

**Universidade Federal do Rio de Janeiro**  
**COPPE**  
**Programa de Engenharia Elétrica - PEE**  
**Disciplina: Otimização Natural**  
**Aluno: Gustavo Martins da Silva Nunes**  
**Professor: José Gabriel**  
**Data: 15/03/2016**

## **Lista 1 - Resolução**

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

## a) Usando integração por partes

A integração por partes é feita da seguinte forma:  $\int_a^b u dv = uv \Big|_a^b - \int_a^b v du$ . Chamando  $u = x$  e  $dv = e^{-x}$ , temos:

$$\begin{aligned}\int_0^1 xe^{-x} dx &= -xe^{-x} \Big|_0^1 + \int_0^1 e^{-x} dx \\&= -e^{-1} - e^{-x} \Big|_0^1 \\&= -e^{-1} - e^{-1} + 1 \\&= 1 - 2e^{-1} \\&\approx 0.2642\end{aligned}$$

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

O código em MATLAB, exibido a seguir, implementa a solução do item em questão.

```
N = 10;
x = rand(N,1);
sum(x.*exp(-x))/N

ans = 0.2749
```

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

O código em MATLAB, exibido a seguir, implementa a solução do item em questão.

```
N = 10;
x = random('exp', 1, N, 1);
sum(x(x < 1))/N
```

```
ans = 0.3494
```

**1\*. Calcular  $\int_0^1 x^2 e^{-x} dx$  de três formas diferentes:**

a) Usando integração por partes

Chamando  $u = x^2$  e  $dv = e^{-x}$ , temos:

$$\begin{aligned}\int_0^1 x^2 e^{-x} dx &= -x^2 e^{-x} \Big|_0^1 + \int_0^1 2x e^{-x} dx \\ &= -e^{-1} + 2(1 - 2e^{-1}) \\ &= 2 - 5e^{-1} \\ &\approx 0.1606\end{aligned}$$

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

O código em MATLAB, exibido a seguir, implementa a solução do item em questão.

```
N = 10;
x = rand(N,1);
sum((x.^2).*exp(-x))/N

ans = 0.1542
```

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

O código em MATLAB, exibido a seguir, implementa a solução do item em questão.

```
N = 10;
x = random('exp', 1, N, 1);
y = x.^2;
sum(y(y < 1))/N
```

```
ans = 0.1639
```

**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 obtido.**

O valor de pi pode ser encontrado através do cálculo da área de um círculo de raio unitário  $x^2 + y^2 \leq 1$ . Isolando a variável  $y$ , temos:  $y = \pm\sqrt{1 - x^2}$ . Devido à simetria da função, de modo a simplificar o cálculo, podemos considerar, somente, uma metade da circunferência e depois multiplicá-la por 2, para obter o valor correto. Sendo assim, uma forma de se calcular  $\pi$  é apresentada a seguir:

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

O método de Monte Carlo consiste em obter amostras de uma certa distribuição aleatória e, depois, calcular a média dos valores retornados pela função de interesse, quando ela recebe as amostras sorteadas. Essencialmente, o método de Monte Carlo executa uma aproximação do valor esperado de uma função  $f(x)$ , dado que seus valores são obtidos mediante o uso de amostras da variável aleatória  $p(x)$ . O valor esperado é dado por:

$$E[f(x)] = \int_{-\infty}^{+\infty} f(x)p(x)dx \quad (2)$$

Conforme dito anteriormente, como o método de Monte Carlo sorteia um número finito, e não infinito, de amostras, o cálculo da média é, na verdade, uma aproximação desse valor esperado. No caso dessa questão,  $p(x)$  é uma variável uniforme entre -1 e 1, dada por:

$$p(x) = \begin{cases} 1/2, & |x| \leq 1 \\ 0, & |x| > 1 \end{cases} \quad (3)$$

Substituindo (3) em (2):

$$\int_{-\infty}^{+\infty} f(x)p(x)dx = \int_{-1}^1 f(x)\frac{1}{2}dx \quad (4)$$

É preciso escolher a função  $f(x)$ , de modo que a Equação (4) fique igual à Equação (1). A igualdade é alcançada se fizermos  $f(x) = 4\sqrt{1 - x^2}$ . Portanto:

$$E[f(x)] = \int_{-1}^1 4\sqrt{1 - x^2}\frac{1}{2}dx = \pi \quad (5)$$

A aproximação de Monte Carlo do valor esperado será tão melhor quanto mais amostras forem utilizadas. Isso, de fato, se confirma, quando utilizamos  $N = 20$  amostras e  $N = 1000000$  amostras. O resultado obtido, após se efetuar esses cálculos, é muito mais próximo de  $\pi$  neste caso do que naquele. O código que efetua esse cálculo é exibido a seguir, tal como os resultados obtidos para  $N = 20$  e  $N = 1000000$ .

```

N = 20;
% N = 1000000;

x = random('unif', -1, 1, N, 1);

y = 4*sqrt(1-x.^2);

sum(y)/N % Média

% Resultado para N = 20
% ans = 3.2492

% Resultado para N = 1000000
% ans = 3.1415

```

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  $e^{\frac{-J(x)}{T}}$ , 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  $n = 100000$ .

Tendo em vista que a função  $J(x) = x^2$ , segue que a distribuição de estados será proporcional a  $e^{\frac{-x^2}{0,1}}$ , o que corresponde a uma Gaussiana, cuja média  $\mu = 0$  e a variância  $\sigma^2 = 0,05$ . Portanto, é esperado que, após um tempo consideravelmente grande (e, idealmente, infinito), os estados  $x(n)$  serão obtidos segundo uma distribuição que convergirá para essa Gaussiana, sendo os mais prováveis aqueles que contém uma baixa energia. O algoritmo executado encontra-se abaixo.

```

x_atual = random('norm', 0, sqrt(0.05));
epsilon = 0.1;

N = 100000;
x = zeros(1,N);
J = zeros(1,N);

for i = 1:N

    J_atual = x_atual ^ 2;
    x(i) = x_atual;
    J(i) = J_atual;

    r = random('unif', -1, 1);
    x_futuro = x_atual + epsilon * r;
    J_futuro = x_futuro ^ 2;
    J_dif = J_futuro - J_atual;

    if J_dif < 0
        x_atual = x_futuro;
    else
        beta = rand(1,1);
    end
end

```

```

        if beta < exp(-(J_dif)/0.1)
            x_atual = x_futuro;
        end
    end

% Plot dos gráficos

hist(x((N-10000):end), 100)
set(gca, 'FontSize', 30)
title('Histograma dos últimos 10000 estados x(n)', 'FontSize', 30)
disp(['Média dos últimos 10000 estados x(n): ' num2str(mean(x((N-10000):end)))])
disp(['Variância dos últimos 10000 estados x(n): ' num2str(var(x((N-10000):end)))])

% Valores encontrados para média e variância
%
% Média dos últimos 10000 estados x(n): -0.014273
% Variância dos últimos 10000 estados x(n): 0.050015

figure

hist(J((N-10000):end), 100)
set(gca, 'FontSize', 30)
title('Histograma dos últimos 10000 valores J(x)', 'FontSize', 30)

figure

plot(x, 'o ')
set(gca, 'FontSize', 30)
xlabel('n', 'FontSize', 30)
ylabel('x(n)', 'FontSize', 30)
title('Valores dos estados x(n)', 'FontSize', 30)

figure

plot(1:10000, x(1:10000), 'o ')
set(gca, 'FontSize', 30)
xlabel('n', 'FontSize', 30)
ylabel('x(n)', 'FontSize', 30)
title('Valores dos primeiros 10000 estados x(n)', 'FontSize', 30)
axis([1 10000 -0.8 0.8])

figure

```

```

plot(90000:100000, x((N-10000):end), 'o')
xlabel('n', 'FontSize', 30)
ylabel('x(n)', 'FontSize', 30)
title('Valores dos últimos 10000 estados x(n)', 'FontSize', 30)
axis([90000 100000 -0.8 0.8])

```

Os 3 gráficos presentes na Figura 1 mostra o valor dos estados  $x(n)$  em cada uma das 100000 execuções do algoritmo.

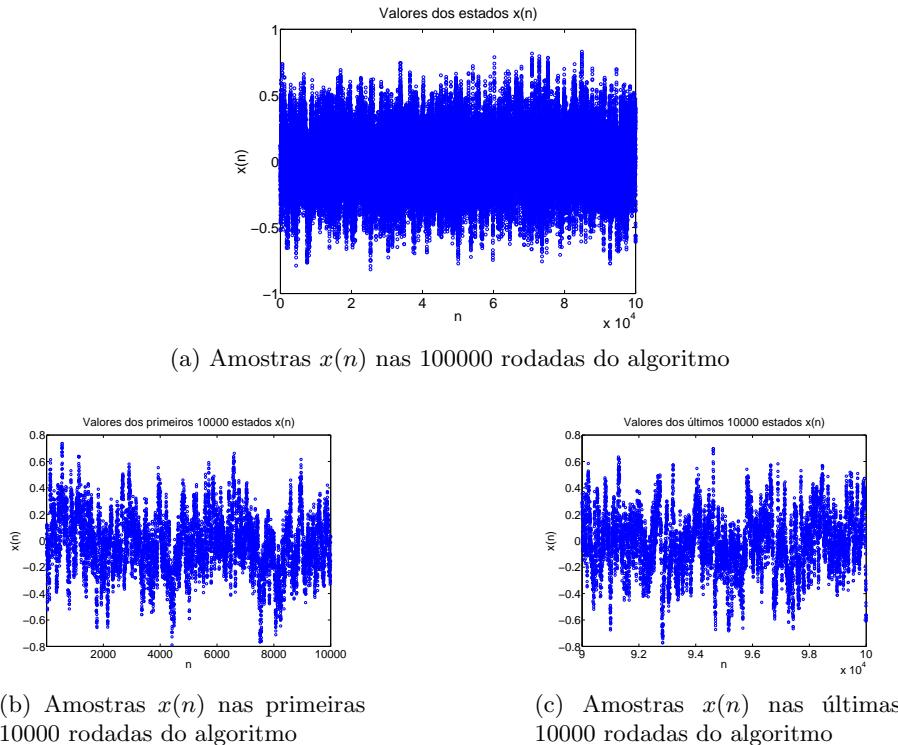


Figura 1: Valores das amostras  $x(n)$

A Figura 2 exibe o histograma dos estados  $x(n)$  referentes às 10000 últimas iterações do algoritmo. Como era de se esperar, observa-se que esse histograma se assemelha a uma Gaussiana. Calculando a média e a variância dessa distribuição, chega-se aos valores  $\mu = -0,014273$ , para a média, e  $\sigma^2 = 0,050015$ , para a variância, os quais são bem próximos aos valores da distribuição Gaussiana em questão ( $\mu = 0$  e  $\sigma^2 = 0,05$ ), mostrando que a distribuição dos estados está convergindo para essa Gaussiana. A Figura 3 exibe a distribuição dos valores da função  $J(x) = x^2$ , tendo os valores mais prováveis se concentrando no valor mínimo da função (que, no caso, é 0), como era de se esperar.

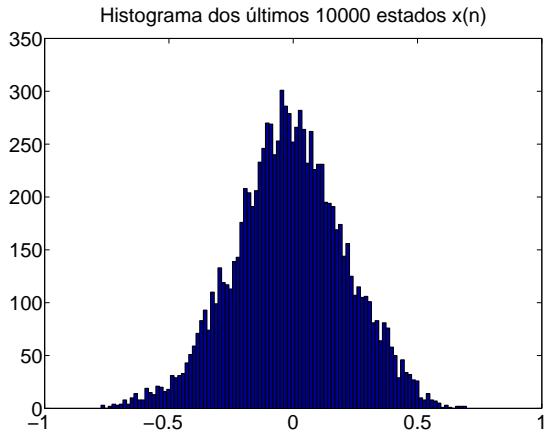


Figura 2: Histograma dos últimos 10000 valores dos estados  $x(n)$

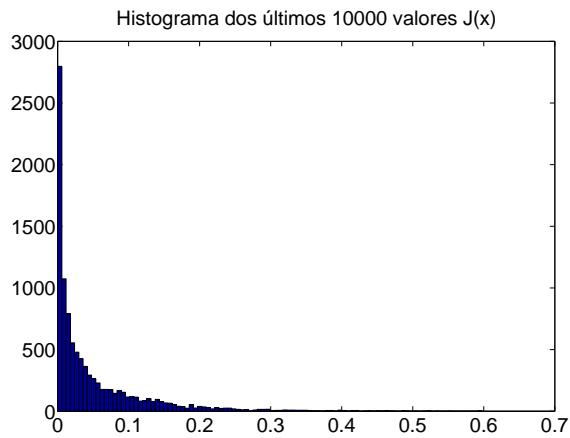


Figura 3: Histograma dos últimos 10000 valores da função  $J(x)$

**b) Execute manualmente (cálculos no papel) os 10 primeiros passos do algoritmo (ou seja, até  $n = 10$ ).**

- Escolhendo aleatoriamente um valor inicial  $x(0) = 1$  para a primeira iteração ( $n = 1$ ), temos:

1.  $J_0 = x(0)^2 = 1^2 = 1$
3. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = -0,4$
4.  $\hat{x}(1) = x(0) + 0,1 \times (-0,4) = 1 - 0,04 = 0,96$
5.  $\hat{J}_1 = \hat{x}(1)^2 = 0,96^2 = 0,9216$

6.  $\Delta J = \hat{J}_1 - J_0 = 0,9216 - 1 = -0,0784$
7. Como  $\Delta J < 0$ , a transição é aceita diretamente
8.  $J_1 = \hat{J}_1 = 0,9216$
9.  $x(1) = \hat{x}(1) = 0,96$

- Começa-se, então, a segunda iteração ( $n = 2$ ):

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = 0,1$
2.  $\hat{x}(2) = x(1) + 0,1 \times (0,1) = 0,96 + 0,01 = 0,97$
3.  $\hat{J}_2 = \hat{x}(2)^2 = 0,97^2 = 0,9409$
4.  $\Delta J = \hat{J}_2 - J_1 = 0,9409 - 0,9216 = 0,0193$
5. Como  $\Delta J > 0$ , escolhe-se aleatoriamente um número  $k$  entre 0 e 1, para decidir se a transição será aceita:  $k = 0,4$
6.  $e^{\frac{-\Delta J}{T}} = e^{\frac{-0,0193}{0,1}} = e^{-0,193} \approx 0,8245$
7. Como  $k = 0,4 < 0,8245 = e^{\frac{-\Delta J}{T}}$ , a transição é aceita.
8.  $J_2 = \hat{J}_2 = 0,9409$
9.  $x(2) = \hat{x}(2) = 0,97$

- Terceira iteração ( $n = 3$ ):

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = 0,5$
2.  $\hat{x}(3) = x(2) + 0,1 \times (0,5) = 0,97 + 0,05 = 1,02$
3.  $\hat{J}_3 = \hat{x}(3)^2 = 1,02^2 = 1,0404$
4.  $\Delta J = \hat{J}_3 - J_2 = 1,0404 - 0,9409 = 0,0995$
5. Como  $\Delta J > 0$ , escolhe-se aleatoriamente um número  $k$  entre 0 e 1, para decidir se a transição será aceita:  $k = 0,1$
6.  $e^{\frac{-\Delta J}{T}} = e^{\frac{-0,0995}{0,1}} = e^{-0,995} \approx 0,37$
7. Como  $k = 0,1 < 0,37 = e^{\frac{-\Delta J}{T}}$ , a transição é aceita.
8.  $J_3 = \hat{J}_3 = 1,0404$
9.  $x(3) = \hat{x}(3) = 1,02$

- Quarta iteração ( $n = 4$ ):

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = 0,3$
2.  $\hat{x}(4) = x(3) + 0,1 \times (0,3) = 1,02 + 0,03 = 1,05$
3.  $\hat{J}_4 = \hat{x}(4)^2 = 1,05^2 = 1,1025$
4.  $\Delta J = \hat{J}_4 - J_3 = 1,1025 - 1,0404 = 0,0621$
5. Como  $\Delta J > 0$ , escolhe-se aleatoriamente um número  $k$  entre 0 e 1, para decidir se a transição será aceita:  $k = 0,6$

$$6. e^{\frac{-\Delta J}{T}} = e^{\frac{-0,0621}{0,1}} = e^{-0,621} \approx 0,54$$

7. Como  $k = 0,6 > 0,54 = e^{\frac{-\Delta J}{T}}$ , a transição não é aceita.

$$8. J_4 = J_3 = 1,0404$$

$$9. x(4) = x(3) = 1,02$$

- Quinta iteração ( $n = 5$ ):

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = -0,8$

$$2. \hat{x}(5) = x(4) + 0,1 \times (-0,8) = 1,02 - 0,08 = 0,94$$

$$3. \hat{J}_5 = \hat{x}(5)^2 = 0,94^2 = 0,8836$$

$$4. \Delta J = \hat{J}_5 - J_4 = 0,8836 - 1,0404 = -0,1568$$

5. Como  $\Delta J < 0$ , a transição é aceita.

$$6. J_5 = \hat{J}_5 = 0,8836$$

$$7. x(5) = \hat{x}(5) = 0,94$$

- Sexta iteração ( $n = 6$ )

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = 0,7$

$$2. \hat{x}(6) = x(5) + 0,1 \times (0,7) = 0,94 + 0,07 = 1,01$$

$$3. \hat{J}_6 = \hat{x}(6)^2 = 1,01^2 = 1,0201$$

$$4. \Delta J = \hat{J}_6 - J_5 = 1,0201 - 0,8836 = 0,1365$$

5. Como  $\Delta J > 0$ , escolhe-se aleatoriamente um número  $k$  entre 0 e 1, para decidir se a transição será aceita:  $k = 0,3$

$$6. e^{\frac{-\Delta J}{T}} = e^{\frac{-0,1365}{0,1}} = e^{-1,365} \approx 0,26$$

7. Como  $k = 0,3 > 0,26 = e^{\frac{-\Delta J}{T}}$ , a transição não é aceita.

$$8. J_6 = J_5 = 0,8836$$

$$9. x(6) = x(5) = 0,94$$

- Sétima iteração ( $n = 7$ )

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = -0,1$

$$2. \hat{x}(7) = x(6) + 0,1 \times (-0,1) = 0,94 - 0,01 = 0,93$$

$$3. \hat{J}_7 = \hat{x}(7)^2 = 0,93^2 = 0,8649$$

$$4. \Delta J = \hat{J}_7 - J_6 = 0,8649 - 0,8836 = -0,0187$$

5. Como  $\Delta J < 0$ , a transição é aceita

$$6. J_7 = \hat{J}_7 = 0,8649$$

$$7. x(7) = \hat{x}(7) = 0,93$$

- Oitava iteração ( $n = 8$ )

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = -0,8$
2.  $\hat{x}(8) = x(7) + 0,1 \times (-0,8) = 0,93 - 0,08 = 0,85$
3.  $\hat{J}_8 = \hat{x}(8)^2 = 0,85^2 = 0,7225$
4.  $\Delta J = \hat{J}_8 - J_7 = 0,7225 - 0,8649 = -0,1424$
5. Como  $\Delta J < 0$ , a transição é aceita
6.  $J_8 = \hat{J}_8 = 0,7225$
7.  $x(8) = \hat{x}(8) = 0,85$

- Nona iteração ( $n = 9$ )

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = 0,8$
2.  $\hat{x}(9) = x(8) + 0,1 \times (0,8) = 0,93 + 0,08 = 1,01$
3.  $\hat{J}_9 = \hat{x}(9)^2 = 1,01^2 = 1,0201$
4.  $\Delta J = \hat{J}_9 - J_8 = 1,0201 - 0,7225 = 0,2976$
5. Como  $\Delta J > 0$ , escolhe-se aleatoriamente um número  $k$  entre 0 e 1, para decidir se a transição será aceita:  $k = 0,5$
6.  $e^{\frac{-\Delta J}{T}} = e^{\frac{-0,2976}{0,1}} = e^{-2,976} \approx 0,05$
7. Como  $k = 0,5 > 0,05 = e^{\frac{-\Delta J}{T}}$ , a transição não é aceita.
8.  $J_9 = J_8 = 0,7225$
9.  $x(9) = x(8) = 0,85$

- Décima iteração ( $n = 10$ )

1. Escolhe-se aleatoriamente um valor  $r$  entre -1 e 1:  $r = -0,6$
2.  $\hat{x}(10) = x(9) + 0,1 \times (-0,6) = 0,85 - 0,06 = 0,79$
3.  $\hat{J}_{10} = \hat{x}(10)^2 = 0,79^2 = 0,6241$
4.  $\Delta J = \hat{J}_{10} - J_9 = 0,6241 - 0,7225 = -0,0984$
5. Como  $\Delta J < 0$ , a transição é aceita
8.  $J_{10} = \hat{J}_{10} = 0,6241$
9.  $x(10) = \hat{x}(10) = 0,79$

**4. Escrever um programa 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.**

O pseudo-código do S.A, que implementa a solução dessa questão, encontra-se descrito a seguir:

```

 $x_i \leftarrow 0$ 
 $J_i \leftarrow -x_i + 100(x_i - 0.2)^2(x_i - 0.8)^2$ 
 $T0 \leftarrow 1$ 
 $k_{max} \leftarrow 9$  (quantidade de temperaturas que serão utilizadas)
 $N \leftarrow 100000$  (número de iterações no Algoritmo de Metropolis)

para  $k = 0 \dots k_{max}$ 
     $T = T0(0,9)^k$ 
    para  $i = 1 \dots N$ 
         $r \leftarrow$  amostra de uma variável aleatória simétrica (no caso, uniforme ou Gaussiana)
         $\hat{x}_{i+1} \leftarrow x_i + \epsilon \times r$ 
         $\hat{J}_{i+1} = -\hat{x}_{i+1} + 100(\hat{x}_{i+1} - 0.2)^2(\hat{x}_{i+1} - 0.8)^2$ 
         $\Delta J = \hat{J}_{i+1} - J_i$ 
        Se  $\Delta J < 0$ 
             $x_{\{i+1\}} = \hat{x}_{i+1}$ 
             $J_{\{i+1\}} = \hat{J}_{i+1}$ 
        Senão
             $\beta \leftarrow$  amostra de uma variável aleatória uniforme entre  $[0,1]$ 
            Se  $\beta < e^{\frac{-\Delta J}{T}}$ 
                 $x_{\{i+1\}} = \hat{x}_{i+1}$ 
                 $J_{\{i+1\}} = \hat{J}_{i+1}$ 
            Senão
                 $x_{\{i+1\}} = x_i$ 
                 $J_{\{i+1\}} = J_i$ 

```

```

        Fim Se
    Fim Se
Fim Loop

x_finais(k) = x_i (guarda os valores x_i da ultima
                    iteracao do Algoritmo de Metropolis)
k ← k + 1
Fim Loop

```

O código, em MATLAB, que minimiza a função  $J(x)$  é mostrado a seguir:

```

x_atual = 0;
epsilon = 0.1;

T0 = 1;
N = 100000;
x = zeros(1,N);
J = zeros(1,N);
J_min = inf;
X_min = inf;

X = [];
J_rodadas = [];
k = 0;
interrompe = 0;
indice_T = 1;

while (~interrompe)

    T_atual = (0.9^k) * T0;

    disp(['T: ' num2str(T_atual)])

    %      if T_atual < 0.1
    %          interrompe = 1;
    %      end

    if k > 9
        interrompe = 1;
    end

    T(indice_T) = T_atual;
    indice_T = indice_T + 1;

    if (~interrompe)

```

```

for i = 1:N

    x(i) = x_atual;
    J_atual = - x_atual + 100 * ((x_atual - 0.2).^2) .* ((x_atual - 0.8).^2);
    J(i) = J_atual;

    if J_atual < J_min
        J_min = J_atual;
        X_min = x_atual;
    end

    r = random('unif', -1, 1);
    % r = random('norm', 0, 1);
    x_futuro = x_atual + epsilon * r;
    J_futuro = - x_futuro + 100 * ((x_futuro - 0.2).^2) .* ((x_futuro - 0.8).^2);
    J_dif = J_futuro - J_atual;

    if J_dif < 0
        x_atual = x_futuro;
    else
        beta = rand(1,1);

        if beta < exp(-(J_dif)/T_atual)
            x_atual = x_futuro;
        end
    end
end

X = [X; x];
J_rodadas = [J_rodadas; J];

k = k + 1;
end

end

```

Os dez primeiros valores de  $x(n)$  podem ser obtidos trocando a condição de parada do loop da temperatura. Em vez de utilizar um valor de temperatura, utiliza-se o valor de  $k$  para essa condição. No caso, como  $k = 0$  inicialmente, o loop deve parar quando  $k > 9$ . Os primeiros 10 valores de  $x(n)$  estão exibidos a seguir, para cada uma das diferentes variáveis aleatórias utilizadas (uniforme e Gaussiana). Os 10 primeiros valores do algoritmo correspondem aos últimos

estados  $x(n)$  de cada uma das 10 temperaturas utilizadas.

```
% Resultados dos 10 primeiros x(n) usando variável aleatoria uniforme
%
% X(:,end)
%
% ans =
%
%    0.3041
%    0.1229
%    0.1448
%    0.8402
%    0.1417
%    0.6628
%    0.8097
%    0.7775
%    0.2803
%    0.7753

% Resultados dos 10 primeiros x(n) usando variável aleatoria Gaussiana
%
% X(:,end)
%
% ans =
%
%    0.9989
%    0.6901
%    0.7380
%    0.6671
%    0.7535
%    0.7659
%    0.8294
%    0.8342
%    0.4339
%    0.8992
```

**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:  $J(x) = [(x + 2)^2 - 3][10(x - 3)^2 + 5]$ . Seu ponto mínimo global ocorre em  $x_{min} \approx -2,49$  e vale  $J(x_{min}) \approx -845,64$ . O código e seu resultado são exibidos a seguir:

```

x_atual = 0;
epsilon = 0.1;

T0 = 1;
N = 100000;
x = zeros(1,N);
J = zeros(1,N);
J_min = inf;
X_min = inf;

X = [];
J_rodadas = [];
k = 0;
interrompe = 0;
indice_T = 1;

while (~interrompe)

    T_atual = (0.9^k) * T0;

    disp(['T: ' num2str(T_atual)])

    if T_atual < 0.1
        interrompe = 1;
    end

    T(indice_T) = T_atual;
    indice_T = indice_T + 1;

    if (~interrompe)

        for i = 1:N

            x(i) = x_atual;
            J_atual = (((x_atual + 2) ^ 2) - 3) * (10 * ((x_atual - 3) ^ 2) + 5);
            J(i) = J_atual;

            if J_atual < J_min
                J_min = J_atual;
                X_min = x_atual;
            end

            r = random('unif', -1, 1);
            x_futuro = x_atual + epsilon * r;
            J_futuro = (((x_futuro + 2) ^ 2) - 3) * (10 * ((x_futuro - 3) ^ 2) + 5);
            J_dif = J_futuro - J_atual;

    end
end

```

```

if J_dif < 0
    x_atual = x_futuro;
else
    beta = rand(1,1);

    if beta < exp(-(J_dif)/T_atual)
        x_atual = x_futuro;
    end
end
end

X = [X; x];
J_rodadas = [J_rodadas; J];

k = k + 1;
end

end

% Resultados do código
%
% J_min = -845.6411
% X_min = -2.4936

```

A Figura 4 mostra a distribuição das energias nas 10000 últimas iterações do algoritmo na última temperatura executada ( $T = 0.1094$ ). Como se observa, os valores que mais ocorreram foram aqueles próximos ao ponto de mínimo global da função.

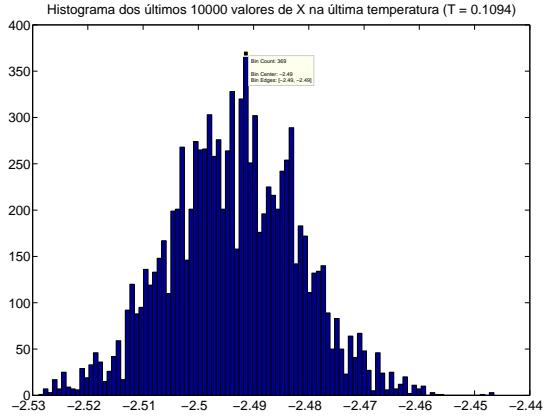


Figura 5: Distribuição dos estados  $x(n)$  nas últimas 10000 iterações da última temperatura ( $T \approx 0,1094$ )

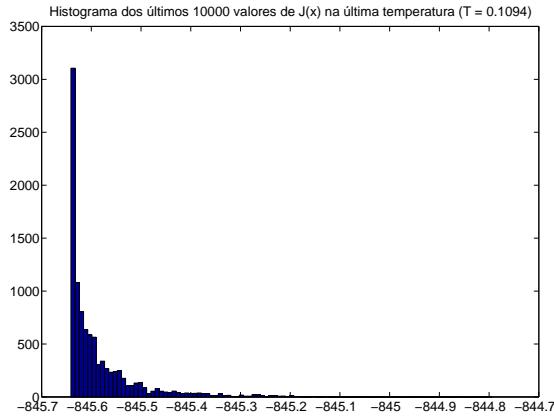


Figura 4: Distribuição dos valores da função  $J(x)$  nas últimas 10000 iterações da última temperatura  $T \approx 0,1094$

A Figura 5 mostra a distribuição de estados  $x(n)$ , também obtidos nas 10000 últimas iterações do algoritmo na última temperatura. Como era de se esperar, os valores mais observados se concentraram em torno da coordenada que resulta no ponto de mínimo da função. O pico desse histograma, inclusive, encontra-se em  $x = -2,49$ , que, de fato, é a coordenada do mínimo da função  $J(x)$ .