

# **CPE723 Otimização Natural:**

## **Lista de Exercícios #1**

Data de entrega: Quinta, 21 de Março, 2019

*Prof. José Gabriel Rodríguez Carneiro Gomes, Terças e Quintas: 08:00-10:00*

**Vinicius Mesquita de Pinho**

## Questão 1

Calcular  $\int_0^1 xe^{-x}dx$  de três formas diferentes.

a) **Primeiro calculando a integral indefinida.**

$$\int xe^{-x}dx = -e^{-x}x - e^{-x} + k, \quad (1)$$

onde  $k$  é uma constante, que daqui para frente assumiremos  $k = 0$ . Aplicando integral por parte com  $u = x$  e  $v' = e^{-x}$ , teremos

$$-e^{-x}x - e^{-x} + k = -e^{-x} - \int -e^{-x}dx. \quad (2)$$

Como  $\int -e^{-x}dx = e^{-x}$ , teremos

$$-e^{-x} - \int -e^{-x}dx = -e^{-x}x - e^{-x}. \quad (3)$$

Calculando os limites:

$$\int_0^1 xe^{-x}dx = -\frac{2}{e} - (-1) = -\frac{2}{e} + 1 \approx 0.26424\dots \quad (4)$$

b) **Pelo método de Monte Carlo, usando 10 números escolhidos aleatoriamente com densidade uniforme entre 0 e 1.**

Os números gerados:  $\mathcal{I} = (0.66606563, 0.1280393, 0.15019057, 0.53518197, 0.72382377, 0.56306944, 0.18701318, 0.96037606, 0.79777229, 0.23135354)$

O cálculo:

$$\int_0^1 xe^{-x}dx \approx \frac{\sum_{i \in \mathcal{I}} ie^{-i}}{|\mathcal{I}|} \quad (5)$$

onde  $|\mathcal{I}|$  é a cardinalidade do conjunto  $\mathcal{I}$ .

Para o conjunto  $\mathcal{I}$  o resultado foi 0.2634.

c) **Pelo método de Monte Carlo, usando 10 números escolhidos aleatoriamente com densidade exponencial (note que as amostras geradas a partir da p.d.f. exponencial devem ser limitadas ao intervalo de 0 a 1).**

Os números gerados:  $\mathcal{W} = (0.55005713, 0.39015194, 0.23349916, 0.41284879, 0.22265732, 0.69167714, 0.43932533, 0.09646257, 0.60023859, 0.98892791)$

Utilizando a equação (5) para o conjunto  $\mathcal{W}$ , o resultado encontrado foi 0.2632.

## Questão 2

Usando  $N = 20$  números aleatórios, escolhidos a partir de uma p.d.f. uniforme entre  $-1$  e  $+1$ , calcular uma aproximação para o número  $\pi$  pelo método de Monte Carlo. Faça o mesmo no computador, utilizando um valor alto para  $N$  (por exemplo, 1.000.000). Comente o resultado.

A fórmula para  $\pi$  utilizada é,

$$\pi = \int_{-1}^1 \frac{dx}{\sqrt{1-x^2}} \quad (6)$$

Primeiro, para o  $N = 20$ .

Os números gerados:  $\mathcal{Q} = (-0.85289404, 0.92237068, 0.32455622, 0.0946911, 0.8676302, -0.92568284, 0.76944606, -0.90241725, 0.20247619, -0.69989244, -0.64724326, 0.91118243, 0.2354838, 0.8327723, -0.21647745, -0.5894811, -0.81595565, -0.61292112, 0.12647561, -0.33242752)$

O cálculo:

$$\int_{-1}^1 \frac{dx}{\sqrt{1-x^2}} \approx \frac{\sum_{q \in \mathcal{Q}} \left( \sqrt{1-q^2} \right)^{-1}}{|\mathcal{Q}|} \quad (7)$$

onde  $|\mathcal{Q}|$  é a cardinalidade do conjunto  $\mathcal{Q}$ .

Utilizando o conjunto  $\mathcal{Q}$ , o resultado obtido foi 3.142843.

Para um valor de  $N$  alto, foi utilizado  $N = 10^7$ , onde o valor obtido foi 3.1457290.

A lei dos grandes números garante que a com um número maior amostras utilizadas, a aproximação converge para o resultado da integral.

### Questão 3

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  $e^{-J(x)/T}$ , com temperatura  $T = 0.1$ . Começando de um valor  $x(0)$  qualquer, aplique sempre perturbações  $\epsilon R$  ao valor  $x(n)$  atual. Neste caso,  $R$  é uma variável aleatória uniforme. Considere  $\epsilon = 0.1$ .

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

O algoritmo em Python.

```
# Ponto inicial x = 0
x_inicial = np.array(0)

J_inicial = funcao_J(x_inicial)
x_atual = x_inicial
J_atual = J_inicial

# Parametros utilizados
N = 10**5
T_inicial = 0.1
T = T_inicial
epsilon = 0.1

fim = 0
n = 0
k = 1
J_min = J_atual
x_min = x_atual

# Para armazenar os J e os x
todos_J = np.array([]), todos_x = np.array([])

while not(fim):
    n = n + 1
    x = x_atual + epsilon*(np.random.uniform(-1, 1))
    J = funcao_J(x)
    todos_J = np.append(todos_J, J)
    todos_x = np.append(todos_x, x)
    if (np.random.uniform(0,1) < np.exp((J_atual-J)/T)):
        x_atual = x
        J_atual = J
    if (J < J_min):
        J_min = J
        x_min = x
    if (n % N == 0):
        fim = 1
```

**b) Execute manualmente os 10 primeiros passos do algoritmo.**

Utilizando a mesma lógica proposta na letra a), executamos os 10 primeiros passos do algoritmo, manualmente.

Iteração	x atual	x candidato	J(x candidato)	r (uniforme)	q	r < q
<b>1</b>	0.0000	0.0703	0.8262	0.9646	~ 3300	V
<b>2</b>	0.0703	0.0770	0.7146	0.8961	3.0537	V
<b>3</b>	0.0770	0.1033	0.3512	0.2745	37.8699	V
<b>4</b>	0.1033	0.1109	0.2666	0.3008	2.3294	V
<b>5</b>	0.1109	0.2090	-0.2062	0.1587	113.0382	V
<b>6</b>	0.2090	0.2666	-0.1403	0.0232	0.5177	V
<b>7</b>	0.2666	0.2908	-0.0771	0.0128	0.5312	V
<b>8</b>	0.2908	0.3590	0.1326	0.8441	0.1229	F
<b>9</b>	0.2908	0.3750	0.1782	0.5509	0.0779	F
<b>10</b>	0.2908	0.3051	-0.0345	0.5210	0.6532	V

## Questão 4

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.

```
# Definindo a funcao
def funcao_J(x):
    return -x + 100*((x-0.2)**2)*((x-0.8)**2)

5 # Definindo ponto inicial
x_inicial = np.array(0)

J_inicial = funcao_J(x_inicial)
x_atual = x_inicial
10 J_atual = J_inicial

# Parametros utilizados
N = 1000
K = 8
15 T_inicial = 5e-3
T = T_inicial
epsilon = 10e-2

fim = 0
20 n = 0
k = 1
J_min = J_atual
x_min = x_atual

25 todos_J = np.array([])
todos_x = np.array([])

while not(fim):
    n = n + 1
30 x = x_atual + epsilon*(np.random.normal(0, 1))
    J = funcao_J(x)
    todos_J = np.append(todos_J, J)
    todos_x = np.append(todos_x, x)
    if (np.random.uniform(0,1) < np.exp((J_atual-J)/T)):
35 x_atual = x
    J_atual = J
    if (J < J_min):
        J_min = J
        x_min = x
40 if (n % N == 0):
        k = k + 1
        T = T_inicial/(np.log(1+k))
    if k == K:
        fim = 1
```

O cálculo manual dos dez primeiros valores de  $x(n)$ :

Neste caso, quando o valor de  $q = \exp((J_{atual} - J)/T)$  é muito grande, representei por  $\sim\text{inf}$  na tabela.

Iteração	x atual	x candidato	J(x candidato)	$r \sim U(0,1)$	q	$r < q$
<b>1</b>	0.0000	0.0775	0.7051	0.0940	$\sim\text{inf}$	V
<b>2</b>	0.0775	-0.0023	2.6372	0.3660	0	F
<b>3</b>	0.0775	0.0766	0.7205	0.6785	0.0454	F
<b>4</b>	0.0775	0.1179	0.1953	0.2147	$\sim\text{inf}$	V
<b>5</b>	0.1179	0.1338	0.0604	0.8711	$\sim\text{inf}$	V
<b>6</b>	0.1338	0.1121	0.2538	0.7671	0	F
<b>7</b>	0.1338	0.1154	0.2204	0.9551	0	F
<b>8</b>	0.1338	0.2042	-0.2036	0.5003	$\sim\text{inf}$	V
<b>9</b>	0.2042	0.1346	0.0545	0.1564	0	F
<b>10</b>	0.2042	0.4453	0.3118	0.5321	0	F

## Questão 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:

$$f(x, y) = (x - 3)^2 + xy + (y - 1)^2 \quad (8)$$

Essa função tem mínimo em  $\{x, y\} = \{\frac{10}{3}, -\frac{2}{3}\}$ .

O código em Python escrito para solução do problema:

```
# Definindo a funcao (que recebe um pont pt que eh funcao de x e y)
def chosen_function(pt):
    x = pt[0]
    y = pt[1]
    return (x-3)**2 + x*y + (y-1)**2

# Ponto inicial
pt_inicial = np.array([0,0])

J_inicial = chosen_function(pt_inicial)
pt_atual = pt_inicial
J_atual = J_inicial

# Parametros utilizados
N = 1000
K = 8
T_inicial = 5e-1
T = T_inicial
epsilon = 10e-2

fim = 0
n = 0
k = 1
J_min = J_atual
pt_min = pt_atual

while not(fim):
    n = n + 1
    pt = pt_atual + epsilon*(np.random.normal(0, 1, 2))
    J = chosen_function(pt)
    if (np.random.uniform(0,1) < np.exp((J_atual-J)/T)):
        pt_atual = pt
        J_atual = J
    if (J < J_min):
        J_min = J
        pt_min = pt
    if (n % N == 0):
        k = k + 1
        T = T_inicial/(np.log(1+k))
    if k == K:
        fim = 1
```