Universidade Federal do Rio Janeiro
Bacharelado em Ciência da Computação
Estatística e Probabilidade

Trabalho de Simulação

Eduardo da Silva Barbosa - DRE: 116150432
Tainá da Silva Lima - DRE: 116165607
Rio de Janeiro
2019

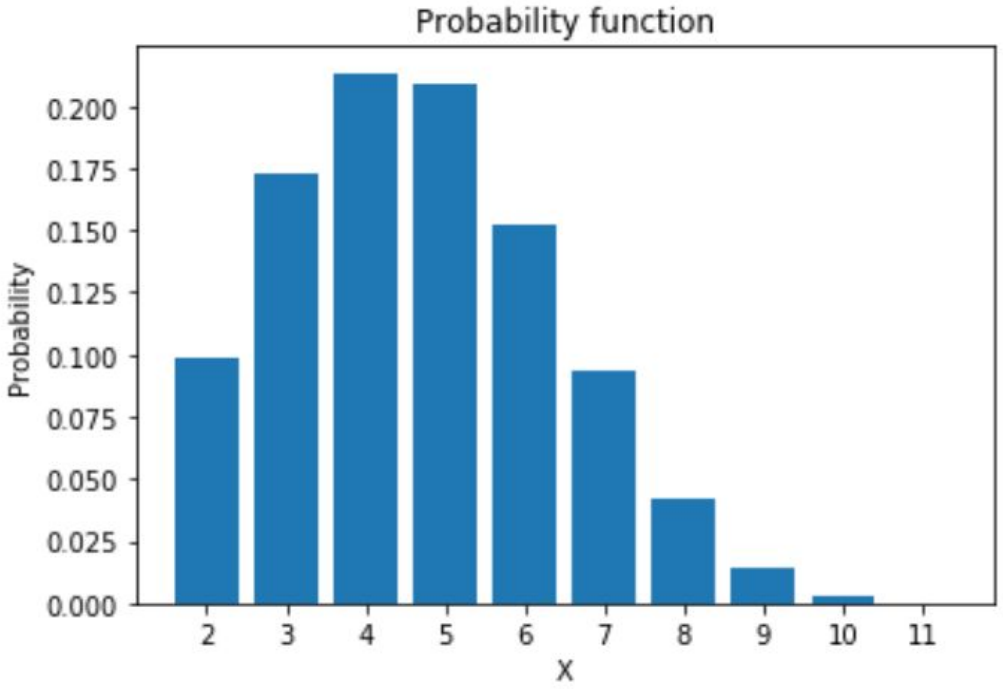
Questão 1

Uma urna contém 10 bolas numeradas de 1 a 10. Bolas são extraídas com reposição da urna até que um número anterior seja retirado novamente. Seja X a variável aleatória que conta o número de retiradas até que isto ocorra. Construa um algoritmo para a simulação de X de forma a obter sua função de probabilidade empírica e sua esperança matemática. (Faça 5.000 simulações)

Solução:

```
experimento(): #Função que representa um experimento.
      defina bolas #Lista de 10 posições zeradas, que guardará a
      quantidade de vezes que cada bola foi retirada.
      passos = 0 #Variável que contará a quantidade total de
      bolas retiradas.
      para i de 0 até 10 faça: #Representa 11 retiradas de
bolas.
            passos = passos + 1 #Soma um na quantidade total de
            bolas retiradas.
            bola = random(0,9) #Uma das 10 bolas é sorteada
            bolas[bola] = bolas[bola] + 1 #Soma um na quantidade
            total da bola retirada.
            se a quantidade da bola retirada for maior que 1:
                  retorna passos

função_probabilidade(frequencia_X, N°_Simulações): #Função que
encontra a função probabilidade de X.
      defina distribuição _X #Lista de 12 posições zeradas, que
      guardará a probabilidade de cada X.
      para i de 2 até 11 faça:
            distribuição _X[i] = frequencia_X[i]/N°_Simulações
      retorna distribuição _X

esperança(X, P):#Retorna a esperança de X dada a função de
probabilidade P.
      Soma = 0 #Armazenará o resultado final da esperança.
      para i de 0 até 11 faça:
            soma = soma + X[i]*P[i]
      retorna soma


variância(X,P): #Retorna a variância de X dada sua distribuição
de probabilidade
```

```
    esperancaX2 = 0 #Guarda o valor da E(X²)
    para i de 0 até 11 faça: #Para cada x
        esperancaX2 = esperancaX2 + X[i]*X[i]*P[i]
    retorna esperancaX2 - esperança(X,P)*esperança(X,P)

#MAIN

defina frequencia_X #Lista de 12 posições zeradas, que armazenará a quantidade de ocorrência de cada X = {2,3,...,11}. OBS: as posições 0 e 1 não serão utilizadas.
defina imgX #Lista com todos os possíveis valores de X (2,3,...,11)
para i de 0 até 5000 faça:
    resultado = experimento() #Chama o experimento e guarda o X em resultado.
    frequencia_X[resultado] = frequencia_X[resultado] + 1 #Atualiza a quantidade de ocorrência de um determinado X.


distribuição_X = função_probabilidade(frequencia_X, 5000) #Armazena a probabilidade de cada X.
esp = esperança(imgX, distribuição_X[2:])#distribuição_X sem as duas primeiras posições.
var = variância(imgX, distribuição_X[2:]) #distribuição_X sem as duas primeiras posições.

imprime(distribuição_X[2:])
imprime(esp)
imprime(var)
```



	x	P(X=x)
0	2	0.0988
1	3	0.1730
2	4	0.2134
3	5	0.2086
4	6	0.1526
5	7	0.0940
6	8	0.0418
7	9	0.0142
8	10	0.0034
9	11	0.0002

A esperança da variável X é:4.685199999999999
A variância da variável X é:2.9197009600000143

Questão 2

Seja X uma variável aleatória com densidade:

$$f(x) = \begin{cases} \frac{10}{x^2} & , x > 10 \\ 0 & , x \leq 10 \end{cases}$$

Dê um método para simular a variável aleatória X. Para 1.000 simulações da variável X, obtenha a média de X e compare o resultado com o valor verdadeiro.

Solução:

Dado a variável aleatória X apresentada pela questão, onde sua função de densidade é:

$$f(x) = \begin{cases} \frac{10}{x^2} & , x > 10 \\ 0 & , x \leq 10 \end{cases}$$

A partir disso, podemos calcular a sua função de densidade acumulada, dada por:

$$F_x(x) = \int_{-\infty}^{x} f(t)dt = \int_{10}^{x} \frac{10}{t^2} dt = \left[-\frac{10}{t}\right]_{10}^{x}$$

$$F_x(x) = 1 - \frac{10}{x}, x \geq 10$$

Utilizando isso, podemos considerar U = F(X) , onde por conta de U ser a função de distribuição acumulada de X, U tem distribuição uniforme em (0,1). Ou seja:

$$U = 1 - \frac{10}{X}$$

Portanto:

$$X = \frac{10}{1-U}$$

Tal definição será utilizada no código abaixo para simular X, bem como calcular sua esperança.

```
#MAIN
media = 0
para i de 0 até 999 faça:
  x = 10/(1-random(0,1)) #Simulação de X (X = 10/1-U)
  media = media + x
```

```
imprime("A esperança de X é aproximadamente:", media/1000)
```

Executando o código acima várias vezes, é possível verificar que a média resulta em valores relativamente distintos, de maneira que não esteja convergindo para nenhum valor.
Isso se dá porque ao calcular da esperança de maneira analítica, temos que esta tende ao infinito, tornando-a divergente.

$$E(X) = \int_{-\infty}^{\infty} x f(x) dx = \int_{10}^{\infty} x \frac{10}{x^2} dx$$

$$= 10 \int_{10}^{\infty} \frac{1}{x} dx$$

$$= 10[lnx]_{10}^{\infty}$$

$$= 10\,[lim_{x \to \infty} lnx - ln10] = \infty$$

Questão 3

Faça um algoritmo, usando o método de Monte Carlo, para calcular:

$$\int_{-\infty}^{\infty} 4x^2 e^{-3x^2} dx\,.$$

Solução:

```
num_de_pontos = 10000
f(x): #Representa o integrando
  Retorna (4*(x**2)*math.exp(-3*(x**2)))

gerador_de_pontos(n,inferior,superior):#Gera n pontos aleatórios cujo o x está no intervalo enviado.
  pontos = [] #Armazenará os n pontos gerados
  para i de 0 até n-1 faça:
      x = random(inferior,superior)#Gera um randômico entre o valor inferior e o superior
      ponto = [x, f(x)] # Encontra o ponto a partir de x.
      pontos.append(ponto) #Adiciona o ponto gerado à pontos.

  retorna pontos
```

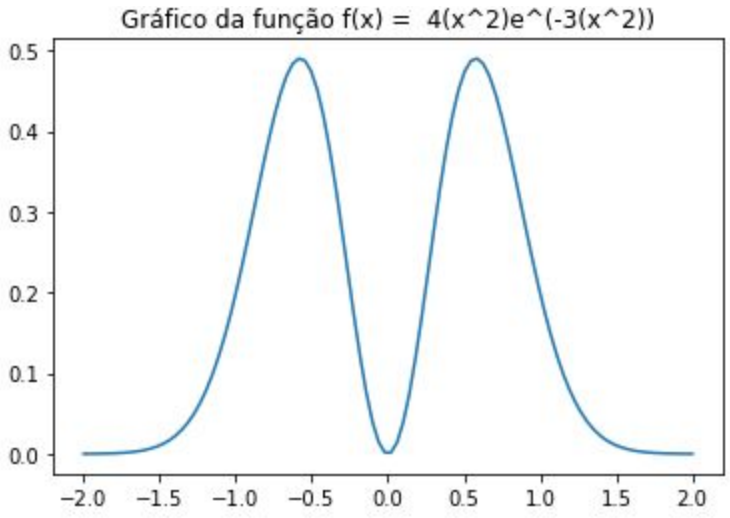
```
monteCarlo(inferior, superior):#Calcula a integral através do método de Monte Carlo
      pontos=          gerador_de_pontos(num_de_pontos,inferior, superior)
      soma = 0
      para cada ponto em pontos:
            soma = soma + ponto[1]
      media = soma/num_de_pontos #Dado os pontos escolhidos, é feita a média de suas imagens por f (aproximação da esperança)
      retorna (superior - inferior)*media #A aproximação da integral desejada é dada pela média*(tamanho do intervalo)

#MAIN

imprime("O resultado da integral desejada é aproximadamente: ", monteCarlo(-1.5,1.5))
```

O método de Monte Carlo permite que integrais definidas de funções que analiticamente não seriam possíveis de se encontrar um resultado, sejam resolvidas de uma maneira relativamente simples.
Basicamente o método calcula tais integrais utilizando a ideia da esperança, encontrada através de manipulações algébricas na integral. Outra maneira de se ver é através do cálculo do ponto médio de uma função, dado pela seguinte equação:

$$E = \frac{1}{b-a}\int_a^b f(x)dx$$

$$\int_a^b f(x)dx = E(b-a)$$

Para a resolução deste exercício, não foi feita a integral abaixo do integrando de -∞ a ∞, e sim somente abaixo desses dois "morros" visíveis no gráfico, que é onde a maior parte da área desejada está. Como a função decresce rapidamente para 0 para x > 1.5 e x < -1.5 aproximadamente), a área abaixo de f(x) nesses intervalos são muito pequenas. Portanto, o erro gerado será pequeno.
Sendo assim, a integral foi feita entre -1.5 e 1.5. Foi criado vários pontos dentro deste intervalo e calculado suas respectivas imagens, e, partir disso, a média foi encontrada e multiplicada pelo tamanho do intervalo dos limites de integração.
Executando várias vezes este algoritmo, chegamos a valores muito próximos, a maioria deles por volta de 0.68.



Questão 4

Considere uma partícula que se move ao longo de 6 nós numerados 0, 1, 2, 3, 4, 5 e arranjados em torno de um círculo. A cada passo, a partícula é igualmente provável de se mover uma posição no sentido ou horário ou anti-horário. Isto é, se X_n é a posição da partícula após seu n-ésimo passo, então:

$$P(X_{n+1} = i + 1 \mid X_n = i) = P(X_{n+1} = i - 1 \mid X_n = i) = 1/2$$

(onde i + 1 := 0 se i = 5, e i - 1 := 5 se i = 0).
Suponha que a partícula começa no 0 e continua a se mover de acordo com a regra acima, até que todos os nós (1, 2, 3, 4, 5) tenham sido visitados. Construa um algoritmo para simular a variável aleatória X, que representa o número de movimentos feitos pela partícula até que todos os nós sejam visitados. Calcule também E(X). Faça 5.000 simulações.

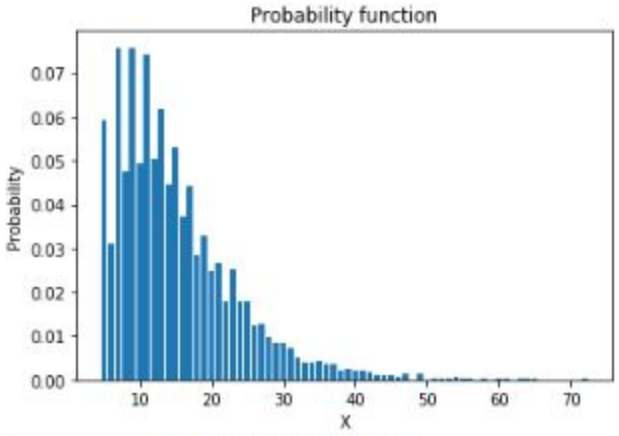
Solução:

```
experimento(): #Representa o experimento de visitar todos os nós ao menos uma vez.
   defina nó #Lista de 6 posições zeradas, onde cada posição representa um nó.
  nó[0] = 1 #O nó 0 já começa como visitado.
  i = 0 #Representa o atual nó, que começa no 0.
  cont = 0 #Representa a quantidade de movimentos da particula.
```

```
 enquanto(0 pertence nó): #Enquanto houver nós não visitado o código abaixo será executado.
      cont = cont + 1 #soma o total de movimentos em 1.
      se(random(1,10) > 5):#Caso o random seja maior que 5 ele vai para o próximo nó.
            nó[(i+1)%6]+=1  #Soma em um o nó para o qual a partícula se moverá.
            i = (i+1)%6 #Atualiza a posição atual da partícula
      senão:#Caso contrário ele volta um nó.
            se(i == 0):
                  nó[5]+=1  #Soma em um o nó para o qual a partícula se moverá.
                  i = 5 #Atualiza a posição atual da partícula.
            senão:
                  nó[(i-1)%6]+=1 Soma em um o nó para o qual a partícula se moverá.
                  i=(i-1)%6   #Atualiza a posição atual da partícula.

  retorna cont #Retorna a quantidade de movimentos

simulação(num_de_simulações):
      defina X #Um conjunto de listas, onde, para cada lista o primeiro elemento é a chave (um valor de X) e o segundo é o valor (número de vezes em que esse x ocorreu)
      para i de 0 até num_de_simulações faça: #Para cada simulação
            temp = experimento() #Faço um experimento e guardo seu resultado em temp
            se (temp é alguma chave de algum x que pertence a X):
            #Se temp já foi gerado por algum experimento
                  x[1] = x[1] + 1  #Adiciono 1 ao seu contador
            senão:
                  x[1] = 1 #Coloco 1 no seu contador
      retorna X

função_de_probabilidade(X,num_de_simulações): #Retorna a função de probabilidade.
      para cada x em X:
            x[1] = x[1]/num_de_simulações
      retorna X

esperança(X): # Retorna a esperança de X
      soma = 0
      para cada x em X:
```

```
            soma = soma + x[0]*x[1]
        retorna soma
#MAIN
num_de_simulações = 5000
x = simulação(num_de_simulações)
x = função_de_probabilidade(x, num_de_simulações)
esp = esperança(x)
imprime(x)#Imprime a função de probabilidade
imprime("Esperança: ", esp)
```

	x = número de movimentos	P(X=x)
0	5	0.0442
1	6	0.0082
2	7	0.0530
3	8	0.0504
4	9	0.0744
5	10	0.0250
6	11	0.0476
7	12	0.0592
8	13	0.0758
9	14	0.0756
10	15	0.0286
11	16	0.0020
12	17	0.0248
13	18	0.0268
14	19	0.0374
15	20	0.0178
16	21	0.0312
17	22	0.0618
18	23	0.0044
19	24	0.0492
20	25	0.0180
21	26	0.0026
22	27	0.0178
23	28	0.0446
24	29	0.0002
25	30	0.0098
26	31	0.0328
27	32	0.0036
28	33	0.0006
29	34	0.0126
30	35	0.0122
31	36	0.0040
32	37	0.0006
33	38	0.0074
34	39	0.0082
35	40	0.0052
36	41	0.0002
37	42	0.0022
38	43	0.0012
39	44	0.0040
40	45	0.0002
41	46	0.0016
42	47	0.0004
43	49	0.0010
44	51	0.0036
45	52	0.0014
46	53	0.0014
47	54	0.0012
48	55	0.0002
49	56	0.0004
50	58	0.0020
51	60	0.0004
52	61	0.0002
53	63	0.0002
54	64	0.0002
55	65	0.0002
56	72	0.0002




```
Esperança:  15.033600000000002
```

Questão 5

Seja a Cadeia de Markov com quatro estados 0, 1, 2 e 3 com probabilidade de transição dada por:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0,3 & 0 & 0,7 & 0 \\ 0 & 0,3 & 0 & 0,7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

começando no estado 1. Por meio de 1.000 simulações estocásticas, obtenha:
(a) a probabilidade de a cadeia ser absorvida no estado 0;
(b) a probabilidade de a cadeia ser absorvida no estado 1;
(c) o número médio de visitas ao estado 1 antes da absorção;
(d) o número médio de visitas ao estado 2 antes da absorção.

Solução:

```
classe CadeiaMarkov:
      MarkovCadeia(num_estados,P): #Construtor
            n = num_estados
            estado_atual = -1
            P = P
            defina visitacao #Lista de 4 posições zeradas, com
            exceção da posição 1 que começa com o valor 1. Tal
            lista representa a quantidade de vezes que cada
            estado foi visitado.
      inicia(estado_inicial): #Inicializa a cadeia dado o estado
      inicial
            estado_atual = estado_inicial
            enquanto(algum estado de absorção não for atingido ):
                  estado_atual = move()#Faz um movimento,
            retornando o estado para onde me movi
                  visitação[estado_atual]=visitação[estado_atual]
            +1
            retorna estado_atual, visitação
```

```
      # A função move realiza o movimento e sua ideia é comparar
      o random sorteador (0 até 1) com cada coluna da matriz de
      transição,
      # referente a linha do estado atual:
      # 1) Caso o random seja menor que a probabilidade da
      primeira saída(coluna), então o número da coluna será o
      novo estado atual.
      # 2) Caso contrário deverá ir para a próxima coluna e ver
      se o random é menor que a soma da probabilidade da coluna
      anterior com a atual.
      #     Caso seja, a atual coluna será o novo estado, caso
      contrário volta para o passo 2.
      move():
            temp = 0
            prox_estado = -1
            random = ?
            para i de 0 até 4 faça:
                  temp = temp + P[estado_atual][i]
                  se (random < temp) e ( P[estado_atual][i] é
                  diferente de 0):
                        prox_estado = i
                        retorna prox_estado

num_de_simulações = 1000
num_estados = 4
P = [[1,0,0,0],
       [0.3,0,0.7,0],
       [0,0.3,0,0,7],
       [0,0,0,1]]

defina ondeFoiAbsorvido #Lista que guarda o número de vezes em
que a cadeia foi absorvido no 0 e em 3
defina numVisitaçõesTotal #Lista que guarda o número de visitas
feitas ao estado 1 e 2 antes da absorção

para i de 0 até num_de_simulações: #Para cada simulação
      cadeia = CadeiaMarkov(num_estados,P)#Crio uma cadeia de
      markov com num_estados estados e matriz de probabilidade
      de transição P
      estado_final, visitação = cadeia.inicia(1)#Inicia a cadeia
      no estado 1

      se (estado_final == 0):
            ondeFoiAbsorvido[0] = ondeFoiAbsorvido[0] + 1
```

```
        senão se  (estado_final == 3):
              ondeFoiAbsorvido[1] = ondeFoiAbsorvido[1] + 1

        numVisitaçõesTotal[0] = numVisitaçõesTotal[0] +
        visitações[1]
        numVisitaçõesTotal[1] = numVisitaçõesTotal[1] +
        visitações[2]

imprime("Probabilidade da cadeia ser absorvida no estado
0:",ondeFoiAbsorvido [0]/num_de_simulações)
imprime("Probabilidade da cadeia ser absorvida no estado
3:",ondeFoiAbsorvido [1]/num_de_simulações)
imprime("O valor médio de visitas de visitas ao estado 1 antes
da absorção é:", numVisitaçõesTotal[0]/num_de_simulações)
imprime("O valor médio de visitas de visitas ao estado 2 antes
da absorção é:", numVisitaçõesTotal[1]/num_de_simulações)
```

```
Probabilidade da cadeia ser absorvida no estado 0: 0.39
Probabilidade da cadeia ser absorvida no estado 3: 0.61
O valor médio de visitas de visitas ao estado 1 antes da absorção é: 1.251
O valor médio de visitas de visitas ao estado 2 antes da absorção é: 0.861
```