lista1

September 19, 2023

1 Estatística Computacional - Lista de Exercícios 1

Aluno: Guilherme Barão Dre: 121062490

[161]:
```
import numpy as np
import pandas as pd
from random import seed
from random import random
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

Garantindo que os valores serão replicáveis:

[162]:
```
rng = np.random.default_rng(seed=1)
```

1.1 Questão 1:

1. Considere-se uma cadeia de Markov com estados $E = \{0, 1, 2\}$ com matriz de transição

$$\mathbf{P} = \begin{pmatrix} 1/2 & 1/6 & 1/3 \\ 1/5 & 1/4 & 11/20 \\ 1/3 & 1/6 & 1/2 \end{pmatrix}$$

Assumindo que a distribuição inicial $\pi = (1/4, 1/6, 7/12)$. Simule a trajetória de uma cadeia de Markov.

[163]:
```
matriz_prob = np.array([[1/2, 1/6, 1/3],
                                                [1/5, 1/4, 11/20],
                                                [1/3, 1/6, 1/2]])
dist_inicial = np.array([[1/4, 1/6, 7/12]])
```

[181]:
```
estado = dist_inicial
estados = dist_inicial
for x in range(50):
        estado = np.dot(estado, matriz_prob)
        estados = np.append(estados, estado, axis=0)
        historico = pd.DataFrame(estados)
```

```
px.line(historico, title='Trajeto da cadeia de Markov')
```

1.2 Questão 2/3:

2. Seja X uma variável aleatória com distribuição de Laplace padrão
3. cuja densidade é dada por $f(x) = \frac{1}{2}e^{-|x|}$, onde $x \in \mathbb{R}$. Utilize o método da transformação inversa para gerar uma amostra aleatória de tamanho 1000 dessa distribuição. Utilize o método de aceitação-rejeição e compare os resultados.

Pelo método da transformação inversa:

$f(x) = \frac{1}{2}e^{|-x|}$

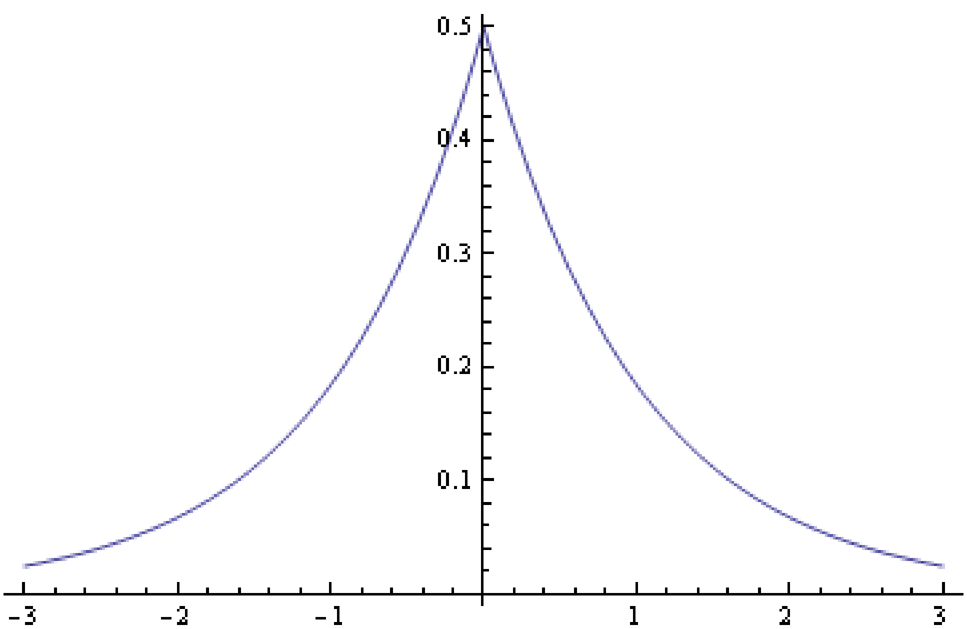
$F(x) = \begin{cases} \frac{1}{2}e^{x}, se\ x < 0 \\ 1 - \frac{1}{2}e^{x}, se\ x \geq 0 \end{cases}$

$U = F(x) \qquad F(x) = \frac{1}{2}$ quando $x = 0$

$x = \begin{cases} \ln(2U), se\ U < \frac{1}{2} \\ -\ln(2 - 2U), se\ U \geq \frac{1}{2} \end{cases}$

[165]:
```
U = rng.uniform(low=0,high=1,size=1000)
laplace_inversa = []
for u in U:
        if u < 0.5  :
                laplace_inversa.append(np.log(2*u))
        else:
                laplace_inversa.append(-np.log(2-(2*u)))
```

Pelo método de aceitação/rejeição:



$max(f(x)) = \frac{1}{2}$, tomando $-5 \leq x \leq 5$ como domínio:

[166]:
```
laplace_AR = []
while len(laplace_AR) < 1000:
        Ux = rng.uniform(low=-5,high=5,size=1)
        Uy = rng.uniform(low=0,high=1/2,size=1)
        f_u = 1/2*(np.exp(-np.abs(Ux)))
        if Uy < f_u:
                laplace_AR.append(float(Ux))
```

[167]:
```
original = rng.laplace(loc=0.0, scale=1.0, size=1000)
```

[168]:
```
fig = go.Figure()
fig.add_trace(go.Histogram(x=laplace_inversa, name='Método da tranformação␣
 ↪inversa'))
fig.add_trace(go.Histogram(x=laplace_AR, name='Método de aceitação-rejeição'))
fig.add_trace(go.Histogram(x=original, name='Numpy'))

fig.update_layout(barmode='overlay', title=dict(text='Histograma dos valores␣
 ↪gerados', font=dict(size=30)))
# Reduce opacity to see both histograms
fig.update_traces(opacity=0.75)
fig.show()
```

Podemos ver que ambos os métodos geram um histograma parecido com o dos valores gerados pelo Numpy, indicando uma boa aproximação.

1.3 Questão 4/5

4. Escreva uma função para gerar variáveis aleatórias a partir de uma distribuição Lognormal(μ, σ) usando um método de transformação e gere uma amostra aleatória de tamanho 1000. Compare o histograma com a curva de densidade lognormal fornecida pela função `dlnorm` em R.

5.

Como a Lognormal é a exponencial de uma distribuição normal, podemos utilizar o método Box-Muller para gerar amostras da distribuição normal e transformá-las em Lognormal.

O método:

Suppose U_1 and U_2 are independent random variables that are uniformly distributed in the interval (0, 1). Let

$$Z_0 = R\cos(\Theta) = \sqrt{-2\ln U_1}\cos(2\pi U_2)$$

and

$$Z_1 = R\sin(\Theta) = \sqrt{-2\ln U_1}\sin(2\pi U_2).$$

Then Z_0 and Z_1 are independent random variables with a standard normal distribution.

Ao obtermos a variável $Z_0 \sim N(0,1)$, podemos obter $X \sim Lognormal(\mu, \sigma)$ com a operação $X = e^{\mu+\sigma Z_0}$

[169]:
```
def lognormal(media, dp):
        U1 = np.random.uniform(size = 1000)
        U2 = np.random.uniform(size = 1000)
        R = np.sqrt(-2 * np.log(U1))
        theta = 2 * np.pi * U2
        Z0 = R * np.cos(theta)
        Z1 = R * np.sin(theta)
        X = np.exp(media + dp*Z0)
        return X

boxmuller = lognormal(0,1)
numpy = np.random.lognormal(mean=0,sigma=1, size=1000)
```

[170]:
```
fig = go.Figure()
fig.add_trace(go.Histogram(x=boxmuller, name='Lognormal gerada por Box-Muller'))
fig.add_trace(go.Histogram(x=numpy, name='Lognormal gerada pelo Numpy'))

fig.update_layout(barmode='overlay', title=dict(text='Histograma dos valores␣
↪gerados', font=dict(size=30)))
# Reduce opacity to see both histograms
fig.update_traces(opacity=0.75)
fig.show()
```

Como podemos ver, os valores gerados pelo método se aproximam bastante dos valores gerados pelo Numpy.

1.4 Questão 6

6. Sejam U_1 e U_2 variáveis aleatórias *i.i.d.* com distribuição $\mathscr{U}(0,1)$. Sejam $Y_1 = U^{1/\alpha}$ e $Y_2 = U_2^{1/\beta}$. Se $Y_1 + Y_2 \leq 1$. $X = Y_1/(Y_1+Y_2) \sim \mathscr{B}e(\alpha, \beta)$. Implemente este algoritmo.

Implementando o algoritmo:

[171]:
```
def algoritmo(n,a,b):
        beta = []
        erros = []
        while len(beta) < n:
                U1 = np.random.uniform()
                U2 = np.random.uniform()

                Y1 = U1 ** (1/a)
                Y2 = U2 ** (1/b)
                x = Y1/(Y1+Y2)
                if Y1+Y2 <= 1:
```

```
                                x = Y1/(Y1+Y2)
                                beta.append(x)
                        else:
                                erros.append(x)
        return beta
```

Repetindo o algoritmo 1000 vezes e comparando com a função geradora de variáveis beta do numpy:

[172]:
```
a=1
b=3

X = algoritmo(1000, a, b)
beta = np.random.beta(a=a,b=b,size=1000)
```

[173]:
```
fig = go.Figure()
fig.add_trace(go.Histogram(x=X, name='Beta gerada pelo algoritmo'))
fig.add_trace(go.Histogram(x=beta, name='Beta gerada pelo numpy'))

fig.update_layout(barmode='overlay', title=dict(text=f'Histograma dos valores gerados (a={a}, b={b})', font=dict(size=30)))
# Reduce opacity to see both histograms
fig.update_traces(opacity=0.75)
fig.show()
```

1.5 Questão 7

7. Simular de uma distribuição discreta que assume valores $0, 1, 2$ com probabilidades $1/3, 3/4, 5/12$;

Nota: assumindo a segunda probabilidade como 1/4 para que elas possam somar 1.

Simulando com o método da transformação inversa, estou gerando uma variável $U \sim Unif(0,1)$.

De forma que:

$$x = \begin{cases} 0, \; se \;\; U < \frac{1}{3} \\ 1, \; se \;\; \frac{1}{3} \leq U < \frac{1}{3} + \frac{1}{4} \\ 2, \; se \;\; \frac{1}{3} + \frac{1}{4} \leq U < 1 \end{cases}$$

[174]:
```
p = [1/3,1/4,5/12]
cum_p = np.cumsum(p)
X = np.random.uniform(size = 1000)
valores = []
for x in X:
        if x < cum_p[0]:
                valores.append(0)
        elif cum_p[0] <= x < cum_p[1]:
                valores.append(1)
        elif x >= cum_p[1]:
```

```
                valores.append(2)

ideais = [p * 1000 for p in p]
ideais = pd.DataFrame({'nomes': [0,1,2], 'numeros':ideais})
```

[175]:
```
fig = go.Figure()
fig.add_trace(go.Histogram(x=valores, name='Valores gerados pela transformação inversa'))
fig.add_trace(go.Histogram(x=ideais['nomes'], y=ideais['numeros'], name='Valores ideais, dadas as probabilidades', histfunc='sum'))

fig.update_layout(barmode='group', title=dict(text='Histograma dos valores gerados', font=dict(size=30)))
# Reduce opacity to see both histograms
fig.show()
```

Como podemos ver, os valores gerados ficaram em proporções bem próximas das ideais.

1.6 Questão 8:

8. Gere 1000 variáveis aleatórias binomiais para $n = 5$ e $p = 0,3, 0,5, 0,8$. Em cada caso, determine as frequências relativas observadas e as probabilidades teóricas correspondentes. Como está o acordo entre elas? 2

Novamente gerando com o método da transformação inversa, utilizando o truque de gerar n variáveis $Ber(p)$ e depois somá-las para gerar variáveis $Binom(n, p)$.

Ou seja, geramos uma variável $U \sim Unif(0, 1)$ tal que

$$x = \begin{cases} 0, & se \ \ U > p \\ 1, & se \ \ U < p \end{cases}$$

E repetimos o processo n vezes e somamos os resultados para gerar uma $Binom(n, p)$.

[176]:
```
def binomial(n, p):
        binomiais = []
        while len(binomiais) < 1000:
                U = np.random.uniform(size = n)
                x = len([x for x in U if x < p])
                binomiais.append(x)
        return binomiais
```

[177]:
```
x1 = binomial(5,0.3)
x2 = binomial(5,0.5)
x3 = binomial(5,0.8)
```

[178]:
```
x1_freq = pd.Series(x1).value_counts() / len(x1)
x2_freq = pd.Series(x2).value_counts() / len(x2)
x3_freq = pd.Series(x3).value_counts() / len(x3)
```

[179]:
```python
from scipy.stats import binom
x1_teorico = binom(5, 0.3)
x2_teorico = binom(5, 0.5)
x3_teorico = binom(5, 0.8)
```

[180]:
```python
x = [0,1,2,3,4,5]
fig = make_subplots(rows=3, cols=1, shared_xaxes=True,
                        subplot_titles=("p = 0.3","p = 0.5", "p = 0.8"),␣
 ↪vertical_spacing=0.05)
fig.add_trace(go.Bar(y=x1_freq, x=x1_freq.index, name='Frequência gerada',␣
 ↪legendgroup=1), row=1, col=1)
fig.add_trace(go.Scatter(x=x, y=x1_teorico.pmf(x), mode='lines',␣
 ↪name='Frequência teórica', legendgroup=1), row=1, col=1)

fig.add_trace(go.Bar(y=x2_freq, x=x2_freq.index, name='Frequência gerada',␣
 ↪legendgroup=2), row=2, col=1)
fig.add_trace(go.Scatter(x=x, y=x2_teorico.pmf(x), mode='lines',␣
 ↪name='Frequência teórica', legendgroup=2), row=2, col=1)

fig.add_trace(go.Bar(y=x3_freq, x=x3_freq.index, name='Frequência gerada',␣
 ↪legendgroup=3), row=3, col=1)
fig.add_trace(go.Scatter(x=x, y=x3_teorico.pmf(x), mode='lines',␣
 ↪name='Frequência teórica', legendgroup=3), row=3, col=1)

fig.update_layout(barmode='group', height=1000, legend_tracegroupgap=250,␣
 ↪title=dict(text='Histogramas dos valores gerados', font=dict(size=30)))
# Reduce opacity to see both histograms
fig.update_traces(opacity=0.75)
fig.update_yaxes(title_text="<b>Frequência</b>", tickformat=",.2%")
fig.show()
```

Como podemos ver, a frequência das variáveis geradas está bem próxima da frequência teórica para todos os valores de p fornecidos.