Método Montecarlo

El juego de dados

Nuestro sencillo juego implicará dos dados de seis caras. Para ganar, el jugador debe obtener el mismo número en ambos dados. Un dado de seis caras tiene seis resultados posibles (1, 2, 3, 4, 5 y 6). Con dos dados, ahora hay 36 resultados posibles (1 y 1, 1 y 2, 1 y 3, etc., o 6 x 6 = 36 posibilidades). En este juego, la casa tiene más oportunidades de ganar (30 resultados frente a los 6 resultados del jugador), lo que significa que la casa tiene una gran ventaja.

Digamos que nuestro jugador empieza con un saldo de 1.000 dólares y está dispuesto a perderlo todo, así que apuesta 1 dólar en cada tirada (es decir, se lanzan ambos dados) y decide jugar 1.000 tiradas. Como la casa es tan generosa, ofrece pagar 4 veces la apuesta del jugador cuando este gana. Por ejemplo, si el jugador gana la primera tirada, su saldo aumenta en 4 dólares y termina la ronda con un saldo de 1.004 dólares. Si milagrosamente consigue una racha ganadora de 1.000 tiradas, podría irse a casa con 5.000 dólares. Si pierde todas las rondas, podría irse a casa sin nada. No es una mala relación riesgo-recompensa... o tal vez sí lo sea.

Importación de paquetes de Python

Simulemos nuestro juego para averiguar si el jugador tomó la decisión correcta al jugar.

Comenzamos nuestro código importando nuestros paquetes de Python necesarios: *Pyplot* de *Matplotlib* y *random* .

Usaremos *Pyplot* para visualizar nuestros resultados y *random* para simular una tirada normal de dados de seis caras.

```
# Importación de paquetes
import matplotlib.pyplot as plt
import random
```

Función de lanzamiento de dados

A continuación, podemos definir una función que aleatorizará un número entero de 1 a 6 para ambos dados (simulando una tirada). La función también comparará los dos dados para ver si son iguales. La función devolverá una variable booleana, *same_num* , para almacenar si las tiradas son iguales o no. Usaremos este valor más adelante para determinar acciones en nuestro código.

```
# Creación de la función Roll Dice
def roll_dice():
     die_1 = random.randint(1, 6)
     die_2 = random.randint(1, 6)
     # Determinar si los dados son del mismo número
    if die_1 == die_2:
        same_num = True
                                                          else:
        same_num = False
    return same_num
```

Entradas y variables de seguimiento

Toda simulación de Monte Carlo requerirá que usted sepa cuáles son sus entradas y qué información está buscando obtener. Ya definimos cuáles son

nuestras entradas cuando describimos el juego. Dijimos que nuestro número de lanzamientos por juego es 1,000, y la cantidad que el jugador apostará en cada lanzamiento es $1. Además de nuestras variables de entrada, necesitamos definir cuántas veces queremos simular el juego. Podemos usar la variable *num_simulations* como nuestro recuento de simulación de Monte Carlo. Cuanto más alto sea este número, más precisa será la probabilidad predicha con respecto a su valor real.

La cantidad de variables que podemos rastrear generalmente varía con la complejidad de un proyecto, por lo que es importante determinar qué información desea obtener. En este ejemplo, rastrearemos la probabilidad de victoria (victorias por juego divididas por la cantidad total de tiradas) y el saldo final de cada simulación (o juego). Estos se inicializan como listas y se actualizarán al final de cada juego.

```
# Entradas

num_simulaciones = 10000
máx_num_rolls = 1000
apuesta = 1


# Seguimiento de
probabilidad_de_ganar = []
balance_final = []
```

Configuración de la figura

El siguiente paso es configurar nuestra figura antes de ejecutar la simulación. Al hacer esto antes de la simulación, podemos agregar líneas a nuestra figura después de cada juego. Luego, una vez que hayamos ejecutado todas las simulaciones, podemos mostrar el gráfico para mostrar nuestros resultados.

```
# Creación de figuras para saldos de simulación
fig = plt.figure()
plt.title("Juego de dados de Monte Carlo [" +
str(num_simulations) + "simulations]")
plt.xlabel("Número de tirada")
plt.ylabel("Saldo [$]")
plt.xlim([0, max_num_rolls])
```

Simulación de Monte Carlo

En el código que se muestra a continuación, tenemos un bucle *for* externo que itera a través de nuestra cantidad predefinida de simulaciones (10 000 simulaciones) y un bucle *while* anidado que ejecuta cada juego (1000 tiradas). Antes de comenzar cada bucle *while* , inicializamos el saldo del jugador como $1000 (como una lista para fines de representación gráfica) y un recuento de tiradas y victorias.

Nuestro bucle *while* simulará el juego durante 1000 tiradas. Dentro de este bucle, tiramos los dados y usamos la variable booleana devuelta por *roll_dice()* para determinar el resultado. Si los dados tienen el mismo número, sumamos 4 veces la apuesta a la lista *de saldo* y sumamos una ganancia al conteo de ganancias. Si los dados son diferentes, restamos la apuesta de la lista *de saldo* . Al final de cada tirada, sumamos un conteo a nuestra lista *num_rolls* .

Una vez que el número de tiradas llega a 1000, podemos calcular la probabilidad de victoria del jugador como la cantidad de victorias dividida por la cantidad total de tiradas. También podemos almacenar el saldo final del juego completado en la variable de seguimiento *end_balance* . Por último, podemos trazar las variables *num_rolls* y *balance* para agregar una línea a la cifra que definimos anteriormente.

```
# Bucle for que se ejecutará durante la cantidad de simulaciones deseadas
para i en el rango (num_simulations):
     balance = [1000]
     num_rolls = [0]
     num_wins = 0

       # Ejecutar hasta que el jugador haya lanzado 1000 veces
        mientras num_rolls[-1] < max_num_rolls:
          same = roll_dice()

       # Resultado si los dados son  del mismo número
          if same:
              balance.append(balance[-1] + 4 * bet)
              num_wins += 1

             # Resultado si los dados son números diferentes
                                                    else:
            balance.append(balance[-1] - bet)
          num_rolls.append(num_rolls[-1] + 1)

# Almacena las  variables de seguimiento y agrega una línea a la figura

                  win_probability.append(num_wins/num_rolls[-1])
                                  end_balance.append(balance[-1])
    plt.plot(num_rolls, balance)
```

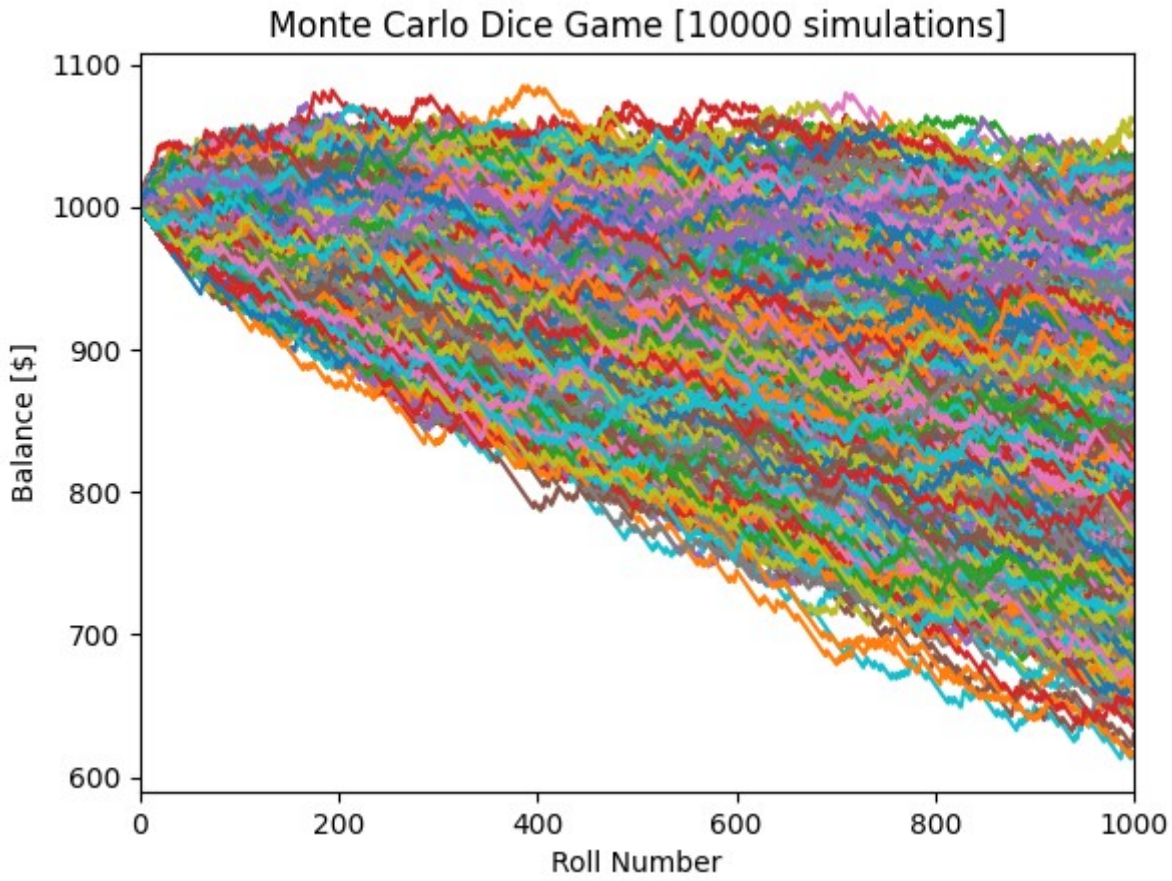
Obtención de resultados

El último paso es mostrar datos significativos de nuestras variables de seguimiento. Podemos mostrar nuestra cifra (que se muestra a continuación) que creamos en nuestro bucle *for* . Además, podemos calcular y mostrar (que se muestra a continuación) nuestra probabilidad de ganar general y el saldo final promediando nuestras listas *win_probability* y *end_balance* .

```
# Mostrar el gráfico después de finalizar las simulaciones
plt.show()

# Promedio de la probabilidad de victoria y el saldo final
 general_probabilidad_victoria =
          suma(probabilidad_victoria)/len(probabilidad_victoria)
general_balance_final = suma(balance_final)/len(balance_final)

# Visualización de los promedios
 print("Probabilidad promedio de ganar después de " +
 str(num_simulations) + "
                 ejecuciones: " + str(overall_win_probability))
print("Saldo final promedio después de " + str(num_simulations) +
"
       ejecuciones: $" + str(overall_end_balance))
```



Simulaciones de juegos de dados [Creado por el autor]

```
Probabilidad promedio de ganar después de 10 000 simulaciones:
0,1667325999999987
Saldo final promedio después de 10 000 simulaciones: $833,663
```

Análisis de resultados

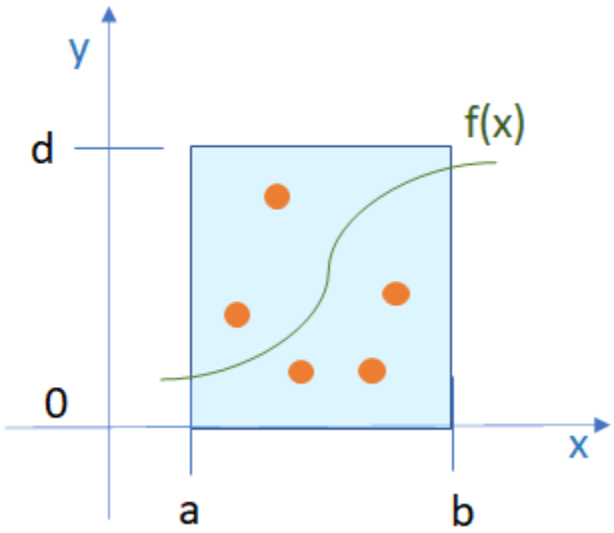
La parte más importante de cualquier simulación de Monte Carlo (o de cualquier análisis) es sacar conclusiones de los resultados. A partir de nuestra cifra, podemos determinar que el jugador rara vez obtiene ganancias después de 1000 tiradas. De hecho, el saldo final promedio de nuestras 10 000 simulaciones es de $833,66 (sus resultados pueden ser ligeramente diferentes debido a la aleatorización). Por lo tanto, aunque la

casa fue "generosa" al pagar 4 veces nuestra apuesta cuando el jugador ganó, la casa salió ganando.

También observamos que nuestra probabilidad de ganar es de aproximadamente 0,1667, o aproximadamente 1/6. Pensemos en por qué podría ser así. Volviendo a uno de los párrafos anteriores, notamos que el jugador tenía 6 resultados en los que podía ganar. También notamos que hay 36 tiradas posibles. Usando estos dos números, esperaríamos que el jugador ganara 6 de 36 tiradas, o 1/6 tiradas, lo que coincide con nuestra predicción de Monte Carlo.

Taller Ejercicio método MonteCarlo

El área bajo la curva de una función **f** puede estimarse mediante el método de Montecarlo, que consiste en lo siguiente:



- Establecer un rectángulo tal que **x** Є [a,b]; **y** Є [0,d], tal que **y**=**f**(x)
- Generar un numero **n** de puntos aleatorios tal que a<**x**<b, 0<**y**<d.
- Indicar cuántos de estos puntos caen bajo la curva **y**=**f**(**x**)

- El área bajo la curva puede estimarse mediante la relación:

$$\frac{\text{Area bajo la curva}}{\text{Area bajo el rectangulo}} = \frac{\text{numero de puntos bajo la curva}}{n}$$

Escriba un programa en python que lea las dimensiones **a**, **b**, **d** de la posición del rectángulo, genere aleatoriamente **n** pares ordenados dentro de ese rectángulo e indique cuántos puntos están bajo la curva $\mathbf{y=f(x)=xe^{-(x/2)}}$ en el intervalo [0,2]

Salir del Inframundo

'Rades (que al parecer es el inframundo) está sobrecargado, y hay que sacar gente. Arman una fila con toda la gente del inframundo que va pasando de derecha a izquierda, y en el principio de la fila está Cerberus, que al parecer es un perro de tres cabezas. Entonces los primeros tres quedan expuestos a una de las cabezas de Cerberus.

Cerberus muerde con una cabeza al azar (puede entonces morder al primero, segundo o tercero), el mordido se va a la tierra de los vivos, el/los que quedaron a la izquierda se quedan en Hades y el resto de la fila avanza para llenar nuevamente 3 lugares.

Ejemplo si Cerberus muerde al primero, ese se salva, el segundo pasa a primero, el tercero a segundo y el siguiente de la fila a tercero.

Si Cerberus muerde al segundo, ese se salva, el primero se queda en Hades y el tercero ocupa el primer lugar, el siguiente de la fila el segundo y así.

Por algún motivo sos amigo de Zeus y ve que no estás en la fila (¿te quedaste dormido?) y te ofrece lo siguiente: Podes elegir el lugar en la fila donde te quieres insertar, pero tiene que ser el mejor lugar posible, siendo este el que más chances te da de volver al mundo de los vivos.