1ª Avaliação de Programação Estatística com Python - 10/12/2019

- Nome: _____
  - Matrícula: _____

Questão 1 (1,0 ponto) Implemente uma função que gere C matrizes $\mathbf{X}_c^{n_c \times p}$, $c = 1, \ldots, C$, aleatoriamente, onde n_c é a quantidade de linhas da matriz $\mathbf{X}_c$. As p colunas de cada matriz $\mathbf{X}_c$ devem ser sorteadas de distribuições normais com médias e variâncias diferentes. A função deve então concatenar todas as matrizes geradas e retornar a matriz $\mathbf{X}^{n \times p}$, onde $n = \sum_{c=1}^{C} n_c$. Dica 1: A função pode receber como parâmetros uma lista com a quantidade de linhas de cada matriz $\mathbf{X}_c$, uma lista com as médias das normais usadas para gerar as colunas de cada matriz e outra lista com as variâncias correspondentes. Dica 2: considere usar a função *numpy.random.multivariate_normal*.

In [41]:

```
import numpy as np
def Q01(C, p, n, param):
    np.random.seed(26112000) # fixando semente
    Xc = [] # lista vazia para armazenar as um array de matrizes, gerados aleatoriament
e

    for i in range(C): # Gerando as 'C' Matrizes aleatorias
        # param[0][i] parametro de localização // param[1][i] parametro de escala
        Xc.append(np.random.normal(param[0][i], param[1][i], (n[i],p)))
        # array de matrizes gerados , com parametros de localização, e escala antes inf
ormados como parametros
        # possuindo n[i] linhas, informado como parametro da função

    Xnp = np.vstack((Xc)) # Concatenação das matrizes do arrey gerado pelo for

    return Xnp # Matriz gerada aleatoriamente

# Parametro de locação
mu = [
    [1],
    [0.7],
    [1]
]

# Parametro de escala
covar = [
    [0.9],
    [0.3],
    [0.1]
]

Q1 = Q01(2,2, [4, 8], param = (mu, covar))
print(Q1)
```

```
[[2.17647332 0.43153427]
 [0.75788747 0.2511219 ]
 [0.21238821 2.02135922]
 [0.76892883 1.56120292]
 [0.39857431 0.63624282]
 [0.99232054 0.74462402]
 [0.72958699 0.69084601]
 [0.96664009 0.63697451]
 [0.48694137 0.47475784]
 [0.90117164 1.01453529]
 [0.30150657 0.36292072]
 [0.10985934 0.81263064]]
```

Questão 2 (1,0 ponto) Faça uma função que recebe uma matriz $\mathbf{X}^{n \times p}$ e um inteiro k e retorna uma matriz $W^{k \times p}$, cujas linhas foram selecionadas aleatoriamente e sem reposição de $\mathbf{X}^{n \times p}$.

In [42]:

```
def Q02(Xnp, k):
    np.random.seed(26112000) # fixando semente
    Wkp = [] # Array vazia, para armazenar as linhas da matriz da questão , aleatoriame
nte

    for j in range(k): # for para a seleção das k linhas da matriz Xnp

        Wkp.append(Xnp[np.random.choice(len(Xnp[:, 1]), size = 1, replace = True)])
        # Criação do array com tamanho k, composto de k matrizes geradas aleatoriament
e

    Wkp = np.vstack(Wkp) # Concatenação dos k arrays de matrizes aleatorias

    return Wkp # Matriz k por p, feita por concatenação de matrizes geradas aleatoriame
nte
Q2 = Q02(Q1, 5)

print(Q2)
```

```
[[0.30150657 0.36292072]
 [0.21238821 2.02135922]
 [0.48694137 0.47475784]
 [0.90117164 1.01453529]
 [0.75788747 0.2511219 ]]
```

Questão 3 (1,0 ponto) Faça uma função que recebe duas matrizes $\mathbf{X}^{n \times p}$ e $\mathbf{W}^{k \times p}$ e retorna a matriz de distâncias Euclidianas $\mathbf{D}^{n \times k}$ entre as linhas de $\mathbf{X}$ e as de $\mathbf{W}$.

In [43]:

```
def Q03(Xnp, Wkp):

    Dnk = np.zeros((len(Xnp), len(Wkp))) # Matriz composta de zeros, onde possui len(Xnp) linhas e len(Wkp) colunas

    for i in range(len(Xnp)): # for para fixar a primeira linha de Xnp
        for j in range(len(Wkp)): # for para varrer as linhas de Wkp, fixando a linha i da matriz Xnp
            Dnk[i,j] = np.sqrt(np.sum((Xnp[i,:] - Wkp[j,:])**(2))) # Calculo da distancia euclidiana

    return Dnk # Matriz de distancias de Xnp, para  cada linha de Wkp

Q3 = Q03(Q1, Q2)
print(Q3)
```

```
[[1.87622177 2.52689012 1.69008476 1.4022427  1.43001205]
 [0.46987498 1.85237945 0.35131869 0.77674343 0.        ]
 [1.66083122 0.         1.57078174 1.21988402 1.85237945]
 [1.28622074 0.72213661 1.1224437  0.56243547 1.31012755]
 [0.29004674 1.39757387 0.18408188 0.62905427 0.52671061]
 [0.78925366 1.4961107  0.57291872 0.28488632 0.54635447]
 [0.53924747 1.42750131 0.32491691 0.36635508 0.44063387]
 [0.71938034 1.57652052 0.50638435 0.38319481 0.43870251]
 [0.21654931 1.57078174 0.         0.68040166 0.35131869]
 [0.88555053 1.21988402 0.68040166 0.         0.77674343]
 [0.         1.66083122 0.21654931 0.88555053 0.46987498]
 [0.4888432  1.21306923 0.50630909 0.81666434 0.85745701]]
```

Questão 4 (1,0 ponto) Qual função de NumPy pode ser usada para obter para cada linha da matriz $\mathbf{D}$, retornada pela função da questão acima, a coluna que representa a menor distância da linha correspondente em $\mathbf{X}$ para as linhas de $\mathbf{W}$? Use a função para retornar essa informação.

In [44]:

```
Q4 = np.argmin(Q3, axis = 1) # Atribui 0, caso a menor distancia da linha de X para a de W esteja na primeira coluna
                             # Atribui 1, caso a menor distancia da linha de X para a de W esteja na segunda coluna
                             # ... idem para j colunas
print(Q4)
```

```
[3 4 1 3 2 3 2 3 2 3 0 0]
```

Questão 5 (2,0 pontos) Faça uma função que atualiza as linhas de $\mathbf{W}$, fazendo com que cada linha assuma o valor médio das linhas de $\mathbf{X}$ para as quais a mesma foi a mais próxima. A função deve retornar a matriz $\mathbf{W}$ modificada. Use a função de NumPy mencionada na Questão 4.

In [45]:

```
def Q05(Xnp, Wkp, Dnk, c):
    k = np.argmin(Dnk, axis = 1) # Array com a localizacação das menores distancias da
 linhas
    for i in range(c):
        X = Xnp[k == i,:] # quem são os X que possuem as menores distancia para com W
        Wkp[i,:] = np.apply_along_axis(func1d = np.mean, arr = X, axis = 0) # Wkp esta
 assumindo por linha o valor medio
# das linhas de X  para as quais a mesma foi
# a mais proxima

    return Wkp # Matriz atualizada, com os valores medios das linhas de X mais proximas
de W

Q5 = Q05(Q1, Q2, Q3, 4)

print(Q5)
```

```
[[0.20568296 0.58777568]
 [0.21238821 2.02135922]
 [0.53836756 0.60061555]
 [1.16110688 0.8777742 ]
 [0.75788747 0.2511219 ]]
```

Questão 6 (1,5 pontos) Implemente uma classe, chamada *KMeans*, cujo construtor define os seguintes atributos: k e t_{max}. A classe deve conter um método *fit*, que recebe como parâmetro uma matriz $\mathbf{X}$. O método *fit* irá então seguir os seguintes passos:

1. Use a função implementada na Questão 2 para obter a matriz $\mathbf{W}$ (note que o segundo parâmetro da função receberá como argumento um dos atributos definidos no construtor);
2. Repita as operações abaixo t_{max} vezes (note que isso é um atributo):
   A. Use a função da Questão 3 para calcular as distâncias entre $\mathbf{X}$ e $\mathbf{W}$
   B. Use a função da Questão 5 para atualizar as linhas de $\mathbf{W}$

Ao final do método *fit*, a matriz $\mathbf{W}$ resultante deve ser guardada como um atributo.

In [46]:

```
class KMeans:
    def __init__(self, k, tmax):
        self.k, self.tmax = k, tmax # Atributos k, e tmax, para o uso no Monte Carlo

    def fit(self,X, k): # Função para ajustar o modelo
        W = Q02(X,k) # Criação das matrizes, k por p, de forma aleatoria

        for i in range(self.tmax): # for para a geração do monte carlo, com tmax iteraç
ões
            dists = Q03(X, W) # Calculando as distancias euclidianas, da linha i de X
 para cada linhas W
            new_W = Q05(X, W, dists, k) # W atualizado com as medias das menores distan
cias X , para W

        return new_W # Matriz da ultima iteração do Monte Carlo ,  'treinada'
X = Q1
k = 4
tmax = 100 # 100 iterações de monte carlo
km = KMeans(k, tmax) # A propia classe KMeans

print(km.fit(X, k)) # AJustando o modelo
```

```
[[0.26998007 0.60393139]
 [0.49065852 1.79128107]
 [0.65813861 0.47224191]
 [1.2591514  0.70691702]]
```

Questão 7 (1,0 ponto) Adicione o método *predict* à classe *KMeans*. O método irá receber uma nova matriz $\mathbf{X}$ e irá usar a função de NumPy mencionada na Questão 4 para retornar os índices das linhas de $\mathbf{W}$ mais próximas às linhas de $\mathbf{X}$.

In [47]:

```
class KMeans:
    def __init__(self, k, tmax):
        self.k, self.tmax = k, tmax

    def fit(self,X, k):
        W = Q02(X,k)
        for i in range(self.tmax):
            dists = Q03(X, W)
            new_W = Q05(X, W, dists, k)

        self.W = W

        return new_W

    def predict(self,X):
        dists = Q03(X, self.W) # Calculo das distancias Euclidianas de X linha i, para
 cada linha W
        return np.argmin(dists , axis = 1) # Retorna as menores distancias de X, para u
m Wi, on de Wi são as linhas deW

X = Q1
k = 5
tmax = 100

km = KMeans(k, tmax)

# lista de medias para a geração das matrizes aleatorias
mu1 = [
    [2],
    [1.444]
]

# lista de variancias para a geração das matrizes aleatorias
covar1 = [
    [0.4625],
    [0.9584]
]
X1 = Q01(2,2,[15,30], param = (mu1, covar1)) # Geração da matriz Aleatorias

print(km.fit(X,k)) # AJustando o modelo
print()
print(km.predict(X1)) # Retornando os indices das menores distancias de X
```

```
[[0.20568296 0.58777568]
 [0.49065852 1.79128107]
 [0.53836756 0.60061555]
 [1.2591514  0.70691702]
 [0.75788747 0.2511219 ]]

[3 3 1 1 1 3 3 3 3 1 3 1 3 1 1 1 2 2 3 3 3 3 4 3 1 3 3 3 3 3 2 4 3 2 3 3 3
 3 0 3 3 4 1 3 4]
```

Questão 8 (1,5 ponto) Adicione o método *score* à classe *KMeans*. O método irá receber uma nova matriz $\mathbf{X}$ e irá usar a função da Questão 3 para calcular as distâncias para a matrix $\mathbf{W}$. Após isso, o método retornará o somatório das menores distâncias.

In [48]:

```
class KMeans:
    def __init__(self, k, tmax):
        self.k, self.tmax = k, tmax

    def fit(self,X, k):
        W = Q02(X,k)
        for i in range(self.tmax):
            dists = Q03(X, W)
            new_W = Q05(X, W, dists, k)

        self.W = W

        return new_W

    def predict(self,X):
        dists = Q03(X, self.W)
        return np.argmin(dists , axis = 1)

    def score(self, X):
        d = Q03(X, self.W) # Calculo da matriz de distancias euclidianas
        d = np.apply_along_axis(min, 1, d) # atualizando a matriz das distancias, aplic
ando a função min, para achar as                                        # minimas
 distancias, por linha, na matriz distancias, utilizando a função
# apply_along_axis()
        return np.sum(d)
X = Q1
k = 5
tmax = 100
km = KMeans(k, tmax)

mu2 = [
    [0],
    [0]
]

covar2 = [
    [0.7457],
    [0.11515]
]
X1 = Q01(2,2,[20,4], param = (mu1, covar1))
X2 = Q01(2,2,[17, 48], param = (mu2, covar2))
km.fit(X,k)
print(km.fit(X,k))
print()
print(km.predict(X1))
print()
print(km.score(X2)) # Somatorio de todas as menores distancias da matriz X2, para a mat
riz W
```

```
[[0.20568296 0.58777568]
 [0.49065852 1.79128107]
 [0.53836756 0.60061555]
 [1.2591514  0.70691702]
 [0.75788747 0.2511219 ]]

[3 3 1 1 1 3 3 3 3 1 3 1 3 1 1 1 1 3 3 3 3 3 4 3]

45.35684922351373
```