CPE-723 – Otimização Natural

Lista de Exercícios #1

Amanda Isabela de Campos (DRE 120074842)

01) Prova de 2007 - Questão 1

1. *Método de Monte Carlo*:

a) Efetue as três primeiras iterações do cálculo de $\int_0^1 x^3\,dx$, usando os três números aleatórios a seguir, que foram sorteados de uma distribuição uniforme no intervalo $[0, 1]$:

0.9501, 0.2311, 0.6068, ...

$\int_0^1 x^3\,dx$ = 0.25

(0.9501^3+0.2311^3+0.6068^3)/3
0.3644719103879999

Utilizando um gerador de números com densidade uniforme tem-se o valor da integral calculada como:

```
import numpy as np
N = 100000
x = np.random.uniform(0,1,N)
print(np.mean(x**3)) = 0.24949623682519967
```

b) Efetue as três primeiras iterações do cálculo de $\int_0^1 x^2\,e^{-x}dx$, usando os quatro números aleatórios a seguir, que foram sorteados de uma distribuição exponencial $f_x(x) = e^{-x}$ no intervalo $[0, +\infty]$:

0.0512, 1.4647, 0.4995, 0.7216 ...

$\int_0^1 x^2\,e^{-x}dx$ = 0.1606

(0.0512^2 + 0.4995^2 + 0.7216^2)/4 = 0.19320706250000003

Utilizando um gerador de números aleatórios com densidade exponencial tem-se o valor da integral calculada como:

```
import numpy as np
N = 100000
x = np.random.exponential(1,N)
x = x**2
print(np.sum(x[x<1])/N)  = 0.1608640519397457
```

02) Prova de 2007 - Itens 2(a) e 2(b)

2. (*Algoritmo de Metropolis*) Considere uma variável aleatória $X \in \{1, 2, 3, 4, 5\}$ e uma função custo $J(x) = (x - 3)^2$. Considere $T = 1$.

a) Calcule os fatores de Boltzmann exp(-*J*(*x*) / *T*), para *x* = 1, 2, 3, 4, 5.

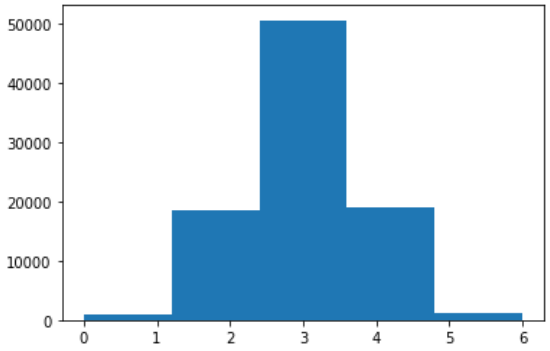
x	J(x)	exp(-*J*(*x*) / *T*)
1	4	0.01831564
2	1	0.36787944
3	0	1
4	1	0.36787944
5	4	0.01831564

b) Proponha um algoritmo para gerar uma distribuição de Boltzmann/Gibbs para a variável aleatória *X*, conforme os custos *J*(*x*).

Nesse caso, a perturbação contínua "x_hat = x + epsilon*R" foi substituída por uma perturbação discreta, tipo "x_hat = x (+ ou -) 1").

```
N = 100000
M = 10000
T = 1

def J(x):
   return (x-3)**2

x = np.zeros(N)
n=0
x[n] = np.random.choice([1,2,3,4,5])

for n in range(1,N):
   perturbacao = np.random.choice([-1,1])

   if x[n] == 5:
      if perturbacao == 1:
         x_hat = 1
   elif x[n] == 1:
      if perturbacao == -1:
         x_hat= 5
   else:
      x_hat = x[n-1] + perturbacao

   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[M:], 5)
plt.show()
```

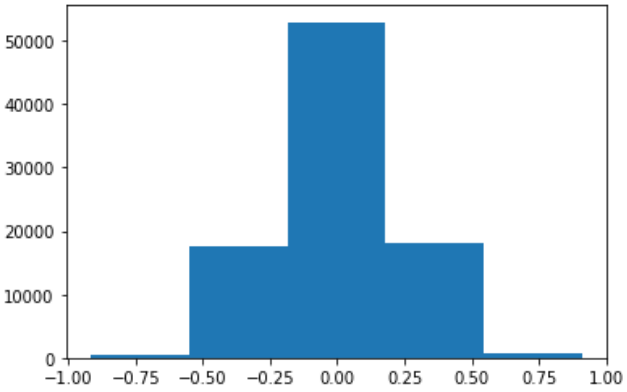


03) Lista de Exercícios 1 da CPE723 Edição Presencial - Exercício 3(a)

Escrever um algoritmo para gerar números x(n) com energia J(x) = x^2 , de forma que as probabilidades dos números gerados sejam proporcionais aos fatores de Boltzmann exp(−J(x)/T), com temperatura T = 0.1. Começando de um valor x(0) qualquer, aplique sempre perturbações eR ao valor x(n) atual. Neste caso, R é uma variável aleatória uniforme. Considere e = 0.1:

a) Execute o algoritmo proposto no computador, calculando x(n) até n = 100.000.

```
N = 100000
M = 10000
T = 1
x = np.zeros(N)
n=0
x[0] = np.random.uniform(0,1)
e = 0.1; T = 0.1;
J0 = x[0]**2; Xatual = x[0]; Jatual = J0;
for n in range(1,N):
    x_hat = x[n-1] + e*np.random.randn()
    if np.random.uniform(0,1)< np.exp(((x[n-1])**2 - (x_hat)**2)/T):
        x[n] = x_hat
    else:
        x[n] = x[n-1]
y = np.linspace(0,6,1000)
plt.figure(1)
plt.hist(x[M:], 5)
plt.show()
```



04) Prova de 2008 - Questão 1

(*Algoritmo de Metropolis*) Nos itens a seguir, considere o uso do algoritmo de Metropolis e de uma variável aleatória binária *R* (com dois valores equiprováveis, ou seja, $p_R(0) = p_R(1) = 0{,}5$ para a geração de uma variável aleatória *X* com função densidade de probabilidade arbitrária, dada por $f_X(x)$:

a) Qual deve ser a função custo *J*(*x*), para que a densidade de probabilidade de *x* seja $f_X(x)$?

$$(1/Z)*\exp(-J(x)/T) = f_x(x)$$

Aplicando a função logarítmica natural f(x) = ln x em ambos os lados resulta em:

$$\ln(1/Z) + J(x) = -T*\ln(f_x(x))$$

$$\ln(1) - \ln(Z) + J(x) = -T*\ln(f_x(x))$$

A constant Z pode ser desconsiderada já que ln(Z) vai ser cancelada nos cálculos de DeltaJ. Logo, a função custo será:

$$J(x) = -T*\ln(f_x(x))$$

b) Utilizando um pseudo-código, descreva o algoritmo de Metropolis aplicado à geração da variável aleatória em questão. Defina e use os parâmetros (tamanho da perturbação, número de iterações, etc.) que você julgar necessários.

```
N = 100000
M = 10000
def J(x):
   return -T*np.log(fx)
x = np.zeros(N)
n=0
x[0] = np.random.uniform(0,1)
e = 0.1; T = 0.1;
J0 = J(x[0]); Xatual = x[0]; Jatual = J0;

for n in range(1,N):
   perturbacao = np.random.choice([-1,1])
   x_hat = x[n-1] + e*perturbacao
   if np.random.rand(1)< np.exp(((J[n-1]) - J(x_hat))/T):
```

```
    x[n] = x_hat
  else:
    x[n] = x[n-1]
```

05) Prova de 2011 - Questão 1

(*Algoritmo de Metropolis*) Considere a seguinte expressão:

$$\int_{|x_2|<1}\int_{|x_1|<1}\left(x_1^2+x_2^2\right)e^{-(x_1^2+x_2^2)}dx_1dx_2$$

a) Escreva, utilizando um pseudo-código, um programa para a geração de vetores aleatórios (*x*1, *x*2) que tenham uma densidade conveniente para uma avaliação eficiente desta expressão.

```
import numpy as np

J = lambda x: np.linalg.norm(x)**2
fx = lambda x: np.exp(-J(x))
N = int(3e6)
M = int(0.1*N)
x = np.zeros((N,2))
x[0,:] = np.random.random((1,2))**2-1

eps = 1e-4

F_hat = 0

for i in range(1,N):

   r = np.random.random((1,2))*2 - 1

   x_hat = x[i-1,:] + eps*r

   if x_hat[0,0] > 1 : ### Atualização dos valores x_hat dentro dos limites
      x_hat[0,0] = x_hat[0,0] - 2

   if x_hat[0,1] > 1 :
      x_hat[0,1] = x_hat[0,1] - 2

   if x_hat[0,0] < -1 :
      x_hat[0,0] = -(x_hat[0,0] + 1)

   if x_hat[0,1] < -1 :
      x_hat[0,1] = -(x_hat[0,1] + 1)

   if np.random.uniform() < np.exp(J(x[i-1]) - J(x_hat)):

      x[i] = x_hat
   else: x[i] = x[i-1,:]

   if i<N-M:
      F_hat += fx(x[i])
   print(F_hat/M)
```

```
values = x[-M,:]

n = np.linalg.norm(values, axis=1)**2
z = np.exp(-n)
print(np.sum(z)/M)
```

b) Explique como os vetores gerados pelo programa do item (a) podem ser utilizados para a avaliação da integral.

Os vetores foram gerados de forma que todos os valores aceitos estejam dentro do quadrado de restrições, ou seja, dentro dos limites para x_1: $-1>x_1>1$ e para x_2: $-1>x_2>1$. Para isso foi adotado o artificio semelhante a uma "warp zone", isto é, quando um valor sorteado está fora desse quadrado de limites o valor escolhido é rebatido para o lodo oposto, como se as bordas fossem infinitas. Como exemplo, imagina-se que o resultado da perturbação x_1 seja maior que 1, o resultado que será utilizado será transformado em um resultado maior que -1. Isto é, se $x\_hat_1 = 1.03$ este será transformado em $x\_hat_1 = -0.97$.

06) Prova de 2012 - Itens 1(a), 1(b) e 1(c)

(*Algoritmo de Metropolis*) Nesta questão, consideramos o problema da descrição de todas as configurações possíveis de um sistema com 5 partículas em duas dimensões. A posição de cada partícula é definida por um vetor x*i*, *i* = 1, 2, ..., 5 e o estado do sistema é definido por um vetor x contendo todas as 10 coordenadas. Duas configurações particulares são mostradas na figura do item (b). A função custo na qual estamos interessados é uma combinação linear entre a soma das normas dos vetores x*i* e a soma das "repulsões eletrostáticas" entre as partículas, imaginando o caso em que todas são positivas:

$$J(x) = \sum_i \left||x_i|\right|^2 + \lambda \sum_{j \geq 1} \frac{1}{\left||x_i - x_j|\right|^2}$$

a) Escreva, utilizando pseudo-código, uma implementação do algoritmo de Metropolis que, para temperatura *T* e a partir de uma configuração inicial qualquer, permita a geração de estados seguindo uma distribuição de Boltzmann em função dos seus custos *J*.

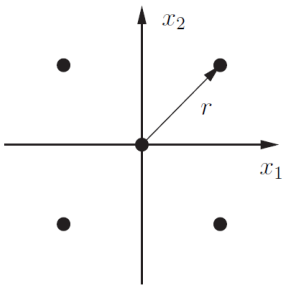
```
#Função custo
def J(X):
   L=0.5
   Xorigin=np.zeros([2,1])
   JA=np.mean(np.sum(np.power(X-np.tile(Xorigin,(1,P)),2),axis=0))
   JB=0
   for i in range(0,P):
      for j in range(i+1,P):
         JB+= np.sum(1/(np.power(X[:,i]-X[:,j],2)))
   JB=JB/(P*(P-1)/2)
   return JA+L*JB

N=int(1e5); epsilon=1e-1
np.random.seed(0); P=5; X=np.random.normal(0,1,[2,P])
fim=0; n=0; Jmin=J(X); Xmin=X; T=1; M = 0.1*N

#Algoritmo de Metropolis
for n in range(0,N):
```

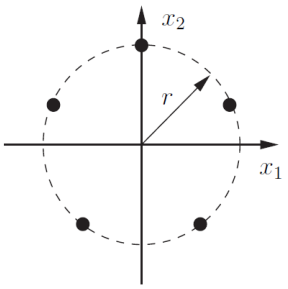
```
Xhat=X+epsilon*np.random.normal(0,1,np.shape(X))
if np.random.uniform()<np.exp((J(X)-J(Xhat))/T):
    X=Xhat
else:
    X=X
```

b) Para as duas soluções locais a seguir, uma expressão simplificada para *J* pode ser calculada em função da variável escalar positiva *r*:

$$J_1(r) = 4r^2 + \frac{6.5}{r^2} \qquad \text{e} \qquad J_2(r) = 5r^2 + \frac{5}{r^2}$$

Assumindo *T* = 0,1, calcule a proporção entre as probabilidades de um estado que tem a configuração da direita com *r* = 1:0000 e outro estado que tem a configuração da esquerda com *r* = 1.1291.





$J_1(1.1291) = 10.1980390$ $J_2(1.0) = 10.0$

Sabe-se que as probabilidades de cada estado são dadas por:

$$p_1 = p(r = 1.1291) = \frac{\exp(-\frac{J_1(r)}{T})}{z}$$

$$p_2 = p(r = 1.0000) = \frac{\exp(-\frac{J_2(r)}{T})}{z}$$

$$\frac{p_2}{p_1} = \frac{\exp(-\frac{10.198}{0.1})}{\exp(-\frac{10.0}{0.1})} = 7.2456$$

A seguir está o código adotado:

```
J_1 = lambda r: 4*r**2 + 6.5/(r**2)
J_2 = lambda r: 5*r**2 + 5/(r**2)
T = 0.1
r_dir = 1
r_esq = 1.1291

print('r=1.1291', 'J_1 = ', J_1(r_esq))
print('r=1.0', 'J_2 = ', J_2(r_dir))
```

#r=1.1291 J_1 = 10.19803906649535
#r=1.0 J_2 = 10.0

p = np.exp(-(J_2(r_dir))/0.1) / np.exp(-(J_1(r_esq))/0.1)
print(p) #7.245573023774906

c) Considere a definição da variável aleatória "distância média à origem": $L(x) = (1/5)\sum_i ||x_i||$. Explique como o algoritmo do item (a) é modificado, de forma que possamos calcular o valor médio de *L* a uma temperatura *T* arbitrária.

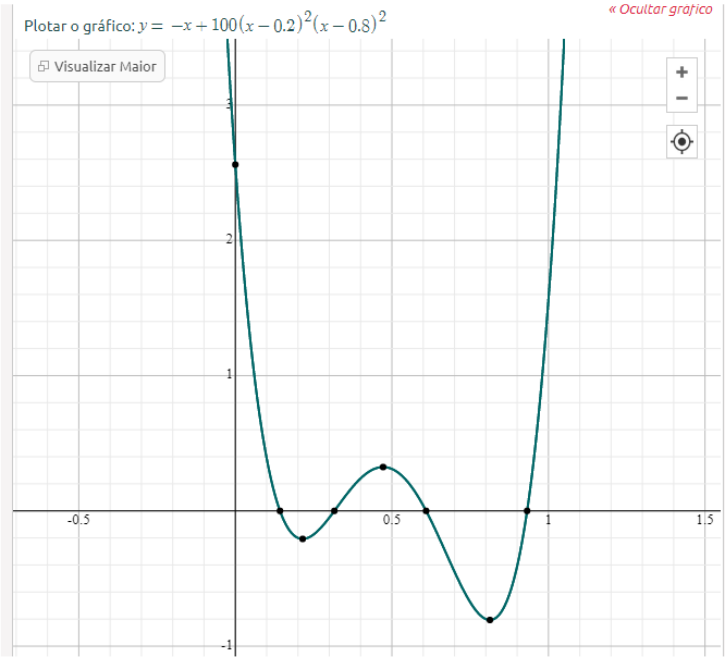
A modificação deve ser incluída no final do loop do algoritmo de Metropolis da seguinte forma:

L = (1./5)* (np.sum(np.tile(X,(1,P))),axis=0))

A estimativa do valor médio da variavel L é feita apenas nos últimos estados.

07) Lista de Exercícios 1 da CPE723 Edição Presencial - Exercício 4(a) (de novo, não é preciso fazer o item (b), cálculo manual com 10 valores)

Escrever um programa de S.A. (pode ser pseudo-código) para minimizar a função escalar J (x) = $-x + 100(x - 0.2)^2 (x - 0.8)^2$. Começando de x (0) = 0 e utilizando geradores de números aleatórios (um uniforme e outro Gaussiano), calcule manualmente os 10 primeiros valores de x (n) gerados pelo S.A.



Pelo gráfico o ponto de mínimo global é (0.81302, -0.80664 e o ponto de mínimo local (0.21499, -0.20729).

```
# Função Custo
def Custo(x):
    J = -x + 100*(pow(x-0.2,2)*(pow(x-0.8,2)))
    return J
```

```
X0 = 0
X = X0
J0 = Custo(X0)
Jatual = J0

N = 100000
K = 8
T0 = 10
e = 5e-2
fim = 0
n = 0
k = 1
Jmin = Jatual
Xmin = X

history_J=np.zeros([int(N*K),1]); history_T=np.zeros([int(N*K),1])

while not(fim):
   T=T0/np.log2(2+k)
   for n in range(0,N):
      X_hat = X + e*np.random.uniform()
      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)
```

O algoritmo apresentou como resultado de mínimo global o ponto (0.83099, -0.79275) que são valores próximos ao esperado para a função dada.

08) Lista de Exercícios 1 da CPE723 Edição Presencial - Exercício 5

Proponha uma função de até 4 variáveis cujo ponto mínimo você conheça, e encontre este ponto mínimo utilizando S.A. (neste exercício, basta entregar o código escrito).

A função escolhida foi a ZAKHAROV FUNCTION (referência: https://www.sfu.ca/~ssurjano/zakharov.html) que segue expressão seguinte, onde *d* é o número de dimensões, nesse exemplo definido igual a 4.

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^4$$

Sabe-se que o ponto de mínimo global é:

$$f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (0, \ldots, 0)$$

Aplicando o Simulated Annealing tem-se o seguinte código:

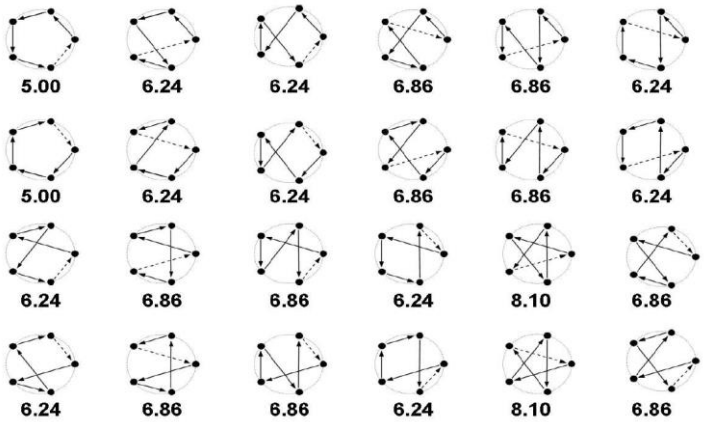
```
import numpy as np
number_variables = 4
upper_bounds = [10, 10, 10, 10]
lower_bounds = [-10, -10, -10, -10]

def Custo(X):
    x1 = X[0]
    x2 = X[1]
    x3 = X[2]
    x4 = X[3]

    value = x1**2 + x2**2 + x3**2 + x4**2 + (0.5*x1 + 0.5*2*x2 + 0.5*3*x3 + 0.5*4*x4)**2 +
(0.5*x1 + 0.5*2*x2 + 0.5*3*x3 + 0.5*4*x4)**4

    return value

#-------------------------------------------------------------------------
# Simulated Annealing
X0=np.zeros((number_variables))
for v in range(number_variables):
    print (v)
    X0[v] = random.uniform(lower_bounds[v],upper_bounds[v])


N=int(1e5);  K=7; 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;
history_J=np.zeros([int(N*K),1]); history_T=np.zeros([int(N*K),1])
X_hat = np.zeros(number_variables)

while not(fim):
    T = T0/np.log2(2+k)
    for n in range(N):

        for k in range(number_variables):
            X_hat[k] = X[k] + e*(random.uniform(lower_bounds[k],upper_bounds[k]))
            X_hat[k] = max(min(X[k],upper_bounds[k]),lower_bounds[k])  # Solução dentro dos limites
        if np.random.uniform()<np.exp((Custo(X)-Custo(X_hat))/T):
            X = X_hat
            if Custo(X) < Jmin:
                Jmin = Custo(X)
                Xmin = X
            history_J[k*N+n] = Custo(X)
```

```
        history_T[k*N+n] = T
    print([k,Jmin])
    k=k+1
    if k == K: fim =1
print(Jmin)
```

O ponto de Jmin = 0 foi encontrado com a convergência do algoritmo.

09) Prova de 2008 - Questão 3

(*Simulated Annealing*) A figura a seguir ilustra todas as soluções possíveis do problema do caixeiro viajante com cinco cidades, no caso em que as cinco cidades estão dispostas uniformemente sobre um círculo, e considerando que a viagem sempre começa pela cidade mais à direita. A seta pontilhada indica o caminho de retorno da última cidade visitada para a cidade inicial. Considera-se que o custo da viagem sobre um lado do pentágono formado pelas cidades é igual a 1.0. O custo total de cada solução é representado logo abaixo da mesma.



a) Utilizando um pseudo-código, descreva um algoritmo de *Simulated Annealing* para resolver este problema. Defina e use quaisquer parâmetros (por exemplo: temperatura inicial, método de resfriamento, número de iterações a temperatura fixa, etc.) que você julgar necessários.

```
def pertubação(X): #Criação de uma nova sequência de cidades visitadas
    pos1 = np.random.randint(1,5)
    pos2 = np.random.randint(1,5)

    aux1 = X[pos1]
    aux2 = X[pos2]

    X[pos1] = aux2
    X[pos2] = aux1

    return X

def Custo(X):  ##Cálculo das distancias
    if (np.array_equal(X,np.array([1,2,3,4,5])) or  np.array_equal(X,np.array([1,5,4,3,2]))):
        J = 5
    elif (np.array_equal(X,np.array([1,2,3,5,4])) or np.array_equal(X,np.array([1,2,4,3,5])) or
          np.array_equal(X,np.array([1,2,5,4,3])) or np.array_equal(X,np.array([1,5,4,2,3])) or
          np.array_equal(X,np.array([1,5,3,4,2])) or np.array_equal(X,np.array([1,5,2,3,4])) or
```

```
        np.array_equal(X,np.array([1,3,2,4,5])) or np.array_equal(X,np.array([1,3,4,5,2])) or
        np.array_equal(X,np.array([1,4,5,3,2])) or np.array_equal(X,np.array([1,4,3,2,5])) ):
       J = 6.24

    elif ( np.array_equal(X,np.array([1,2,4,5,3])) or np.array_equal(X,np.array([1,2,5,3,4])) or
        np.array_equal(X,np.array([1,5,3,2,4])) or  np.array_equal(X,np.array([1,5,2,4,3])) or
        np.array_equal(X,np.array([1,3,2,5,4])) or  np.array_equal(X,np.array([1,3,4,2,5])) or
        np.array_equal(X,np.array([1,3,5,4,2])) or  np.array_equal(X,np.array([1,4,5,2,3])) or
        np.array_equal(X, np.array([1,4,3,5,2])) or  np.array_equal(X,np.array([1,4,2,3,5]))):
       J = 6.86

    else: #( np.array_equal(X,np.array([1,3,5,2,4])) or  np.array_equal(X,np.array([1,4,2,5,3]))):
       J = 8.10
    return J

#Inicialização
X0 = np.array([1,3,5,2,4]) #Percurso inicial aleatório
X = X0
J0 = Custo(X0)
Jatual = J0

N = 100
K = 8
T0 = 1
e = 5e-2
fim = 0
n = 0
k = 1
Jmin = Jatual
Xmin = X

#Simulated Annealing
while not(fim):
    T=T0/np.log2(2+k)
    for n in range(0,N):
        X_hat = pertubação(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)
```

b) Com temperatura fixa T = 1, calcule a probabilidade com que cada uma das soluções acima será gerada, após a convergência do algoritmo.

$$p(J_1) = \frac{e^{(-J_1/T)}}{z} = \frac{e^{(-J_1/T)}}{\sum e^{(-J/T)}}$$

$$p(J = 5) = \frac{e^{(-5)}}{\sum e^{(-J/T)}} = \frac{2e^{(-5)}}{2e^{-5} + 10e^{-6.24} + 10e^{-6.86} + 2e^{-8.10}} = 0.30577921705703975$$

$$p(J = 6.24) = \frac{e^{(-6.24)}}{\sum e^{(-J/T)}} = \frac{10e^{(-6.24)}}{2e^{-5} + 10e^{-6.24} + 10e^{-6.86} + 2e^{-8.10}} = 0.44243839795033324$$

$$p(J = 6.86) = \frac{e^{(-6.86)}}{\sum e^{(-J/T)}} = \frac{10e^{(-6.86)}}{2e^{-5} + 10e^{-6.24} + 10e^{-6.86} + 2e^{-8.10}} = 0.23800727515568074$$

$$p(J = 8.10) = \frac{e^{(-8.10)}}{\sum e^{(-J/T)}} = \frac{2e^{(-8.10)}}{2e^{-5} + 10e^{-6.24} + 10e^{-6.86} + 2e^{-8.10}} = 0.013775109836946233$$

A soma das probabilidades é 1, como esperado.