Trabajo Práctico Especial

Teoría de la Información

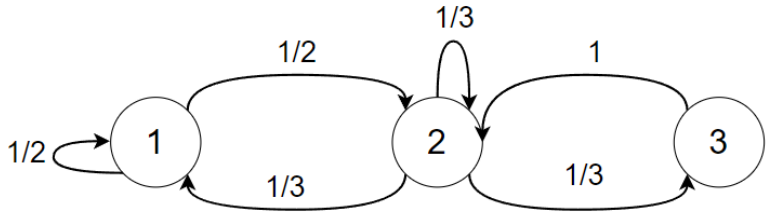
Fuentes de Información

Fecha de entrega: 12/05/2022
Grupo 3
Integrantes:

Raffin, Giuliana
Zárate, Francisca Pilar

a. Calcular la distribución de probabilidades estacionarias.

Este ejercicio propone analizar cómo actúa la fuente descrita a continuación cuando su comportamiento a largo plazo da lugar a una estabilidad probabilística. La fuente a tratar incluye 3 símbolos {1, 2, 3} y la elección del símbolo a emitir respecto del emitido está dada por el siguiente grafo:



Para resolver este problema, como fue solicitado, utilizamos el motor de Montecarlo. Éste método consiste en aproximar expresiones complejas mediante generación aleatoria de probabilidades. Para corroborar el resultado obtenido por simulación comparamos las probabilidades estacionarias de cada uno de los tests realizados con el desarrollo analítico, pedido en el ejercicio 4 del trabajo práctico 2, puesto a continuación.

p1 = ½ p1 + ⅓ p2 → p1 = ⅔ p2 → p1 = ⅓
p3 = ⅓ p2 → p3 = ⅙
p1 + p2 + p3 = 1 → ⅔ p2 + p2 + ⅓ p2 = 1 → p2 = ½

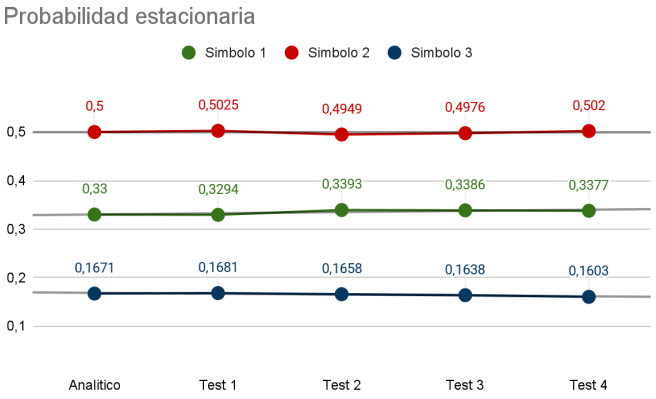
V* = (⅓, ½, ⅙)

↙	1	2	3
1	1/2	1/3	0
2	1/2	1/3	1
3	0	1/3	0

Al comparar los resultados obtenidos tanto computacional como manualmente llegamos a la conclusión de que aplicando Montecarlo se llega a un resultado aproximado casi exacto de la distribución de probabilidades estacionarias.
En la siguiente tabla se muestra la escasa diferencia entre los resultados obtenidos de los tests de simulación utilizando un epsilon de 0.0001 y el desarrollo analítico. El motivo por el cuál se realiza más de un test, es porque, como se puede ver en la tabla, dependiendo del test el resultado se acerca en mayor o menor medida al obtenido analíticamente, es decir, el error es mayor o menor dependiendo de la ejecución.

	resultado analítico	test 1	test 2	test 3	test 4
símbolo 1	1/3 ≃ 0,33	0,3294	0,3393	0,3386	0,3377
símbolo 2	1/2 = 0,5	0,5025	0,4949	0,4976	0,5020
símbolo 3	1/6 ≃ 0,167	0,1681	0,1658	0,1638	0,1603

Para un mejor entendimiento, se adjunta un gráfico con los resultados obtenidos que refleja nuevamente la escasa diferencia entre ellos.



El pseudocódigo que se utilizó de base para la implementación del código final del ejercicio *a* se muestra a continuación:

```
Calcular_Probabilidad_Estacionaria(){
      //inicialización
      emisiones = [0,0,0] //cantidad de emisiones
      Vt = [0,0,0] //vector de estado actual
      Vt_ant = [-1,0,0] //vector de estado anterior
      cant_simb = 0 //cantidad de símbolos emitidos

      //montecarlo
      s = primerSimbolo()
      while (not converge(Vt, Vt_ant) or cant_simb < S_MIN){
            s = sigSimbolo(s);
            cant_simb++
            Vt_ant ← Vt
            emisiones[s] = emisiones[s]+1
            for (i = 0 to #simbolos){
               Vt[i] = emisiones[i]/cant_simb
            }
      }
      return Vt
}

converge(a[], b[]){
      for (i = 0 to #simbolos){
            if (abs(a[i] - b[i]) > ε)
                  return false
      }
      return true
}

primerSimbolo(){
      V0acum = [1/3, 2/3, 1] //tomamos equiprobabilidad para elegir el primer simbolo
      r = rand()
      for (i = 0 to #simbolos){
            if (r < V0acum[i])
                  return i
      }
}

sigSimbolo(ant){
      M_acum = [1/2, 1, 1], [1/3, 2/3, 1], [0, 1, 1] //matriz de transición acumulada
      r = rand()
      for (i = 0 to #simbolos){
            if (r < M_acum[ant][i])
                  return i
      }
}
```

b) Calcular la probabilidad de 1° recurrencia en N pasos (N= 1, 2, …, 5) para cada símbolo (1, 2 y 3)

Este inciso plantea calcular la probabilidad de que se vuelva a emitir por primera vez uno de los símbolos de la fuente en un número de pasos *n* determinado dado que ya se emitió en un tiempo anterior y sin pasar por el mismo en pasos intermedios.
Para ello, es necesario el estudio de la 1° recurrencia en *n* pasos según cada símbolo, en este caso, 1, 2 y 3 siendo n = 1, 2, …, 5.
Así como en el ítem anterior, este problema fue resuelto haciendo uso del motor de Montecarlo por la parte computarizada, y de un desarrollo análitico el cual se muestra a continuación para el caso del símbolo 3.

$$f_{3/3}^{(1)} = 0$$

$$f_{3/3}^{(2)} = P_{3/3}^{(2)} - \sum_{m=1}^{1} f_{3/3}^{(m)} * P_{3/3}^{(2-m)} = P_{3/3}^{(2)} - f_{3/3}^{(1)} * P_{3/3}^{(1)} = \frac{1}{3}$$

$$f_{3/3}^{(3)} = P_{3/3}^{(3)} - \sum_{m=1}^{2} f_{3/3}^{(m)} * P_{3/3}^{(3-m)} = P_{3/3}^{(3)} - f_{3/3}^{(1)} * P_{3/3}^{(2)} - f_{3/3}^{(2)} * P_{3/3}^{(1)} = \frac{1}{9}$$

$$f_{3/3}^{(4)} = P_{3/3}^{(4)} - \sum_{m=1}^{3} f_{3/3}^{(m)} * P_{3/3}^{(4-m)} = P_{3/3}^{(4)} - f_{3/3}^{(1)} * P_{3/3}^{(3)} - f_{3/3}^{(2)} * P_{3/3}^{(2)} - f_{3/3}^{(3)} * P_{3/3}^{(1)} = \frac{5}{54}$$

$$f_{3/3}^{(5)} = P_{3/3}^{(5)} - \sum_{m=1}^{4} f_{3/3}^{(m)} * P_{3/3}^{(5-m)} = P_{3/3}^{(5)} - f_{3/3}^{(1)} * P_{3/3}^{(4)} - f_{3/3}^{(2)} * P_{3/3}^{(3)} - f_{3/3}^{(3)} * P_{3/3}^{(2)} - f_{3/3}^{(4)} * P_{3/3}^{(1)} = \frac{25}{324}$$

En el desarrollo anterior se muestra cómo se realizaría el cálculo de la probabilidad de 1° recurrencia en *n* pasos de manera analítica para el símbolo 3, cabe aclarar que se realiza análogamente para los símbolos 1 y 2. Como se puede observar, el código no hace uso de la fórmula, sino que toma la cantidad de veces que un símbolo se volvió a emitir, cuántas de esas veces emitidas fueron en los *n* pasos pedidos y cuál fue la última vez que se retornó a ese símbolo para calcular las probabilidades de primera recurrencia. Pudimos notar que a medida que el *n* aumenta, el cálculo analítico se complejiza en gran medida, ya que los “caminos” posibles a tener en cuenta aumentan, en este caso. Además, al ser un algoritmo recursivo, hay más probabilidades de arrastrar errores. Esto no sucede al utilizar el motor de Montecarlo para la implementación del código. Permite realizar cuentas simples e independientes, es decir que no dependen unas de otras, y simplifica ampliamente los cálculos y el desarrollo para llegar a los resultados. Los resultados de la ejecución del código mostrados en la tabla siguiente se obtuvieron utilizando un epsilon de 0.0001, de haber utilizado un epsilon más pequeño, el error también lo sería ya que se disminuye aún más la diferencia que debe existir entre la emisión actual y la anterior para que converja.

	1 paso		2 pasos		3 pasos		4 pasos		5 pasos	
	manual	mont.	manual	mont.	manual	mont.	manual	mont.	manual	mont.
símb. 1	1/2 = 0,5	0,4863	1/6 ≃ 0,16	0,1599	1/18 ≃ 0,05	0,0640	2/27 ≃ 0,074	0,0843	7/162 ≃ 0,043	0,0438
símb. 2	1/3 ≃ 0,33	0,3356	1/2 = 0,5	0,5085	1/12 = 0,083	0,0829	1/24 ≃ 0,042	0,0365	1/48 ≃ 0,021	0,0189
símb. 3	0	0,0000	1/3 ≃ 0,33	0,3386	1/9 ≃ 0,11	0,111	5/54 ≃ 0,093	0,0992	25/324 ≃ 0,077	0,0789

El pseudocódigo del cálculo propuesto para todos los símbolos de la fuente descrita se muestra a continuación. Cabe destacar que el código de las funciones *sigSimbolo* y *converge* no sufrió modificaciones respecto del inciso anterior, por lo tanto, no se incluyen en esta sección.

```
CalcularProbabilidadPrimeraRecurrencia (){
      //inicialización
      recurrencias = [0,0,0] //vector con la cantidad de retornos
      fi_i = [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0] //matriz de
      prob. de 1° recurrencia para cada símbolo
      fi_i_ant = [-1, -1, -1], [-1, -1, -1], [-1, -1, -1] //matriz de 1° recurrencia
      anterior para cada símbolo
      retornos_totales = [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]
      ultimo_retorno = [0,0,0] //vector con el último paso donde se retornó a cada
      símbolo
      paso_actual = 0
      s = 0 //se eligió a 0 que representa al 1 como primer símbolo
      pasos = 0
      while (not converge(fi_i[s], fi_i_ant[s]) or paso_actual < 10000){
            s = sigSimbolo(s)
            paso_actual++
            pasos = paso_actual - ultimo_retorno[s]
            if (ultimo_retorno[s] > 0)
                  recurrencias[s]++
            fi_i_ant ← fi_i
            if (pasos < 6){
                  retornos_totales[s][pasos]++
                  for (i = 0 to #simbolos){
                        fi_i[i][pasos] = retornos_totales[i][pasos]/recurrencias[i]
                  }
            }
            ultimo_retorno[s] = paso_actual
      }
      return fi_i //se devuelven las probabilidades de primera recurrencia en n pasos
      obtenida
}
```

Conclusiones generales

Para realizar el trabajo hubo que realizar un razonamiento previo para entender el funcionamiento del motor de Montecarlo para así adaptarlo al problema planteado. Se modificó el pseudocódigo y, finalmente, el código para lograr la mayor eficiencia posible. También se realizó un análisis de los resultados obtenidos por cada ejercicio de manera independiente para tener una mejor organización.

Para acceder al código de ambos incisos en replit haga click aquí.