**CPE-723 – Otimização Natural**

**Lista de Exercícios #2**

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

1. **Considere um processo de Markov *X*(*t*) que tem trˆes estados poss´ıveis: 0, 1, e 2. A evolu¸c˜ao temporal deste processo ´e dada pela matriz de transi¸c˜ao a seguir:**
   1. **Considerando que a distribui¸c˜ao de probabilidade de *X*(0) ´e dada pelo vetor p0 = [0*.*3 0*.*4 0*.*3]*T*, calcule a distribui¸c˜ao de probabilidade de *X*(3) (ou seja, do processo de Markov no instante *t* = 3).**

import numpy as np

M = np.array([[0.5,0.25,0.25],[0.25,0.50,0.25], [0.25, 0.25, 0.50]])

p0 = np.array([[0.3],[0.4], [0.3]])

M2 = M.dot(M)

M3 = M2.dot(M)

print(M3.dot(p0))

[[0.3328125], [0.334375 ], [0.3328125]]

* 1. **Iniciando em X(0) = 1, e usando um gerador de números aleatórios (são necessários apenas três números aleatórios, sorteados de PDF uniforme entre 0 e 1), calcule manualmente uma amostra do processo *X*(*t*) at´e *t* = 3.**

a = np.random.uniform(0.00,0.25,1)

b = np.random.uniform(0.25,0.75,1)

c = np.random.uniform(0.75,1.00,1)

p0 = np.array([a,b,c])

M = np.array([[0.5,0.25,0.25],[0.25,0.50,0.25], [0.25, 0.25, 0.50]])

M2 = M.dot(M)

M3 = M2.dot(M)

print(M3.dot(p0))

[[0.5860114 ], [0.59551058], [0.59795439]]

* 1. **Usando um computador, execute 100 repeti¸c˜oes do item (b). Em cada uma das 100 repeti¸c˜oes, comece a simula¸c˜ao com um valor diferente de *X*(0), assumindo que os eventos *X*(0) = 0, *X*(0) = 1, e *X*(0) = 2 s˜ao equiprovaveis. Armazene as 100 cadeias obtidas em uma matriz X, com 4 colunas (*t* = 0 at´e *t* = 3) e 100 linhas.**

M = np.array([[0.5,0.25,0.25],[0.25,0.50,0.25], [0.25, 0.25, 0.50]])

M2 = M.dot(M)

M3 = M2.dot(M)

C = np.zeros((100,4))

for i in range(100):

    prob = np.random.choice([0,1,2])

    if prob == 0:

        a = np.random.uniform(0.00,0.50,1)

        b = np.random.uniform(0.50,0.75,1)

        c = np.random.uniform(0.75,1.00,1)

        p0 = np.array([a,b,c])

    if prob == 1:

        a = np.random.uniform(0.00,0.25,1)

        b = np.random.uniform(0.25,0.75,1)

        c = np.random.uniform(0.75,1.00,1)

        p0 = np.array([a,b,c])

    if prob == 2:

        a = np.random.uniform(0.00,0.25,1)

        b = np.random.uniform(0.25,0.50,1)

        c = np.random.uniform(0.50,1.00,1)

        p0 = np.array([a,b,c])

    C[i,0] = np.linalg.norm(p0)

    C[i,1] = np.linalg.norm(M.dot(p0))

    C[i,2] = np.linalg.norm(M2.dot(p0))

    C[i,3] = np.linalg.norm(M3.dot(p0))

* 1. **Fazendo histogramas de cada uma das 4 colunas, calcule as distribui¸c˜oes de probabilidade do processo *X*(*t*) em cada um dos 4 instantes: *t* = 0, 1, 2, 3. Comente os resultados obtidos.**

import matplotlib.pyplot as plt

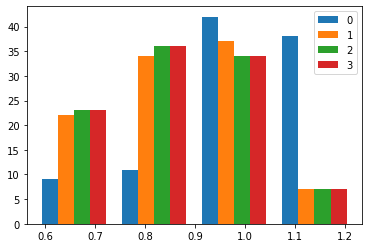
plt.figure(1)

plt.hist(C, 4)

plt.legend('0123')

plt.show()

Observa-se que em todos instantes de tempo o histograma indica que a maior probabilidade está perto do estado 1. E com a diminuição da temperatura esse comportamento é acentuado.

****

1. **Considere um sistema em que s´o h´a 5 estados poss´ıveis: *x* = 1, *x* = 2, *x* = 3, *x* = 4, *x* = 5. Os custos *J* (*x*) de cada um dos estados s˜ao indicados na tabela abaixo:**

|  |  |
| --- | --- |
| ***x*** | ***J* (*x*)** |
| **1** | **0*.*5** |
| **2** | **0*.*2** |
| **3** | **0*.*3** |
| **4** | **0*.*1** |
| **5** | **0*.*4** |

* 1. **Considere um processo de Markov gerado pela aplica¸c˜ao do algoritmo de Metropolis aos dados da tabela acima, com temperatura fixa *T* = 0*.*1. Calcule a matriz de transi¸c˜ao *M* que define o processo *X*(*t*). Obs.: note que o estado *X*(*t*) ´e unidimensional, e portanto a matriz *M* ´e 5 *×* 5.**

A montagem da matriz de transição M segue a seguinte regra: ΔJ < 0, o estado é aceito com uma probabilidade de ¼, caso contrário, a probabilidade de ser aceito é calculada.

(prob. de sortear 1 a partir de 2) x (prob. de aceitar 1 a partir de 2) =

import numpy as np

M = np.zeros([5,5])

M[0,0] = 0

M[1,0] = 1/4

M[2,0] = 1/4

M[3,0] = 1/4

M[4,0] = 1/4

M[0,1] = 1/4\*np.exp(-3)

M[1,1] = 1/4\*(1-np.exp(-3)) + 1/4\*(1-np.exp(-1)) + 1/4\*(1-np.exp(-2))

M[2,1] = 1/4\*np.exp(-1)

M[3,1] = 1/4

M[4,1] = 1/4\*np.exp(-2)

M[0,2] = 1/4\*np.exp(-2)

M[1,2] = 1/4

M[2,2] = 1/4\*(1-np.exp(-2)) + 1/4\*(1-np.exp(-1))

M[3,2] = 1/4

M[4,2] = 1/4\*np.exp(-1)

M[0,3] = 1/4\*np.exp(-4)

M[1,3] = 1/4\*np.exp(-1)

M[2,3] = 1/4\*np.exp(-2)

M[3,3] = 1/4\*(1-np.exp(-4)) + 1/4\*(1-np.exp(-1)) + 1/4\*(1-np.exp(-2)) + 1/4\*(1-np.exp(-3))

M[4,3] = 1/4\*np.exp(-3)

M[0,4] = 1/4\*np.exp(-1)

M[1,4] = 1/4

M[2,4] = 1/4

M[3,4] = 1/4

M[4,4] = 1/4\*(1-np.exp(-1))

* 1. **Iniciando em *X*(0) = 1, calcule manualmente 4 amostras do processo *X*(*t*).**

p0 = np.array([[np.exp(0.5)],[np.exp(0.2)], [np.exp(0.3)], [np.exp(0.1)], [np.exp(0.4)]])

M2 = M.dot(M)

M3 = M2.dot(M)

M4 = M3.dot(M)

print(M4.dot(p0))

p0 = [1.64872, 1.2214, 1.34986, 1.10517, 1.49182]

p4 = [[0.09749708] [1.83688161] [0.71882263] [3.89896649][0.26481064]]

* 1. **Qual ´e o vetor invariante da matriz *M* do item (a) ?**

**Obs.: para facilitar os c´alculos, pode-se usar o computador neste item.**

autovals, autovecs = np.linalg.eig(M)

print ("Autovetores de A: \n", autovecs[:,0])

print ("Autovalores de A: \n",autovals)

inv = np.zeros(5)

z = np.sum(autovecs[:,0])

for i in range(0,5):

    print(i)

    inv[i] =  autovecs[i,0] / z

print('Vetor inveriante da matriz M: ' + str(inv) )

Vetor inveriante da matriz M: [0.01165623 0.23412166 0.08612854 0.63640865 0.03168492]

* 1. **Calcule os fatores de Boltzmann (ou seja, *e−*(*J*(*x*))*/T* ) associados aos dados da tabela acima, e compare-os com o resultado do item (c). Use *T* = 0*.*1.**

|  |  |  |  |
| --- | --- | --- | --- |
| *x* | *J* (*x*) | *e-(J(x))/T* | *e-(J(x))/T/∑ e-(J(x))/T* |
| 1 | 0*.*5 | 0.00673795 | 0.011656231 |
| 2 | 0*.*2 | 0.13533528 | 0.234121657 |
| 3 | 0*.*3 | 0.04978707 | 0.086128544 |
| 4 | 0*.*1 | 0.367879441 | 0.636408646 |
| 5 | 0*.*4 | 0.018315639 | 0.031684921 |

Observa-se que os valores da ultima coluna da tabela anterior correspondem aos mesmo valores da vetor invariante da matriz M calculados no item c.

* 1. ***Simulated Annealing:* Usando um computador, execute 1000 itera¸c˜oes do algoritmo de Metropolis em cada uma das 10 temperaturas a seguir. Na passagem de uma temperatura para a outra, use o estado atual. Comente as distribui¸c˜oes de probabilidade obtidas no final de cada temperatura.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***T*0** | ***T*1** | ***T*2** | ***T*3** | ***T*4** | ***T*5** | ***T*6** | ***T*7** | ***T*8** | ***T*9** |
| **0*.*1000** | **0*.*0631** | **0*.*0500** | **0*.*0431** | **0*.*0387** | **0*.*0356** | **0*.*0333** | **0*.*0315** | **0*.*0301** | **0*.*0289** |

import numpy as np

import matplotlib.pyplot as plt

##

def J(x):

    if x == 1:

        return 0.5

    if x == 2:

        return 0.2

    if x == 3:

        return 0.3

    if x == 4:

        return 0.1

    if x == 5:

        return 0.4

N = 1000

M = 0.1\*N

Temp = np.zeros(10)

Temp[0] = 0.1; Temp[1] = 0.0631; Temp[2] = 0.05; Temp[3] = 0.0431;Temp[4] = 0.0387

Temp[5] = 0.0356; Temp[6] = 0.0333; Temp[7] = 0.0315; Temp[8] = 0.0301; Temp[9] = 0.0289

x = np.zeros(N)

n = 0

x[n] = np.random.choice([1,2,3,4,5])

i = 0; fim = 0

k = 1; K=11

while not(fim):

    T = Temp[i]

    i += 1

    for n in range(1,N):

        perturbacao = np.random.choice([-1,1])

        x\_hat = x[n-1] + perturbacao

        if x\_hat == 6:

            x\_hat = 1

        elif x\_hat == 0:

            x\_hat= 5

        if np.random.uniform(0,1)< np.exp((J(x[n-1]) - J(x\_hat))/T):

            x[n] = x\_hat

        else:

            x[n] = x[n-1]

    y = np.linspace(0,6,1000)

    plt.figure(1)

    plt.hist(x, 5)

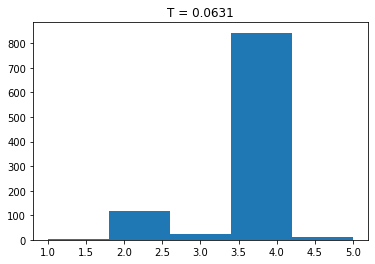
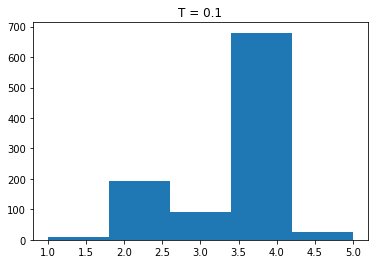
    plt.title('T = ' + str(T))

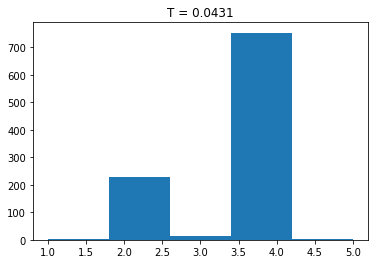
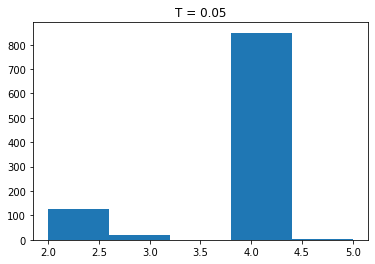
    plt.show()

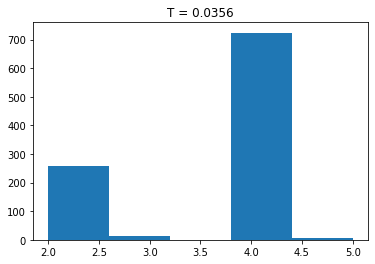
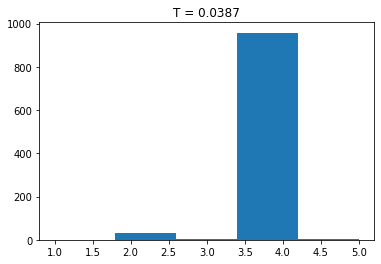
    k+=1

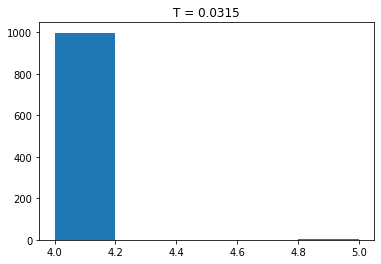
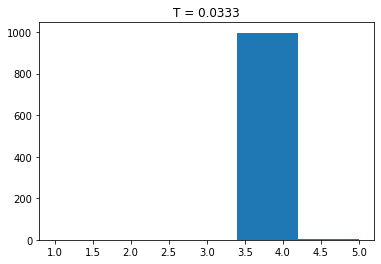
    if k==K: fim=1

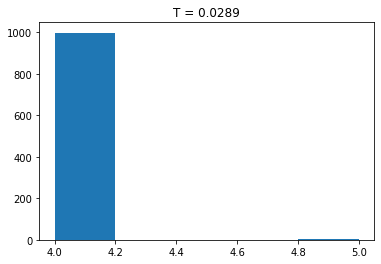
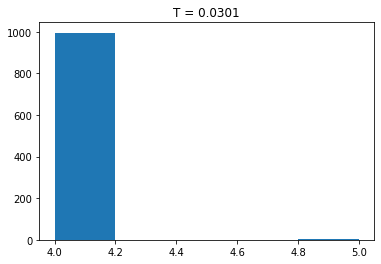
Nos histogramas a seguir estão representadas as distribuições de probabilidade em cada temperatura, e com a diminuição da temperatura observa-se que o algoritmo converge para o ponto de menor custo (em x=4) ou seja, esse procedimento de diminuir a temperatura gradativamente em um algoritmo de Metropolis, consiste em criar um método de otimização por Simulated Annealing.

****

****

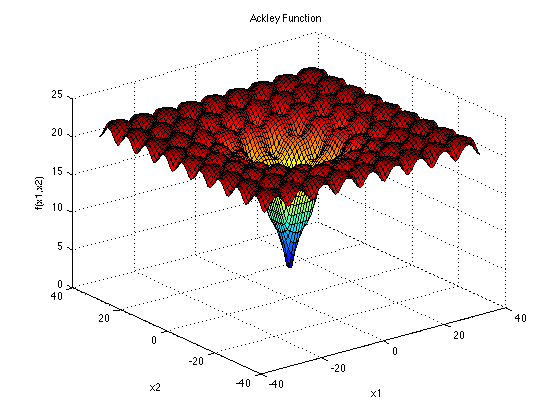
****

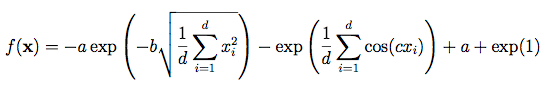
****

****

1. **Proponha uma fun¸c˜ao *J* (x), sendo x um vetor com 10 dimens˜oes, cujo ponto m´ınimo vocˆe conhe¸ca. Evite propor fun¸c˜oes que tenham um s´o ponto m´ınimo. Encontre o ponto m´ınimo global utilizando S.A. Obs.: neste exerc´ıcio, entregue o c´odigo utilizado e alguns coment´arios sobre o resultado obtido.**

A função escolhida para esse problema foi a ACKLEY FUNCTION (Referência: <https://www.sfu.ca/~ssurjano/ackley.html>) que segue a expressão a seguir e nesse caso, será adotado d = 10 para representar uma função de 10 dimensões.





Com valores recomendados para as constantes como: a = 20, b = 0.2 e c = 2π. Tem-se que o ponto de mínimo global é: f(x) = 0 em x = (0,...,0). A seguir está o código adotando o Simulated Annealing para a busca desse ponto de mínimo global.

import numpy as np

numero\_variaveis = 10

b = 32.768  ## Limite superior

a = -32.768 ## Limite inferior para os limites = [a,b)

def Custo(X): # ACKLEY FUNCTION

    numero\_variaveis = 10

    # transformar matriz 2\*10 em vetor de 20 dimensões

    A = np.zeros(2\*numero\_variaveis)

    for i in range(0,numero\_variaveis):

        A[i] = X[0,i]

    for i in range(0,numero\_variaveis):

        A[i+numero\_variaveis] = X[1,i]

    c = 2\*np.pi

    b = 0.2

    a = 20

    d = 2\*numero\_variaveis

    sum1 = 0;

    sum2 = 0;

    for ii in range(0,d):

        xi = A[ii]

        sum1 = sum1 + xi\*\*2

        sum2 = sum2 + np.cos(c\*xi)

    term1 = -a \* np.exp(-b\*np.sqrt(sum1/d))

    term2 = -np.exp(sum2/d)

    y = term1 + term2 + a + np.exp(1)

    return y

# Simulated Annealing

X0 = (b-a)\*np.random.rand(2,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/np.log2(2+k)

    for n in range(N):

        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)

1. **Prova de 2009 - Questão 2, itens (a) e (c).**

**(*Simulated Annealing*) Considere um problema de otimização representado pela função custo a seguir:**

|  |  |
| --- | --- |
| **x** | **J(x)** |
| **1** | **0.3** |
| **2** | **0.1** |
| **3** | **0.1** |
| **4** | **0.2** |

**a) Calcule os fatores de Boltzmann *e-J(x)/T*, para *T* = 1.0 e para *T* = 0*.*1.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | J(x) | | ***e-J*(x)*/T***  ***T=1.0*** | ***e-J*(x)*/T***  ***T=0.1*** |
| 1 | | 0.3 | 0.7408182 | 0.049787 |
| 2 | | 0.1 | 0.9048374 | 0.367879 |
| 3 | | 0.1 | 0.9048374 | 0.367879 |
| 4 | | 0.2 | 0.8187307 | 0.135335 |

**c) Calcule as matrizes de transição M para *T* = 1*.*0 e para *T* = 0*.*1. Calcule os vetores invariantes destas matrizes e compare-os com os resultados do item (a).**

* Para T = 1.0

O vetor invariante de M é ([[0.21987801, 0.26855961, 0.26855961, 0.24300278]])

* Para T = 0.1

O vetor invariante de M neste caso é ([[0.05406459, 0.3994863 , 0.3994863 , 0.1469628 ]])

A comparação entre os resultados do item *a* normalizados (***e-J*(x)*/T/∑* )** com os valores do vetor invariante estão a seguir, onde observa-se que correspondem ao mesmo resultado.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | J(x) | | ***e-J*(x)*/T***  ***T=1.0*** | ***e-J*(x)*/T/∑***  ***T=1.0*** | ***e-J*(x)*/T***  ***T=0.1*** | ***e-J*(x)*/T/∑***  ***T=0.1*** |
| 1 | | 0.3 | 0.7408182 | 0.219878 | 0.049787 | 0.054064 |
| 2 | | 0.1 | 0.9048374 | 0.268560 | 0.367879 | 0.399486 |
| 3 | | 0.1 | 0.9048374 | 0.268560 | 0.367879 | 0.399486 |
| 4 | | 0.2 | 0.8187307 | 0.243003 | 0.135335 | 0.146963 |
| ∑ | |  | 3.3692238 |  | 0.920881 |  |

Os vetores invariantes das matrizes foram calculados com o código a seguir:

import numpy as np

T= 0.1

M[0,0] = 0

M[1,0] = 1/3

M[2,0] = 1/3

M[3,0] = 1/3

M[0,1] = 1/3\*np.exp(-0.2/T)

M[1,1] = 1/3\*(1-np.exp(-0.2/T)) + 1/3\*(1-np.exp(-0.1/T))

M[2,1] = 1/3

M[3,1] = 1/3\*np.exp(-0.1/T)

M[0,2] = 1/3\*np.exp(-0.2/T)

M[1,2] = 1/3

M[2,2] = 1/3\*(1-np.exp(-0.2/T)) + 1/3\*(1-np.exp(-0.1/T))

M[3,2] = 1/3\*np.exp(-0.1/T)

M[0,3] = 1/3\*np.exp(-0.1/T)

M[1,3] = 1/3

M[2,3] = 1/3

M[3,3] = 1/3\*(1-np.exp(-0.1/T))

autovals, autovecs = np.linalg.eig(M)

print ("Autovetores de A: \n", autovecs[:,0])

print ("Autovalores de A: \n",autovals)

inv = np.zeros(4)

z = np.sum(autovecs[:,0])

for i in range(0,4):

    print(i)

    inv[i] =  autovecs[i,0] / z

print('Vetor inveriante da matriz M: ' + str(inv) )

1. **Prova de 2011 - Questão 2, itens (a), (b), e (e).**

**(*Simulated Annealing*) Considere a função custo *J*(*x*1*, x*2) definida pela tabela a seguir:**

|  |  |  |
| --- | --- | --- |
| ***x*1** | ***x*2** | ***J*(*x*)** |
| **0** | **0** | **0.2** |
| **0** | **1** | **0.3** |
| **1** | **0** | **0.3** |
| **1** | **1** | **0.1** |

**a) A aplicação do Algoritmo de Metropolis a um vetor inicial x(0) qualquer, alterando uma componente (*x*1 ou *x*2) de cada vez, define um processo de Markov com duas matrizes de transição: M1 e M2. Calcule estas matrizes de transição, considerando *T* = 0.5. Note que o número de estados possíveis é 4.**

**b) Calcule, para temperatura *T* = 0*.*5, a distribuição de Boltzmann/Gibbs do vetor aleatório *X*. Verifique que esta distribuição de probabilidades define um vetor invariante para ambas as matrizes de transição calculadas no item (a).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | J(x) | | ***e-J*(x)*/T***  ***T = 0.5*** | ***e-J*(x)*/T/∑*** |
| 00 | | 0.2 | 0.6703200 | 0.2591436 |
| 01 | | 0.3 | 0.5488116 | 0.2121688 |
| 10 | | 0.3 | 0.5488116 | 0.2121688 |
| 11 | | 0.1 | 0.8187307 | 0.3165187 |
| ∑ | |  | 2.5866741 |  |

Existem dois autovalores unitários, dessa forma dois vetores invariantes para as matrizes M1 e M2 são calculados, de modo que:

Vetor invariante da matriz M1: ([[0.549834 , 0. , 0.450166 , 0. ],

[0. , 0.40131234, 0. , 0.59868766]])

Vetor invariante da matriz M2: ([[0.549834 , 0.450166 , 0. , 0. ],

[0. , 0. , 0.40131234, 0.59868766]])

Ocorreu que neste caso a distribuição de probabilidades não definiu um vetor invariante para as matrizes de transição.

**e) Quando um número suficientemente grande de iterações do algoritmo do item (c) tiver sido calculado à temperatura *T* = 0*.*1, com que probabilidade teremos a ocorrência do evento *J* = 0*.*3?**

1. **(Opcional/Desafio) Prova de 2012 - Questão 3.**
2. **Prova de 2016 - Questão 2.**

**(*Algoritmo de Metropolis*) Considere uma execu¸c˜ao do algoritmo de Metropolis `a temperatura fixa *T* = 1, com estados [*X*1*X*2] (*X*1 e *X*2 s˜ao vari´aveis aleat´orias bin´arias) e as duas matrizes de transi¸c˜ao dadas a seguir. A matriz M1, `a esquerda, modela as probabilidades de transi¸c˜ao entre estados no caso em que a perturba¸c˜ao, sempre diferente de zero, ´e feita sobre *X*1. A matriz M2, `a direita, ´e para o caso em que a perturba¸c˜ao, sempre diferente de zero, ´e feita sobre *X*2.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **M1** | **00** | **01** | **11** | **10** | **M2** | **00** | **01** | **11** | **10** |
| **00** | **2*/*3** | **0** | **0** | **1** | **00** | **2*/*3** | **1** | **0** | **0** |
| **01** | **0** | **2*/*3** | **1** | **0** | **01** | **1*/*3** | **0** | **0** | **0** |
| **11** | **0** | **1*/*3** | **0** | **0** | **11** | **0** | **0** | **0** | **1*/*3** |
| **10** | **1*/*3** | **0** | **0** | **0** | **10** | **0** | **0** | **1** | **2*/*3** |

* 1. **Considerando *J* (00) = 1, calcule os valores de *J* (01), *J* (11) e *J* (10) de forma que M1 e M2 tenham os valores dados acima.**

M1) 00 → 10 :

00 → 10 :

M1) 10 → 00 : 1

10 → 10 : 0

M2) 11 → 10 : 1

M2) 00 → 01 :

00 → 01 :

00 → 01 : 2/3

M1) 01 → 11 :

01 → 11 :

|  |  |  |  |
| --- | --- | --- | --- |
| x1 | x2 | J(x) | |
| 0 | 0 | | 1 |
| 0 | 1 | | 2.0986 |
| 1 | 1 | | 3.1972 |
| 1 | 0 | | 2.0986 |

* 1. **Calcule uma matriz de transi¸c˜ao M que modele transi¸c˜oes de qualquer um dos quatro estados para qualquer um dos quatro estados.**
  2. **Calcule o vetor invariante da matriz M do item (b). Verifique que ele ´e um vetor invariante tamb´em de M1 e M2, apesar de estas matrizes terem diferentes autovetores correspondentes aos autovalores que tˆem valor igual a 1.**

Vetor inveriante da matriz M (item b): [0.56249654 0.18750115 0.06250115 0.18750115]

Vetor invariante da matriz M1: ([[0.75, 0. , 0. , 0.25],

[0. , 0.75, 0.25, 0. ]])

Vetor invariante da matriz M2: ([[0.75, 0.25, 0. , 0. ],

[0. , 0. , 0.25, 0.75]])

1. **Prova de 2016 - Questão 3.**

**(*Simulated Annealing*) Considere uma fun¸c˜ao custo dada pela tabela a seguir:**

*x* 1 2 3 4

*J* (*x*) 7 1 10 4

**a) Descreva, usando pseudo-c´odigo, a implementa¸c˜ao do algoritmo S.A. b´asico aplicado `a minimiza¸c˜ao da fun¸c˜ao custo acima. Na sua descri¸c˜ao, leve em considera¸c˜ao os seguintes parˆametros: temperatura inicial *T*0, temperatura m´ınima *Tmin*, e o nu´mero de itera¸c˜oes *N* a serem executadas em temperatura fixa.**

import math

import numpy as np

def perturbacao(x): ### +1, +2 ou +3

    e = math.ceil(np.remainder(2+(np.random.uniform()\*3),3))

    print(e)

    x\_hat = x + e

    if x\_hat == 5:

        return 1

    if x\_hat == 6:

        return 2

    if x\_hat == 7:

        return 3

    else:

        return x\_hat

def Custo(x):

    if x==1:

        return 7

    if x==2:

        return 1

    if x==3:

        return 10

    if x==4:

        return 4

X0 = 1

X = X0

J0 = Custo(X0)

Jatual = J0

N = 100

K = 8

T0 = 1

fim = 0

n = 0

k = 1

Jmin = Jatual

Xmin = X

while not(fim):

    T=T0/np.log2(2+k)

    for n in range(0,N):

        X\_hat = perturbacao(X)

        JX = Custo(X)

        JX\_hat = Custo(X\_hat)

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

            X = X\_hat

            JX = JX\_hat

            if JX<Jmin:

                Jmin = JX

                Xmin = X

        if np.remainder(n+1,100)==0:

            print([k,n+1,Xmin,Jmin])

    k+=1

    if k==K: fim=1

print(Jmin)

Resultado: Jmin = 1 e Xmin=2, como esperado.

**b) Calcule as matrizes de transi¸c˜ao do processo de Markov que corresponde ao S.A. `a temperatura *T***

**= 10 e à temperatura *T* = 5 (chamadas de M10 e M5) e os seus respectivos vetores invariantes.**

Vetor invariante de M10: ([[0.20355008, 0.37089243, 0.15079361, 0.27476387]])

Vetor invariante de M5: ([[0.14945343, 0.49620287, 0.08202178, 0.27232191]])

**c) (0.25 ponto extra) Observe o menor dos nu´meros em M10 e o menor dos nu´meros em M5. Qual ´e a rela¸c˜ao entre estes nu´meros e *T* , *Jmax*, *Jmin* e o nu´mero de estados poss´ıveis?**

T = 10 , Jmax = 10, Jmin =1, min(M10 = 0.0863939), n = 4

T = 5 , Jmax = 10, Jmin =1, min(M5 = 0.0550996), n = 4

Com a diminuição da temperatura a probabilidade do estado de menor probabilidade diminuiu, o que indica uma convergência para o ponto de mínimo, como proposto pelo Simulated Annealing.

1. **Prova de 2017 - Questão 3, itens (b) e (c).**

**(*Simulated Annealing*) Considere uma função custo definida sobre cinco estados discretos, chamados de estados 1*,* 2*, ...* 5, com os seguintes valores: *J*(1) = *J*(5) = 4, *J*(2) = 1, *J*(3) = 3 e *J*(4) = 2.**

**b) Calcule uma matriz de transição entre estados à temperatura *T*1 = 1/ln2 e uma matriz de transição entre estados à temperatura *T*2 = 1*/* ln 3.**

Com T1 = 1/ln2:

Com T2 = 1/ln3:

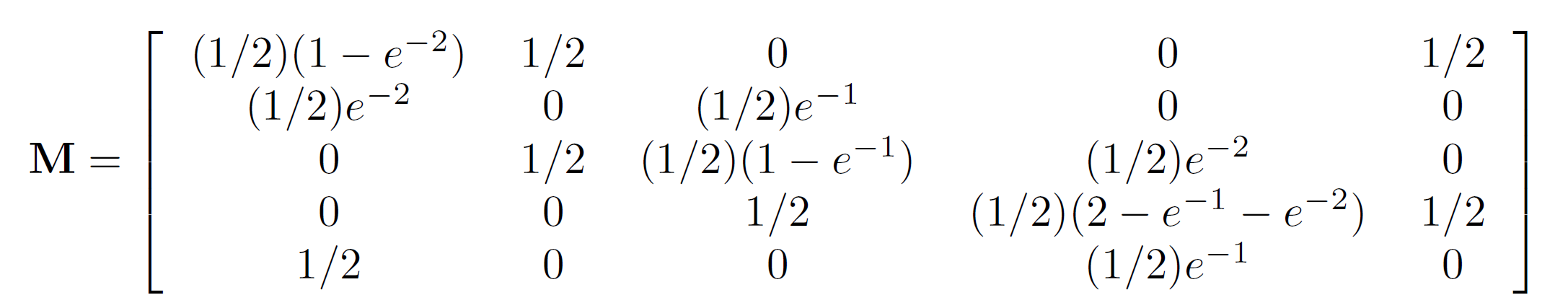
**c) Calcule os vetores invariantes das matrizes encontradas no item (b).**

O vetor invariante da matriz M1 é: ([[0.0625, 0.5 , 0.125 , 0.25 , 0.0625]])

E o vetor invariante da matriz M2 é: ([[0.02439024, 0.65853659, 0.07317073, 0.2195122 , 0.02439024]])

1. **Prova de 2018 - Questão 3.**

**(*Simulated Annealing*) Considere a matriz de transi¸c˜ao dada a seguir (calculada usando *T* = *T*0 = 0*.*1):**

****

1. **Considerando um grafo com cinco estados (estado 1 conectado aos estados 2 e 5; 2 conectado a 1 e 3; 3 conectado a 2 e 4; 4 conectado a 3 e 5; 5 conectado a 4 e 1) e considerando que o custo associado ao estado 1 ´e igual a 0.2, calcule os custos associados aos estados 2, 3, 4 e 5.**

1 → 2 :

3 → 2 :

4 → 3 :

4 → 5 :

|  |  |  |
| --- | --- | --- |
| x | J(x) | |
| 1 | | 0.2 |
| 2 | | 0.4 |
| 3 | | 0.3 |
| 4  5 | | 0.1  0.2 |

1. **Calcule o vetor invariante da matriz M.**

import numpy as np

M = np.zeros([5,5])

M[0,0] = 1/2\*(1-np.exp(-2))

M[1,0] = 1/2\*np.exp(-2)

M[2,0] = 0

M[3,0] = 0

M[4,0] = 1/2

M[0,1] = 1/2

M[1,1] = 0

M[2,1] = 1/2

M[3,1] = 0

M[4,1] = 0

M[0,2] = 0

M[1,2] = 1/2\*np.exp(-1)

M[2,2] = 1/2\*(1-np.exp(-1))

M[3,2] = 1/2

M[4,2] = 0

M[0,3] = 0

M[1,3] = 0

M[2,3] = 1/2\*np.exp(-2)

M[3,3] = 1/2\*(2-np.exp(-1)-np.exp(-2))

M[4,3] = 1/2\*np.exp(-1)

M[0,4] = 1/2

M[1,4] = 0

M[2,4] = 0

M[3,4] = 1/2

M[4,4] = 0

autovals, autovecs = np.linalg.eig(M)

inv = np.zeros(5)

z = np.sum(autovecs[:,0])

for i in range(0,5):

    inv[i] =  autovecs[i,0] / z

print('Vetor inveriante da matriz M: ' + str(inv) )

Vetor inveriante da matriz M: [0.19151597 0.02591887 0.07045479 0.52059439 0.19151597]

1. **Escreva a menor das probabilidades da matriz M em fun¸c˜ao dos valores m´aximo e m´ınimo de *J* (*x*), da temperatura *T* e do nu´mero *N* das transi¸c˜oes poss´ıveis para cada estado. Recalcule esta probabilidade para *T* = *T*0*/* log2 4.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | J(x) | | ***e-J*(x)*/T***  ***T0= 0.1*** | ***e-J*(x)*/T/∑***  ***T=0.1*** | ***e-J*(x)*/T***  *T=T0 / log2* | ***e-J*(x)*/T/∑***  *T=T0 / log2* |
| 1 | | 0.2 | 0.13533 | 0.19151 | 0.54768 | 0.21556 |
| 2 | | 0.4 | 0.01831 | 0.02592 | 0.29995 | 0.11806 |
| 3 | | 0.3 | 0.04979 | 0.07045 | 0.40531 | 0.15953 |
| 4  5 | | 0.1  0.2 | 0.36788  0.13533 | 0.52059  0.19152 | 0.74005  0.54768 | 0.29128  0.21556 |
| ∑ | |  | 0.70665 |  | 2.54069 |  |