienen mayor impacto en la solución
- Evaluar la robustez de nuestra decisión
- Preparar planes de contingencia para diferentes escenarios

¿Cómo sabemos que está bien?

Problema: 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.

Solución: 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.

Código: Verificación de convergencia

```python
def verificar_convergencia(resultados, ventana=1000):
    medias_moviles = []
    for i in range(ventana, len(resultados)):
        media_movil = np.mean(resultados[i-ventana:i])
        medias_moviles.append(media_movil)

    plt.plot(range(ventana, len(resultados)), medias_moviles)
    plt.xlabel('Numero de iteraciones')
    plt.ylabel('Media movil')
    plt.title('Convergencia de la Simulacion')
    plt.grid(True, alpha=0.3)
    plt.show()

    return medias_moviles
```

Problema 1: Gestión de inventario con demanda incierta

Enunciado

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 no servida

Si pedimos poco, nos quedamos sin stock y perdemos ventas. Si pedimos mucho, nos sobra inventario y pagamos costos de almacenamiento. Queremos encontrar la cantidad que minimiza el costo esperado.

Problema 1: Planteo del problema

Planteo del problema

- **Variable de decisión:** Q = cantidad a pedir
- **Variable aleatoria:** $D \sim N(100, 20^2)$, truncada en 0 para evitar demandas negativas
- **Parámetros:** $c_c = 50$ (compra), $c_h = 5$ (holding mensual), $c_s = 100$ (stockout)

Función objetivo Para una realización de demanda D y una decisión Q:

$$C(Q, D) = c_c\, Q + c_h\, \text{máx}(Q - D, 0) + c_s\, \text{máx}(D - Q, 0)$$

Objetivo: Encontrar Q^* que minimiza $E[C(Q, D)]$ mediante simulación.

Código: Problema 1 - Simulación base (1/2)

```python
import numpy as np
import matplotlib.pyplot as plt

def simulacion_inventario(n_simulaciones=10000):
    # Parametros del problema
    media_demanda = 100
    desvio_demanda = 20
    costo_compra = 50
    costo_almacenamiento = 5
    costo_stockout = 100

    # Rango de cantidades a evaluar
    cantidades = range(60, 141, 5)
    costos_promedio = []

    for Q in cantidades:
        costos_totales = []
        for _ in range(n_simulaciones):
            # Generar demanda aleatoria
            demanda = max(0, np.random.normal(media_demanda, desvio_demanda))

            # Calcular costos
            costo_total = Q * costo_compra

            if demanda <= Q:
                # Hay inventario sobrante
                inventario_sobrante = Q - demanda
                costo_total += inventario_sobrante * costo_almacenamiento
```

Código: Problema 1 - Simulación base (2/2)

```
            else:
                # Hay stockout
                stockout = demanda - Q
                costo_total += stockout * costo_stockout

            costos_totales.append(costo_total)
        costos_promedio.append(np.mean(costos_totales))

    return cantidades, costos_promedio

# Ejecutar simulacion
cantidades, costos = simulacion_inventario()

# Encontrar cantidad optima
indice_optimo = np.argmin(costos)
cantidad_optima = list(cantidades)[indice_optimo]
costo_optimo = costos[indice_optimo]

print(f"Cantidad optima: {cantidad_optima} unidades")
print(f"Costo promedio optimo: ${costo_optimo:.2f}")
```

Código: Problema 1 - Sensibilidad vs. demanda media (1/2)

```python
import numpy as np

sim = 10000
desvio_demanda = 20
costo_compra = 50
costo_almacenamiento = 5
costo_stockout = 100

# Rango de demandas medias a evaluar
demandas_medias = np.arange(80, 121, 5)
Q_vals = np.arange(60, 141, 5)
Q_opts = []
C_mins = []

for media_demanda in demandas_medias:
    costos_promedio = []

    for Q in Q_vals:
        costos_totales = []

        for _ in range(sim):
            # Generar demanda aleatoria
            demanda = max(0, np.random.normal(media_demanda, desvio_demanda))

            # Calcular costos
            costo_total = Q * costo_compra
```

Código: Problema 1 - Sensibilidad vs. demanda media (2/2)

```
21              if demanda <= Q:
22                  inventario_sobrante = Q - demanda
23                  costo_total += inventario_sobrante * costo_almacenamiento
24              else:
25                  stockout = demanda - Q
26                  costo_total += stockout * costo_stockout
27
28              costos_totales.append(costo_total)
29
30          costos_promedio.append(np.mean(costos_totales))
31
32      # Encontrar Q optimo para esta demanda media
33      idx_optimo = np.argmin(costos_promedio)
34      Q_opt = Q_vals[idx_optimo]
35      C_min = costos_promedio[idx_optimo]
36
37      Q_opts.append(Q_opt)
38      C_mins.append(C_min)
```

Problema 2: Optimización de reparto

Enunciado

Una empresa de logística busca optimizar la cantidad de repartidores para su servicio de entrega diario. La empresa estima una demanda promedio de 80 pedidos por día. Cada repartidor puede completar 12 entregas diarias, con un costo de contratación de \$120 por día. Adicionalmente, cada pedido que no se logre satisfacer representa un costo de penalización de \$25.

Problema 2: Planteo del problema

Planteo del problema

- **Variable de decisión:** M = número de repartidores a contratar
- **Variable aleatoria:** $D \sim \text{Poisson}(\lambda = 80)$ = demanda diaria
- **Parámetros:** $c = 12$ (capacidad por repartidor), $c_m = 120$ (costo por repartidor), $c_p = 25$ (costo de penalización)

Función objetivo Para un día dado, con M repartidores y una demanda D, el costo total es:

$$C(M, D) = \underbrace{c_m \cdot M}_{\text{Costo Fijo}} + \underbrace{c_p \cdot \text{máx}(D - c \cdot M, 0)}_{\text{Costo de Desabastecimiento}}$$

Objetivo: Encontrar M^* que minimice $E[C(M, D)]$ mediante simulación.

Código: Problema 2 - Simulación base

```python
import numpy as np

# parametros del problema
lam = 80
cap_por_repartidor = 12
costo_repartidor = 120
costo_penal = 25
sim = 10000
np.random.seed(42)

M_vals = np.arange(0, 16) # rango de M
costos_prom = []

for M in M_vals:
    # 1) simular demanda
    demanda = np.random.poisson(lam=lam, size=sim)
    # 2) calcular demanda insatisfecha
    capacidad_total = cap_por_repartidor * M
    pedidos_no_servidos = np.maximum(demanda - capacidad_total, 0)
    # 3) costo total
    costo_operativo = costo_repartidor * M
    costo_penalizacion = costo_penal * pedidos_no_servidos
    costo_total = costo_operativo + costo_penalizacion
    costos_prom.append(costo_total.mean())

# elegir M optimo
idx_optimo = np.argmin(costos_prom)
M_opt, C_min = M_vals[idx_optimo], costos_prom[idx_optimo]
print(M_opt, C_min)
```

Código: Problema 2 - Sensibilidad vs. λ

```python
import numpy as np

sim = 10000
cap_por_repartidor = 12
costo_repartidor = 120
costo_penal = 25

lambdas = np.arange(60, 121, 5)
M_vals = np.arange(0, 20)
M_opts = []
C_mins = []

for lam in lambdas:
    costos_prom = []
    for M in M_vals:
        demanda = np.random.poisson(lam=lam, size=sim)
        capacidad_total = cap_por_repartidor * M
        pedidos_no_servidos = np.maximum(demanda - capacidad_total, 0)
        costo_operativo = costo_repartidor * M
        costo_penalizacion_total = costo_penal * pedidos_no_servidos
        costo_total_promedio = (costo_operativo +
            costo_penalizacion_total).mean()
        costos_prom.append(costo_total_promedio)
    idx_optimo = np.argmin(costos_prom)
    M_opt, C_min = M_vals[idx_optimo], costos_prom[idx_optimo]
    M_opts.append(M_opt)
    C_mins.append(C_min)
```

Problema 3: Optimización de mantenimiento

Enunciado

Una fábrica busca optimizar el programa de mantenimiento para una de sus máquinas críticas. Esta máquina sufre una falla catastrófica después de acumular exactamente 5 fallos menores, cuya probabilidad es del 20 %. Para evitar las paradas costosas asociadas a esta falla, se realizan inspecciones preventivas a intervalos fijos de T días. Los costos asociados son:

- Costo Falla Catastrófica (c_f): \$10,000 (si falla entre inspecciones)
- Costo Inspección (c_i): \$400 (por cada visita)
- Costo Mantenimiento Preventivo (c_p): \$2,000 (adicional si la inspección coincide con el día de la falla)

Problema 3: Planteo del problema

Planteo del problema

- **Variable de decisión:** T = intervalo entre inspecciones (días)
- **Variable aleatoria:** D = duración del ciclo hasta la falla catastrófica
- **Modelo:** $D = D_{fracasos} + r$ donde $D_{fracasos} \sim \text{NegBin}(r = 5, p = 0{,}20)$
- **Parámetros:** $c_f = 10000$, $c_i = 400$, $c_p = 2000$

Función objetivo Para un ciclo de duración D y un intervalo T:

$$C(T, D) = \underbrace{C_{\text{mantenimiento}}(T, D)}_{\text{Preventivo o Catastrófico}} + \underbrace{C_{\text{inspección}}(T, D)}_{\text{Visitas}}$$

Objetivo: Minimizar el Costo Diario Esperado:
$E[C_{\text{diario}}(T)] = \frac{E[C(T,D)]}{E[D]}$

Código: Problema 3 - Simulación base (1/2)

```python
import numpy as np

r, p = 5, 0.20
costo_falla_catastrofica = 10000
costo_preventivo = 2000
costo_inspeccion = 400
sim = 20000
np.random.seed(42)

T_vals = np.arange(1, 15)
costos_por_dia = []

for T in T_vals:
    # 1) simular D
    D_fracasos = np.random.negative_binomial(r, p, size=sim)
    D = D_fracasos + r
    # 2) costos de mantenimiento
    mantenimiento_catastrofico = (D % T != 0) * costo_falla_catastrofica
    mantenimiento_preventivo = (D % T == 0) * costo_preventivo
    costo_mantenimiento = mantenimiento_catastrofico + mantenimiento_preventivo
```

Código: Problema 3 - Simulación base (2/2)

```
    # 3) costos de inspeccion
    num_inspecciones = np.ceil(D / T)
    costo_inspeccion_ciclo = num_inspecciones * costo_inspeccion
    # 4) costo diario esperado
    costo_total_ciclo = costo_mantenimiento + costo_inspeccion_ciclo
    costo_diario_promedio = costo_total_ciclo.mean() / D.mean()
    costos_por_dia.append(costo_diario_promedio)

idx_optimo = np.argmin(costos_por_dia)
T_opt, C_min = T_vals[idx_optimo], costos_por_dia[idx_optimo]
print(T_opt, C_min)
```

Código: Problema 3 - Sensibilidad vs. c_f (setup)

```python
import numpy as np

sim = 10000
r, p = 5, 0.20
costo_preventivo = 2000
costo_inspeccion = 400

cf_vals = np.arange(5000, 20001, 2500)
T_vals = np.arange(1, 15)
T_opts = []
C_mins = []

# media teorica de D
D_mean_fracasos = r * (1 - p) / p
D_mean = D_mean_fracasos + r
```

Código: Problema 3 - Sensibilidad vs. c_f (bucle principal)

```
15 for costo_falla_catastrofica in cf_vals:
16     costos_diarios_prom = []
17     # simular D una sola vez (CRN)
18     D_fracasos_sim = np.random.negative_binomial(r, p, size=sim)
19     D_sim = D_fracasos_sim + r
20     for T in T_vals:
21         mantenimiento_catastrofico = (D_sim % T != 0) * costo_falla_catastrofica
22         mantenimiento_preventivo = (D_sim % T == 0) * costo_preventivo
23         costo_mantenimiento = mantenimiento_catastrofico +
               mantenimiento_preventivo
24         num_inspecciones = np.ceil(D_sim / T)
25         costo_inspeccion_ciclo = num_inspecciones * costo_inspeccion
26         costo_total_ciclo = costo_mantenimiento + costo_inspeccion_ciclo
27         costo_diario_promedio = costo_total_ciclo.mean() / D_mean
28         costos_diarios_prom.append(costo_diario_promedio)
29     idx_optimo = np.argmin(costos_diarios_prom)
30     T_opt, C_min = T_vals[idx_optimo], costos_diarios_prom[idx_optimo]
31     T_opts.append(T_opt)
32     C_mins.append(C_min)
```

Problema 4: Riesgo en proyecto

Enunciado

Una constructora está planificando un proyecto que se desarrolla en tres etapas secuenciales: Diseño, Construcción y Pruebas. Tras consultar con especialistas de cada área, se estima que la duración de cada etapa es incierta pero se dieron estimativos del tiempo mínimo y máximo de cada una. Debido a restricciones de la empresa, el proyecto completo tiene un plazo límite estricto de 25 días. Cada día de retraso más allá de este límite genera un costo de penalización de \$500.

Tarea	**Distribución**	**Mínimo (a_i)**	**Máximo (b_i)**
A (Diseño)	Unif(a_A, b_A)	5 días	10 días
B (Construcción)	Unif(a_B, b_B)	8 días	15 días
C (Pruebas)	Unif(a_C, b_C)	4 días	7 días

Problema 4: Planteo del problema

Planteo del problema

- **Variable aleatoria:** $D_{total} = D_A + D_B + D_C$ donde cada $D_i \sim \text{Unif}(a_i, b_i)$
- **Parámetros:** $L = 25$ (límite), $c_p = 500$ (penalización por día de retraso)
- **Objetivos:**
  - Estimar $P(D_{total} > 25)$ = probabilidad de retraso
  - Estimar $E[c_p \cdot \text{máx}(0, D_{total} - L)]$ = costo esperado de penalización

Código: Problema 4 - Simulación base

```python
import numpy as np

fecha_limite = 25
costo_penalizacion = 500
sim = 100000
np.random.seed(42)

tareas = {'A': (5, 10), 'B': (8, 15), 'C': (4, 7)}

duracion_A = np.random.uniform(tareas['A'][0], tareas['A'][1], size=sim)
duracion_B = np.random.uniform(tareas['B'][0], tareas['B'][1], size=sim)
duracion_C = np.random.uniform(tareas['C'][0], tareas['C'][1], size=sim)
D_total = duracion_A + duracion_B + duracion_C

retraso = np.maximum(D_total - fecha_limite, 0)
E_costo = (retraso * costo_penalizacion).mean()
P_retraso = (D_total > fecha_limite).mean()
print(P_retraso, E_costo)
```

Código: Problema 4 - Sensibilidad vs. b_B

```python
import numpy as np

sim = 100000
fecha_limite = 25
costo_penalizacion = 500

tareas_fijas = {'A': (5, 10), 'C': (4, 7)}
a_B = 8

b_B_vals = np.arange(12, 21, 1)
P_retraso_vals = []
E_costo_vals = []

for b_B in b_B_vals:
    duracion_A = np.random.uniform(tareas_fijas['A'][0], tareas_fijas['A'][1],
        size=sim)
    duracion_B = np.random.uniform(a_B, b_B, size=sim)
    duracion_C = np.random.uniform(tareas_fijas['C'][0], tareas_fijas['C'][1],
        size=sim)
    D_total = duracion_A + duracion_B + duracion_C
    retraso = np.maximum(D_total - fecha_limite, 0)
    E_costo = (retraso * costo_penalizacion).mean()
    P_retraso = (D_total > fecha_limite).mean()
    P_retraso_vals.append(P_retraso)
    E_costo_vals.append(E_costo)
```

Código básico para distribuciones

- Uniforme: valores entre $[a, b]$ (ej.: `np.random.uniform(a,b,n)`)
- Normal: media μ, desvío σ (ej.: `np.random.normal(mu,sigma,n)`)
- Exponencial: tasa λ (ej.: `np.random.exponential(1/lambda,n)`)
- Poisson: tasa λ (ej.: `np.random.poisson(lam,n)`)
- Binomial: n ensayos, prob. p (ej.: `np.random.binomial(n,p,nmuestras)`)

Terminamos

¿Dudas?

¿Consultas?