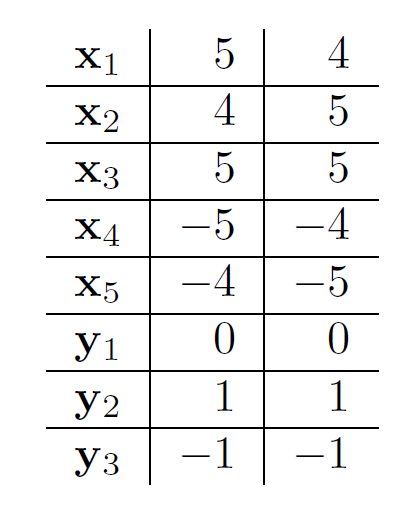
**CPE-723 – Otimização Natural**

**Lista de Exercícios #3**

**Amanda Isabela de Campos (DRE 120074842)**

01) Prova de 2012 - Questão 4;

(*Deterministic Annealing*) Considere um problema de *soft clustering* com cinco vetores de dados x = (*x*1, *x*2) e três centroides iniciais y = (*y*1*, y*2), definidos segundo a tabela a seguir. Considere também que a distância entre dois vetores é quadrática, ou seja, *d*(x,y) = (*x*1 - *y*1)2 + (*x*2 *- y*2)2.



a) Calcule a matriz de probabilidades *p*(y*|*x) que minimiza *J* = *D - TH* com *T* = 10. Calcule também os valores dos centróides atualizados segundo esta matriz.

import numpy as np

X = np.array([[5,4,5,-5,-4],[4,5,5,-4,-5]])

M,N=np.shape(X)

K=2 #número de clusters

Y=np.array([[0,1,-1],[0,1,-1]])

d=np.zeros([K,N])

T = 10

p\_ygivenx=np.zeros([K,N])

# Condição da Partição

for n in range(0,N):

    for k in range(0,K):

        d[k,n]=np.sum(np.power(X[:,n]-Y[:,k],2))

        p\_ygivenx[k,n]=np.exp(-d[k,n]/T)

Zx=np.sum(p\_ygivenx,axis=0)

p\_ygivenx=p\_ygivenx/np.tile(Zx,(K,1))

# Condição do centróide

Y=np.zeros([M,K])

for k in range(0,K):

    y=np.zeros(M)

    w=0

    for n in range(0,N):

        y+=p\_ygivenx[k,n]\*X[:,n]

        w+=p\_ygivenx[k,n]

    Y[:,k]=y/w

py|x = 0.164248 0.164248 0.139656 0.164248 0.164248

0.813524 0.813524 0.84487 0.0222285 0.0222285

0.0222285 0.0222285 0.0154743 0.813524 0.813524

Y = 0.876524 4.50887 -4.17568

0.876524 4.50887 -4.17568

b) Repita o item (a) para *T* = 0*.*1 e comente sobre qual é a diferença.

py|x = 3.25749e-70 3.25749e-70 6.71418e-79 3.25749e-70 3.25749e-70

1 1 1 4.50803e-157 4.50803e-157

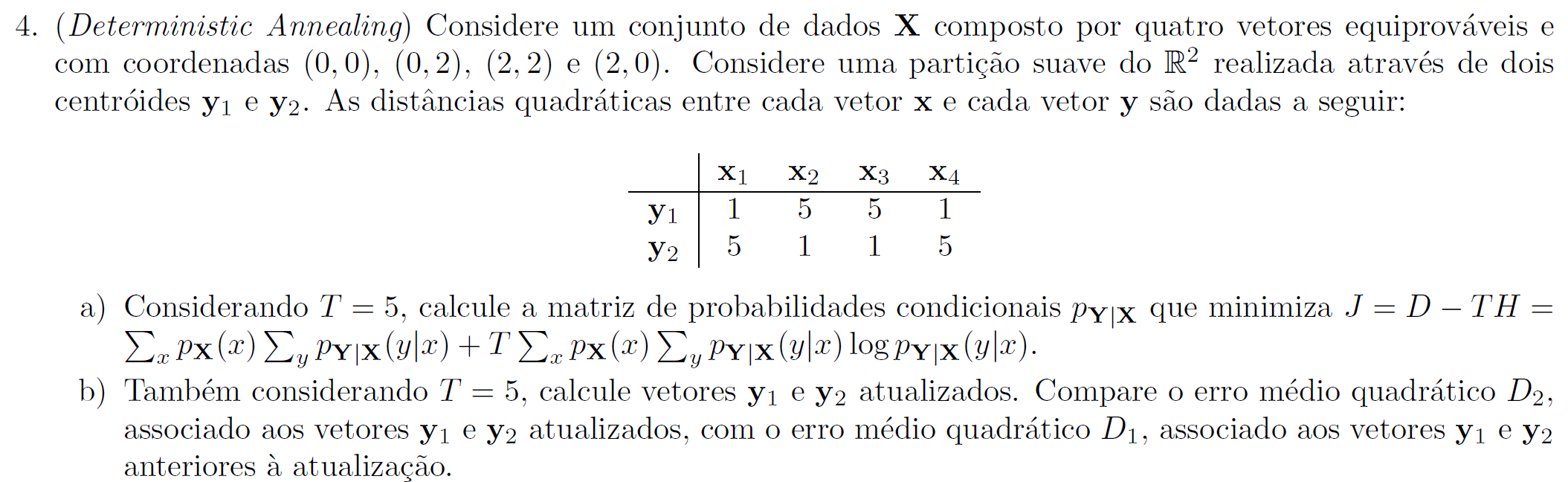
4.50803e-157 4.50803e-157 1.91517e-174 1 1

Y = 2.57644e-09 4.66667 -4.5

2.57644e-09 4.66667 -4.5

Neste caso, para uma temperatura menor a matriz de probabilidades py|x tem em cada linha praticamente todos os valores iguais a zero e apenas um valor igual a um, o que indica um probabilidade de acerto igual a 100%, ou seja o processo convergiu.

**02) Prova de 2017 - Questão 4;**



a) py|x = 0.689974 0.310026 0.310026 0.689974

0.310026 0.689974 0.689974 0.310026

Calculos obtidos com o seguinte código:

X = np.array([[0,0,2,2],[0,2,2,0]])

M,N=np.shape(X)

K=2 #número de clusters

d=np.zeros([K,N])

T = 5

p\_ygivenx=np.zeros([K,N])

d = np.array([[1,5,5,1],[5,1,1,5]])

# Partition Condition

for n in range(0,N):

    for k in range(0,K):

        # d[k,n]=np.sum(np.power(X[:,n]-Y[:,k],2))

        p\_ygivenx[k,n]=np.exp(-d[k,n]/T)

Zx=np.sum(p\_ygivenx,axis=0)

p\_ygivenx=p\_ygivenx/np.tile(Zx,(K,1))

J=-T/N\*np.sum(np.log(Zx))

D=np.mean(np.sum(p\_ygivenx\*d,axis=0))

b) Y = 1 1

0.620051 1.37995

O erro médio quadrático D2 é 2.240102 (atualizado)

Com as coordenadas dos pontos e as distâncias quadráticas é possível calcular as coordenadas dos dois centróides y1 e y2 : y1 (1,0) e y2 (1,-2), portanto pode-se calcular o erro médio quadrático.

O erro médio quadrático D1 é 4.76766 (anterior). O erro médio quadrático diminuiu depois da atualização, isso porque esse processo de atualização é um passo do algoritmo de Deterministic Annealing em busca do clusters com mínimo erro médio quadrático.

O código adotado está reproduzido a seguir.

X = np.array([[0,0,2,2],[0,2,2,0]])

M,N=np.shape(X)

K = 2 #número de clusters

Y=np.array([[0,1],[1,-2]])

d=np.zeros([K,N])

T = 10

p\_ygivenx=np.zeros([K,N])

# Partition Condition

for n in range(0,N):

    for k in range(0,K):

        d[k,n]=np.sum(np.power(X[:,n]-Y[:,k],2))

        p\_ygivenx[k,n]=np.exp(-d[k,n]/T)

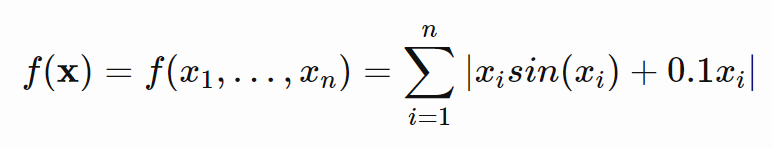
Zx=np.sum(p\_ygivenx,axis=0)

p\_ygivenx=p\_ygivenx/np.tile(Zx,(K,1))

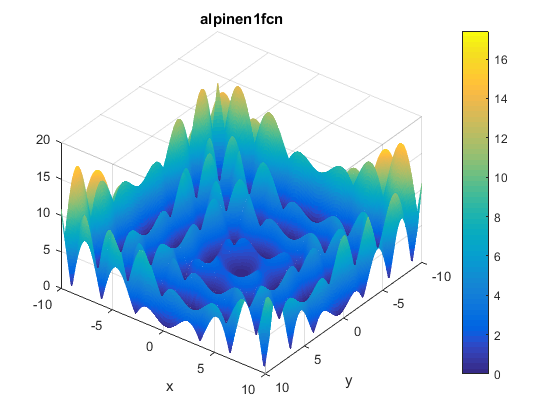
J=-T/N\*np.sum(np.log(Zx))

D=np.mean(np.sum(p\_ygivenx\*d,axis=0))

03) Proponha uma função J (x), sendo x um vetor com 20 dimensões, cujo ponto mínimo você conheça. Evite propor funções que tenham um só ponto mínimo. Encontre o ponto mínimo global utilizando S.A. Entregue o código utilizado e alguns comentários sobre o resultado obtido;

A função adotada como exemplo é a Alpine N. 1 Function definida no espaço n-dimensional. A função tem mínimo global f(x∗) = 0 localizado em x∗=(0,…,0) e segue a expressão: 

O gráfico da função Alpine em duas dimensões está indicado a seguir:



Referencias:

<http://benchmarkfcns.xyz/benchmarkfcns/alpinen1fcn.html>

O código implementado para a busca do mínimo global desta função com o Simulated Annealing e o Fast Simulated Annealing está reproduzido a seguir. Vale observar que em ambos os casos o mínimo global foi encontrado, porem com o FSA a convergência foi mais rápida.

import numpy as np

import math

numero\_variaveis = 20

b = 10.0  ## Limite superior

a = -10.0  ## Limite inferior    limites = [a,b)

def Custo(X):

    numero\_variaveis = 20

    d = numero\_variaveis

    sum = 0;

    for ii in range(0,d):

        xi = X[ii]

        sum = sum + (np.abs(xi \* math.sin(xi) + 0.1 \* xi))

    y = sum

    return y

#------------------------------------------------------------------------------

# Simulated Annealing

X0 = (b-a)\*np.random.rand(numero\_variaveis) + a

N=int(1e5);  K=100; T0=5e-1; e=1e-1

X = X0

Xmin = X0

np.random.seed(0);

fim=0; n=0; k=0; Jmin=Custo(X); Xmin=X; T=T0;

while not(fim):

    T = T0/(1+k)

    # T = T0/np.log2(2+k)

    for n in range(N):

        X\_hat = X + e\*(np.random.standard\_cauchy(np.shape(X)))

        # X\_hat = X + e\*(np.random.normal(0,1,np.shape(X)))

        if np.random.uniform()<np.exp((Custo(X)-Custo(X\_hat))/T):

            X = X\_hat

            if Custo(X) < Jmin:

                Jmin = Custo(X)

                Xmin = X

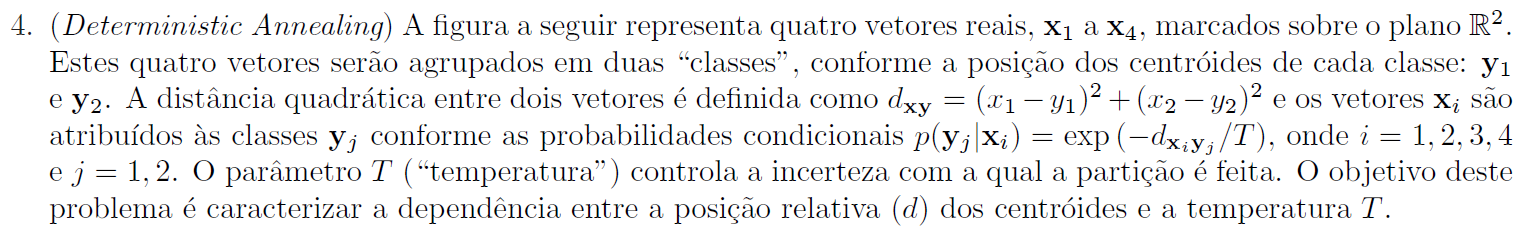
    print([k,Jmin])

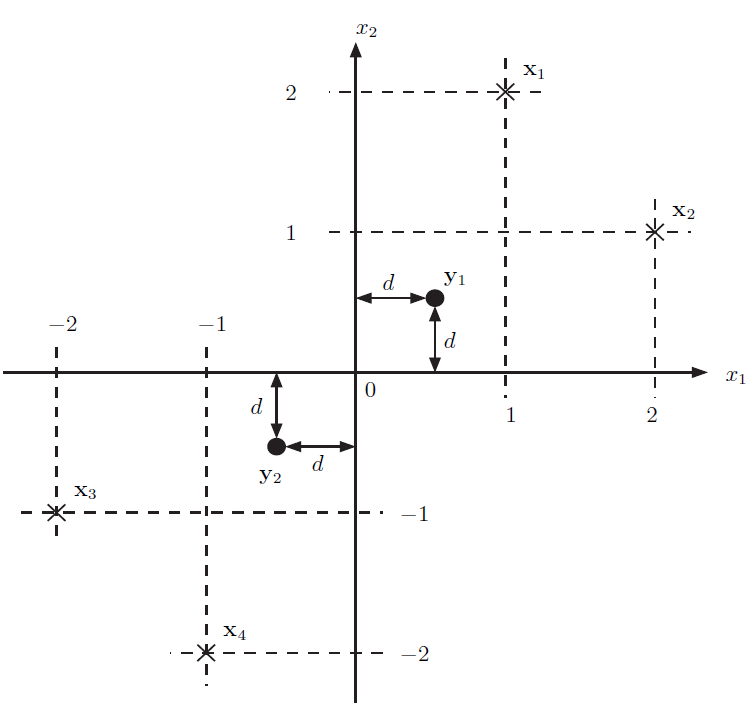
    k=k+1

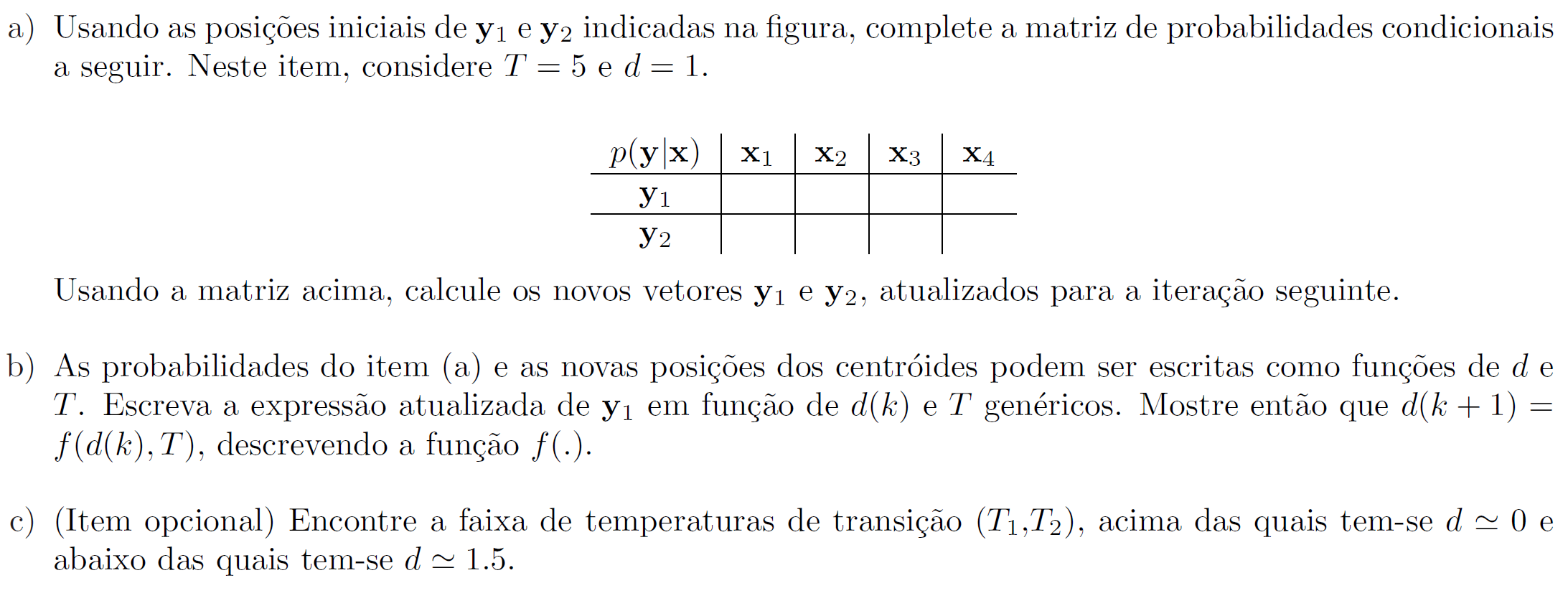
    if k == K: fim =1

print(Jmin)

04) Questão extra (opcional): Prova de 2009 - Questão 4.



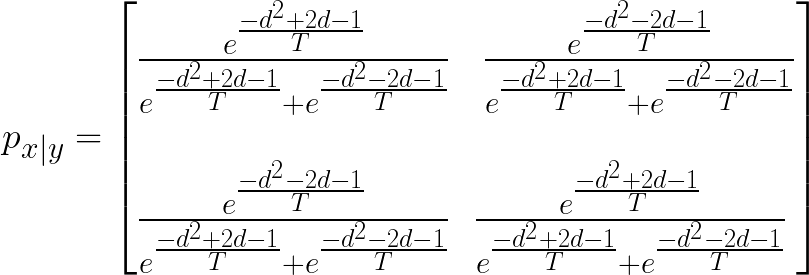
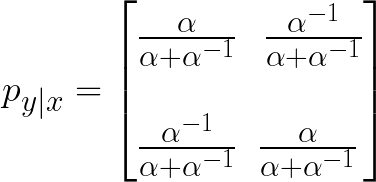




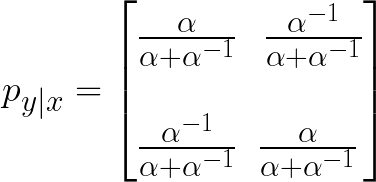
a)

https://latex.codecogs.com/png.latex?d_%7Bxy%7D%3D%5Cleft%20%5B%20%5Cbegin%7Bmatrix%7D%20d%5E%7B2%7D-2d&plus;1%20%26%20d%5E%7B2%7D&plus;2d&plus;1%5C%5C%20d%5E%7B2%7D&plus;2d&plus;1%20%26%20d%5E%7B2%7D-2d&plus;1%20%5Cend%7Bmatrix%7D%20%5Cright%20%5D

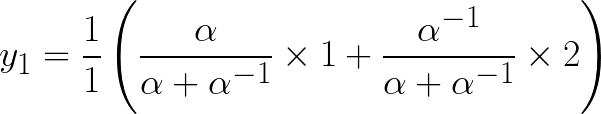
Partição:

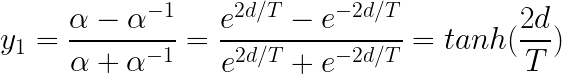


Substituindo https://latex.codecogs.com/png.latex?%5Clarge%20e%5E%7B%5Cfrac%7B2d%7D%7BT%7D%7D por em px|y, temos:



b) Condição da partição: novo y1 (sendo que y2 = - y1)





c) Atualizando d(n), temos: d(n+1) = tanh(2d(n)/T). Numericamente:

