Problema de la ruina

Supongamos que existe un juego justo entre dos jugadores, basado en volados. El jugador A empieza con M pesos, el jugador B tiene crédito y, así, no se puede quedar sin dinero. Los jugadores apuestan un peso cada uno; de modo que si el jugador A gana, obtiene un peso de ganancia, y si no, pierde un peso.

Se dice que el juego llega a la ruina si el jugador A se queda sin dinero. El problema de la ruina suele **simplificarse** de la siguiente forma: el juego se detiene si el jugador A se queda sin dinero o si logra doblar la cantidad de dinero que tenía al principio del juego.

Ejercicios

1. El juego descrito en la sección anterior ¿Es un juego justo? (¿cómo definirías un juego justo?)
   Yo pensaría que no es un juego justo porque el jugador B no se puede quedar sin dinero, entonces el jugador A es el único que pierde. Pero si vemos desde la perpectiva de lanzar una moneda si parece ser un juego justo porque cada tiene la misma probabilidad de ganar una moneda o no.
2. Simula el juego que se expuso en la sección anterior, el capital inicial M, del jugador A, debe ser un *input*. Grafica cinco ejecuciones con $M = 50$.
   Para la simulación definimos una función que nos devolverá un vector sobre como se estuvo moviendo la cantidad de monedas en cada instante n de tiempo, definimos nuestra cota que sera $2 * M$ y dado que son volados tenemos un 50% de probabilidad de ganar por eso usamos esa condición para saber si ganamos una moneda o perdemos una.

```
1  def simular_juego(M=50):
2      y = [M]
3      cota = 2*M
4      while M < cota and M > 0:
5          eleccion = random.random()
6          if eleccion < 0.50:
7              M -= 1
8          else:
9              M += 1
10         y.append(M)
11     return y
12
13 for x in range(5):
14     M = 50
15     y = np.array(simular_juego(M))
16     x = np.arange(1, len(y)+1)
17     plt.plot(x, y)
```

   Podemos ver en nuestra gráfica como las cinco simulaciones van variando, en dos de ellas el jugador A pierde, pero en las otras tres el jugador A gana.

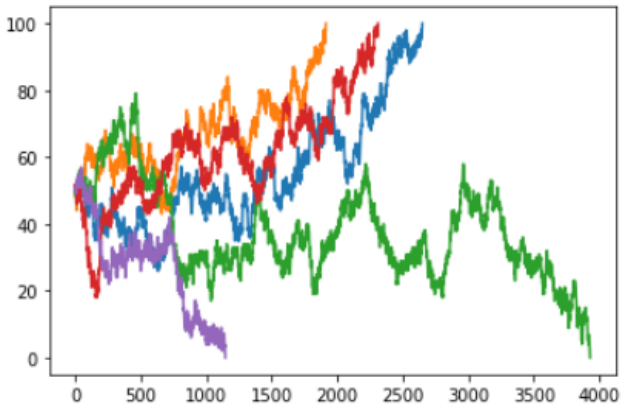

FIG. 1 – *Gráfica cinco ejecuciones con $M = 50$*

3. Realiza una simulación de los tiempos de paro τ para llegar a la ruina o a ganar $2M$, comenzando con un capital de $X_0 = M$:

$$\tau = \min\{n \geq 1 : X_n \in \{0,2M\}\}.$$

y obtén los promedios de K de estas simulaciones (Es decir, estima $\mathbb{E}[\tau]$).
Para la simulación crearemos hilos para que cada uno se encarga de hacer la simulaciones correspondiente, de este modo reducimos el tiempo de simulación para k muy grande, importaremos la librería `threading` y craremos nuestra funcion para calcular los tiempos:

```
1  import threading as th
2
3  def simular_tiempo(M=50, n=0):
4      cota = 2 *M
5      while M < cota and M > 0:
6          eleccion = random.random()
7          if eleccion < 0.50:
8              M -= 1
9          else:
10             M += 1
11         n += 1
12     return n
13
14 def calcular_tiempos(M, k, tiempos):
15     for _ in range(k):
16         ts = simular_tiempo(M)
17         tiempos.append(ts)
```

Al final sólo uniremos cada hilo con sus respectivos y sacaremos su promedio. A cada hilo le daremos que haga una simulación, tendremos un total de hilos dependiendo del número de simulaciones que hagamos, por ejemplo si vamos hacer 200 simulaciones haremos 200 donde cada hilo se encargará de hacer la simulación respectivamente

```
# Para k=200 M = 150

M = 150
num_hilos = 200
tiempos = [[] for i in range(num_hilos)]
hilos = []

# Hare cada hilo 10 simulaciones
for i in range(num_hilos):
    hilos.append(th.Thread(target=calcular_tiempos,
    args=(M, 1, tiempos[i])))

for hilo in hilos:
    hilo.start()
for hilo in hilos:
    hilo.join()

tiempo_total = []
for tiempo in tiempos:
    tiempo_total += tiempo
vec_prom = np.mean(np.array(tiempo_total))
print(vec_prom)
```

Completa la siguiente tabla con $\mathbb{E}[\tau]$ para los siguientes valores

	$K = 100$	$K = 200$	$K = 400$
$M = 20$	380.7	415.38	424.95
$M = 40$	1400.62	1618.1	1530.05
$M = 80$	6659.2	6153.53	17047.1
$M = 150$	20083.6	21752.43	25388.25

4. Realiza una simulación de los tiempos de paro para llegar a la ruina, comenzando con un capital de $X_0 = M$:

$$\tau = \min\{n \geq 1 : X_n = 0\}.$$

y obten los promedios de K de estas simulaciones (Estimaciones de $\mathbb{E}[\tau]$). Completa la siguiente tabla con $\mathbb{E}[\tau]$ para los siguientes valores
En este caso solo debemos cambiar nuestra función en la parte de la condicion que será ahora que M sea mayor a 0:

```
def simular_tiempo_ruina(M=50, n=0):
    while M > 0:
        eleccion = random.random()
        if eleccion < 0.50:
            M -= 1
        else:
            M += 1
        n += 1
    return n
```

	$K = 100$	$K = 200$	$K = 400$
$M = 20$	962787.92	1259972.21	...
$M = 40$	1447270.3	6503360.95	...
$M = 80$	4613876.82	87503360.95	...
$M = 150$	692474.88	4503360.95	...

La ultima columna ya no se pudo calcular debido a que tardo demasiado la maquina, parecia nunca terminar.

5. Compara los resultados de las dos tablas. ¿La probabilidad de ganar el juego depende de la cantidad inicial de capital?
   Si definitivamente, dependiendo de donde inicies puede ser que se tarde mucho más en ganar, pero también en algunos caso cuando se iniciaba con un valor de M muy pequeño también tardaba, mismo caso cuando se tenia un M muy grande aveces ganaba muy rápido. También pude observar que algunos valores M con valore intermedios por ejemplo con $M = 80$ llegaba tardar una gran cantidad de tiempo mucho más cuando se tenia un $M = 150$ con 200 simulaciones, puede concluir que la probabilidad de ganar juego depende de la cantidad inicial del juego y en algunos casos podemos ver que el tiempo de ganancia es más rápido a pesar de la cantidad inicial.