CPE723 Otimização Natural:
Lista de Exercícios #1

Data de entrega: Terça, 19 de Março, 2019

Profs. 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.

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

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

$$-e^{-x}x - e^{-x} + k = -e^{-x} - \int -e^{-x}dx. \tag{2}$$

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

$$-e^{-x} - \int -e^{-x}dx = -e^{-x}x - e^{-x}. \tag{3}$$

Calculando os limites:

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

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:

$$\int_0^1 xe^{-x}d \approx \frac{\sum_{i\in\mathcal{I}} ie^{-i}}{|\mathcal{I}|} \tag{5}$$

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 π 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 π utilizada é,

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

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:

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

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 ejam proporcionais aos fatores de Boltzmann $e^{-J(x)/T}$, com temperatura $T = 0.1$. Começando de um valor $x(0)$ qualquer, aplique sempre perturbações ϵR ao valor $x(n)$ atual. Neste caso, R é uma vriável 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.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)):
          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.
Na 1ª iteração.
O valor de x atual é 0. Com o ruído, o x passa a valer 0.1067.
Avaliando a função em x, temos que J vale 0.01138.
No teste de comparação com a variável uniforme r, o teste foi verdadeiro.
Então no momento, o x atual é 0.1067, e o J atual é 0.01138.
Na 2ª iteração.
O valor de x atual é 0.1067. Com o ruído, o x passa a valer 0.1088.
Avaliando a função em x, temos que J vale 0.01183.
No teste de comparação com a variável uniforme r, o teste foi verdadeiro.
Então no momento, o x atual é 0.1088, e o J atual é 0.01183.
Na 3ª iteração.
O valor de x atual é 0.1088. Com o ruído, o x passa a valer 0.1671.
Avaliando a função em x, temos que J vale 0.0279.
No teste de comparação com a variável uniforme r, o teste foi verdadeiro.
Então no momento, o x atual é 0.1671, e o J atual é 0.0279.
Na 4ª iteração.
O valor de x atual é 0.1671. Com o ruído, o x passa a valer 0.1091.
Avaliando a função em x, temos que J vale 0.0119.
No teste de comparação com a variável uniforme r, o teste foi verdadeiro.
Então no momento, o x atual é 0.1091, e o J atual é 0.0119.
Na 5ª iteração.
O valor de x atual é 0.1091. Com o ruído, o x passa a valer 0.1633.
Avaliando a função em x, temos que J vale 0.0266.
No teste de comparação com a variável uniforme r, o teste foi verdadeiro.
Então no momento, o x atual é 0.1633, e o J atual é 0.0266.
Na 6ª iteração.
O valor de x atual é 0.1633. Com o ruído, o x passa a valer 0.3383.
Avaliando a função em x, temos que J vale 0.1145.
No teste de comparação com a variável uniforme r, o teste foi falso.
Então no momento, o x atual é 0.1633, e o J atual é 0.0266.
Na 7ª iteração.
O valor de x atual é 0.1633. Com o ruído, o x passa a valer 0.1423.
Avaliando a função em x, temos que J vale 0.0202.
No teste de comparação com a variável uniforme r, o teste foi verdadeiro.
Então no momento, o x atual é 0.1423, e o J atual é 0.0202.
Na 8ª iteração.
O valor de x atual é 0.1423. Com o ruído, o x passa a valer 0.0910.
Avaliando a função em x, temos que J vale 0.008.
No teste de comparação com a variável uniforme r, o teste foi verdadeiro.
Então no momento, o x atual é 0.0910, e o J atual é 0.0082.
Na 9ª iteração.
O valor de x atual é 0.0910. Com o ruído, o x passa a valer 0.2206.
Avaliando a função em x, temos que J vale 0.0486.
No teste de comparação com a variável uniforme r, o teste foi falso.
Então no momento, o x atual é 0.0910, e o J atual é 0.0082.
Na 10ª iteração.

O valor de x atual é 0.0910. Com o ruído, o x passa a valer 0.2206.
Avaliando a função em x, temos que J vale 0.0298.
No teste de comparação com a variável uniforme r, o teste foi falso.
Então no momento, o x atual é 0.1726, e o J atual é 0.0298.

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)

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

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

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

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

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

while not(fim):
     n = n + 1
     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)):
          x_atual = x
          J_atual = J
     if (J < J_min):
          J_min = J
          x_min = x
     if (n % N == 0):
          k = k + 1
          T = T_inicial/(np.log(1+k))
     if k == K:
          fim = 1
```