Exercício 01

Renan Salles de Freitas
CPE 723 - Otimização Natural

11 de março de 2018

Exercício 1.a.

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

Exercício 1.b. Segue o código fonte do exercício, em Matlab:

```
% Exercicio 1.1.b
x = rand(10,1);
s = sum(x.*exp(-x))/length(x);

x = [ 0.5228;
      0.0987;
      0.5349;
      0.9142;
      0.4430;
      0.3181;
      0.1843;
      0.8165;
      0.6085;
      0.7067];

s = 0.2789;
```

Exercício 2. Considere:

$$f(x,y) = \begin{cases} 1 & x^2+y^2 \leq 1 \\ 0 & x^2+y^2 \geq 1 \end{cases}$$

E considere o domínio $D = [-1,1] \times [-1,1]$. Temos que:

$$\int_D f(x,y)dxdy = \pi$$

Podemos estimar o valor de π pelo método de integração de Monte Carlo, escolhendo N números de maneira aleatória em D:

$$\pi \approx \frac{4}{N}\sum_{i=1}^{N} f(x_i, y_i)$$

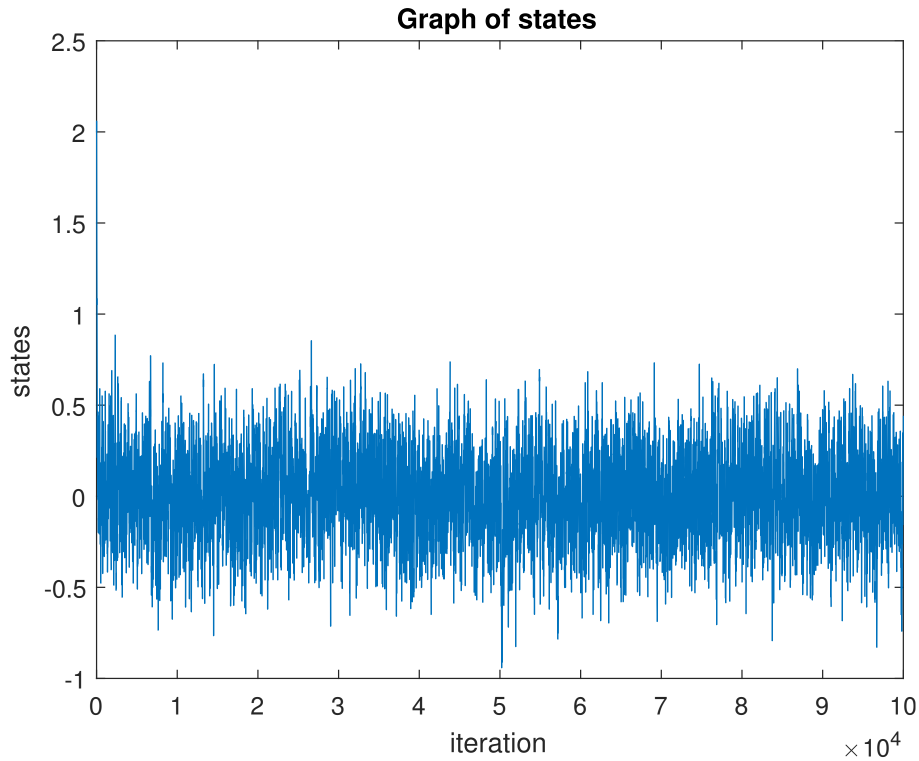
Usando $N = 20$, obtivemos como resultado $\pi = 2.8$. Usando $N = 1.000.000$, obtivemos $\pi = 3.1411$. O resultado se aproxima cada vez mais de π quanto maior for o valor de N. Esse resultado é provado pela lei dos grandes números: quanto mais tentativas são realizadas, mais a probabilidade da média aritmética dos resultados observados irá se aproximar da probabilidade real.

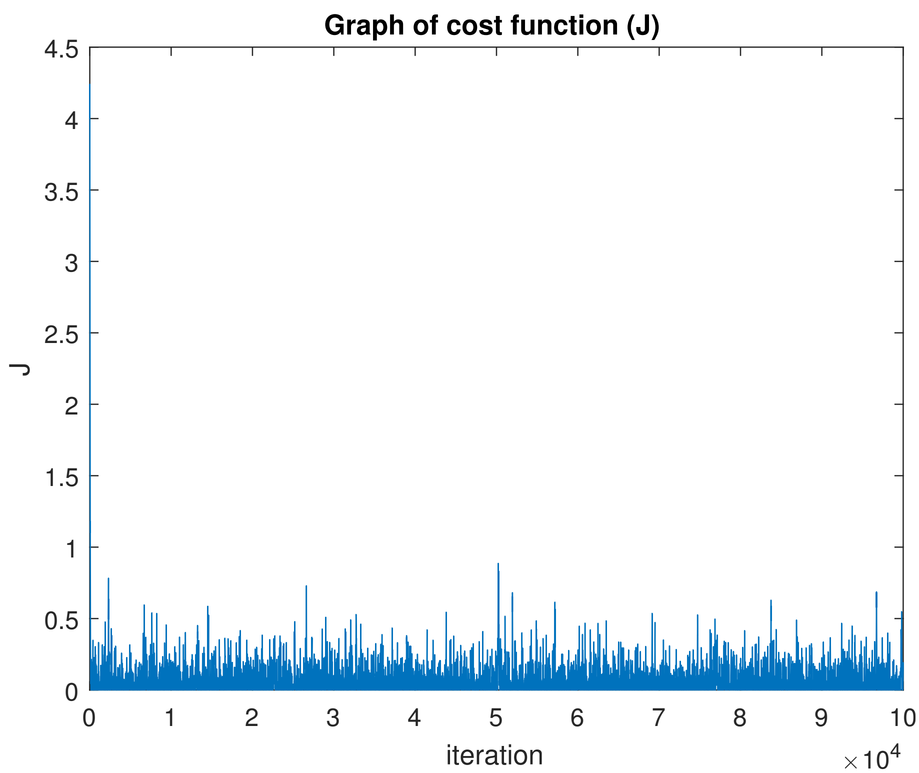
```
% Exercicio 2
a = -1;
b = 1;
xy = (b-a) * rand(20, 2) + a;
s = sum(xy.^2,2);
PI = 4 * length(s(s<1)) / length(s);

xy = (b-a) * rand(1000000, 2) + a;
s = sum(xy.^2,2);
PI_2 = 4 * length(s(s<1)) / length(s);
```

Exercício 3.a. Segue o código fonte do exercício, em Matlab:

```
% Exercicio 03

clear all
clc

x0 = 2;
number_of_iteration = 100000;
epsilon = 0.1;

x = zeros(number_of_iteration, 1);
x(1) = x0;

J = @(n) n.^2;

T = 0.1;
B = @(n) exp(-n/T);

counter = 1;
while counter < number_of_iteration
    r = 2 * rand - 1;
    xk = x(counter) + epsilon * r;
    dJ = J(xk) - J(x(counter));
    counter = counter + 1;
```

```
    if dJ < 0
        x(counter) = xk;
    else
        a = rand;
        if B(dJ) > a
            x(counter) = xk;
        else
            x(counter) = x(counter - 1);
        end
    end
end

figure
plot(1:length(x),x)
title('Graph of states')
xlabel('iteration')
ylabel('states')
path_e0 = strcat('../figs/ex3_','states','.eps');
print(path_e0,'-depsc2','-painters')

figure
plot(1:length(x),J(x))
title('Graph of cost function (J)')
xlabel('iteration')
ylabel('J')
path_e0 = strcat('../figs/ex3_','j','.eps');
print(path_e0,'-depsc2','-painters')
```


Graph of states
states
iteration
2.5
2
1.5
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
8
9
10
×10^4


Graph of cost function (J)
J
iteration
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
1
2
3
4
5
6
7
8
9
10
×10^4

Exercício 3.b. Resolução de dez iterações do algoritmo:

$$\begin{aligned} x_0 &= 2 \\ r &= -0.9091 \\ x_1 &= x_0 + 0.1 * (-0.9091) = 1.9091 \\ \Delta J &= x_1^2 - x_0^2 = -0.3554 < 0 \\ x_1 &= 1.9091 \end{aligned} \tag{1}$$

$$\begin{aligned} x_1 &= 1.9091 \\ r &= -0.3739 \\ x_2 &= x_1 + 0.1 * (-0.9975) = 1.8717 \\ \Delta J &= x_2^2 - x_1^2 = -0.1414 < 0 \\ x_2 &= 1.8717 \end{aligned} \tag{2}$$

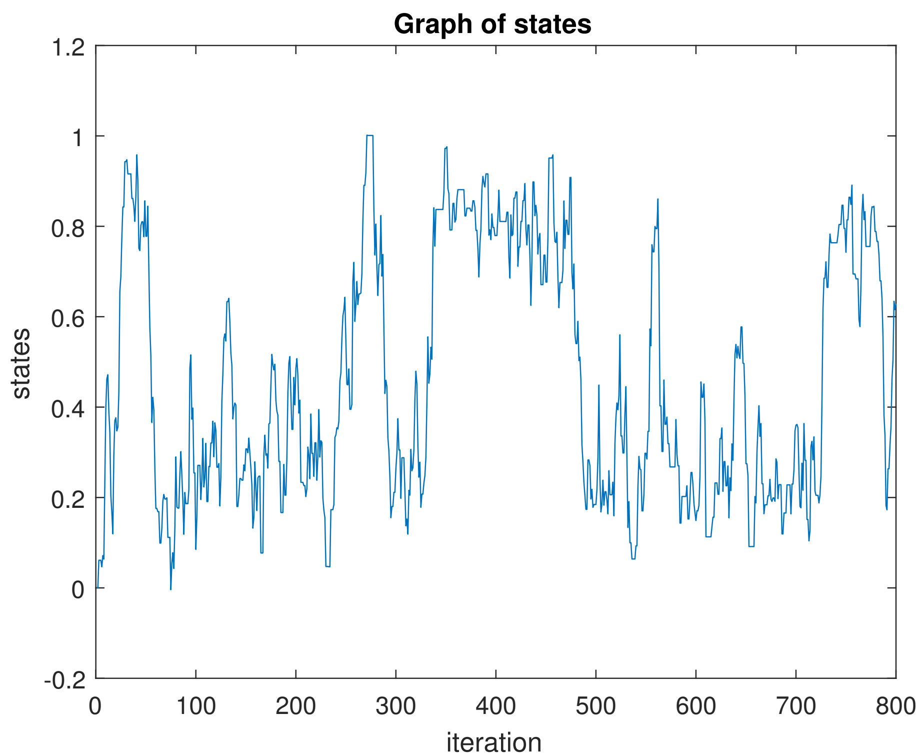
X_k	r	$\hat{X}$	ΔJ	$e^{-\frac{\Delta J}{T}}$	a	X_{k+1}
2	-0.9091	1.9091	-0.3554	-	-	1.9091
1.9091	-0.3739	1.8717	-0.1414	-	-	1.8717
1.8717	0.8264	1.9543	0.3162	0.0424	0.5054	1.8717
1.8717	0.8098	1.9527	0.3097	0.0452	0.1342	1.8717
1.8717	-0.3808	1.8336	-0.1411	-	-	1.8336
1.8336	-0.4297	1.7906	-0.1557	-	-	1.7906
1.7906	0.2195	1.8126	0.0791	0.4535	0.4008	1.8126
1.8126	0.2301	1.8356	0.0839	0.4320	0.9469	1.8126
1.8126	0.6654	1.8791	0.2456	0.0857	0.4103	1.8126
1.8126	0.0788	1.8205	0.0286	0.7509	0.0656	1.8205

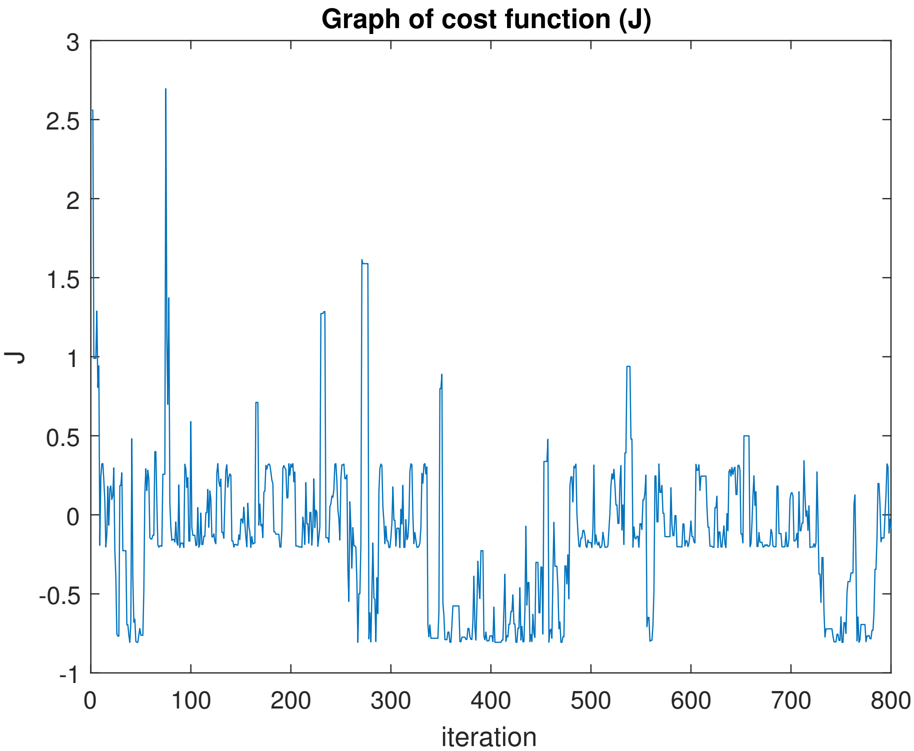
Exercício 4. A função custo (J) pode ser vista na figura 1. Segue o código fonte do exercício, em Matlab:

```
% Exercicio 04.b

clear all
clc

x0 = 0;
number_of_iteration = 100;
number_of_temperature_iteration = 8;
epsilon = 0.1;
T0 = 1;
T = T0;

x = zeros(number_of_iteration * number_of_temperature_iteration , 1);
```

```
x(1) = x0;

J = @(n) -n + 100 * (n - 0.2).^2 .* (n - 0.8).^2;

B = @(n,t) exp(-n/t);

counter = 1;
counter_temperatures = 1;
Jmin = J(x(counter));
xmin = x(counter);
while counter_temperatures <= number_of_temperature_iteration
    while counter < number_of_iteration * counter_temperatures
        r = randn;
        xk = x(counter) + epsilon * r;
        Jxk = J(xk);
        dJ = Jxk - J(x(counter));
        counter = counter + 1;
        if dJ < 0
            x(counter) = xk;
        else
            a = rand;
            if B(dJ,T) > a
                x(counter) = xk;
            else
                x(counter) = x(counter - 1);
            end
        end
        if Jxk < Jmin
            Jmin = Jxk;
            xmin = xk;
        end
    end
    counter_temperatures = counter_temperatures + 1;
    T = T0 / log2(counter_temperatures + 1);
end

figure
plot(1:length(x),x)
title('Graph of states')
xlabel('iteration')
ylabel('states')
path_e0 = strcat('../figs/ex4b_','states','.eps');
print(path_e0,'-depsc2','-painters')

figure
plot(1:length(x),J(x))
title('Graph of cost function (J)')
```

```
xlabel('iteration')
ylabel('J')
path_e0 = strcat('../figs/ex4b_','j','.eps');
print(path_e0,'-depsc2','-painters')
```




Graph of cost function (J)
J
iteration
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0
100
200
300
400
500
600
700
800

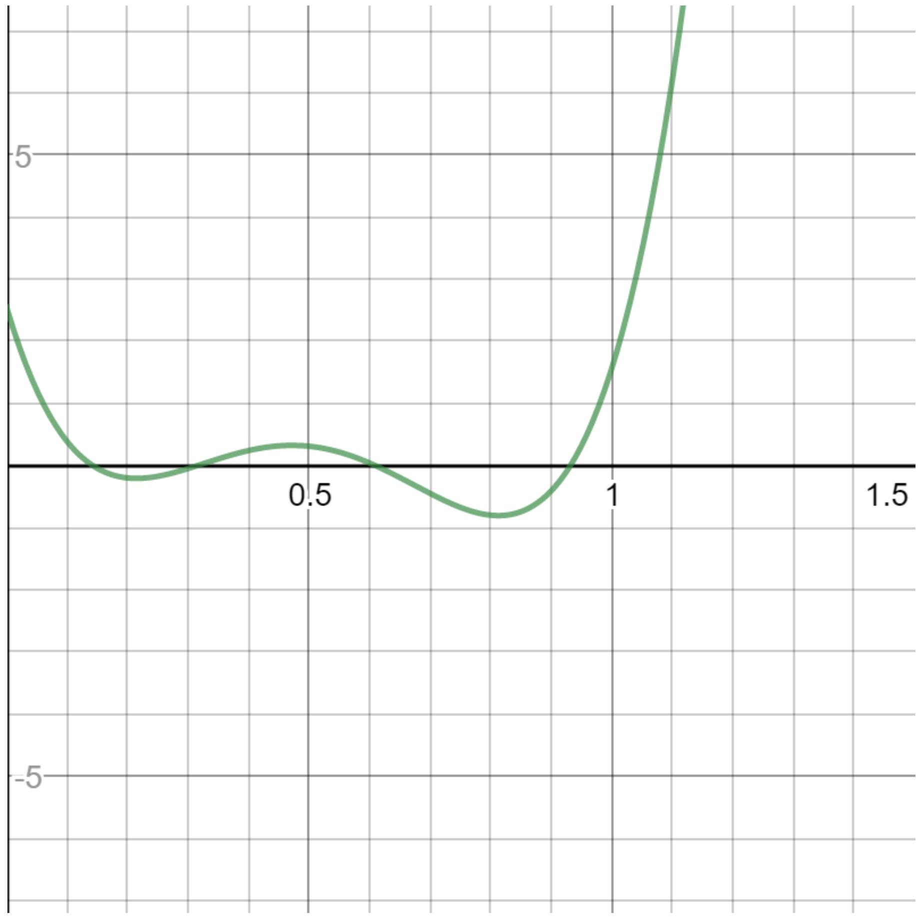


Figura 1: Gráfico da função J

Obtivemos $x_{\min} = 0.8135$ e $J(x_{\min}) = -0.8066$.
Calculando as primeiras iterações para $T = 1$ e $x_0 = 0$

X_k	r	$\hat{X}$	ΔJ	$e^{-\frac{\Delta J}{T}}$	a	X_{k+1}
0	-0.6576	-0.0658	2.7998	0.0608	0.1627	0
0	-0.7593	-0.0759	3.3574	0.0348	0.6375	0
0	0.8124	0.0812	-1.9126	-	-	0.0812
0.0812	0.0695	0.0882	-0.1023	-	-	0.0882
0.0882	-1.8337	-0.0952	6.5316	0.0015	0.8828	0.0882
0.0882	1.8274	0.2709	-0.6752	-	-	0.2709
0.2709	0.6541	0.3363	0.1934	0.8242	0.2174	0.3363
0.3363	-1.5448	0.1818	-0.2324	-	-	0.1818
0.1818	-0.3751	0.1443	0.1583	0.8536	0.7199	0.1443
0.1443	0.2077	0.1651	-0.1050	-	-	0.1651

Vale observar como o algoritmo oscila no mínimo local, mas pode alcançar o mínimo global, como na iteração 8.

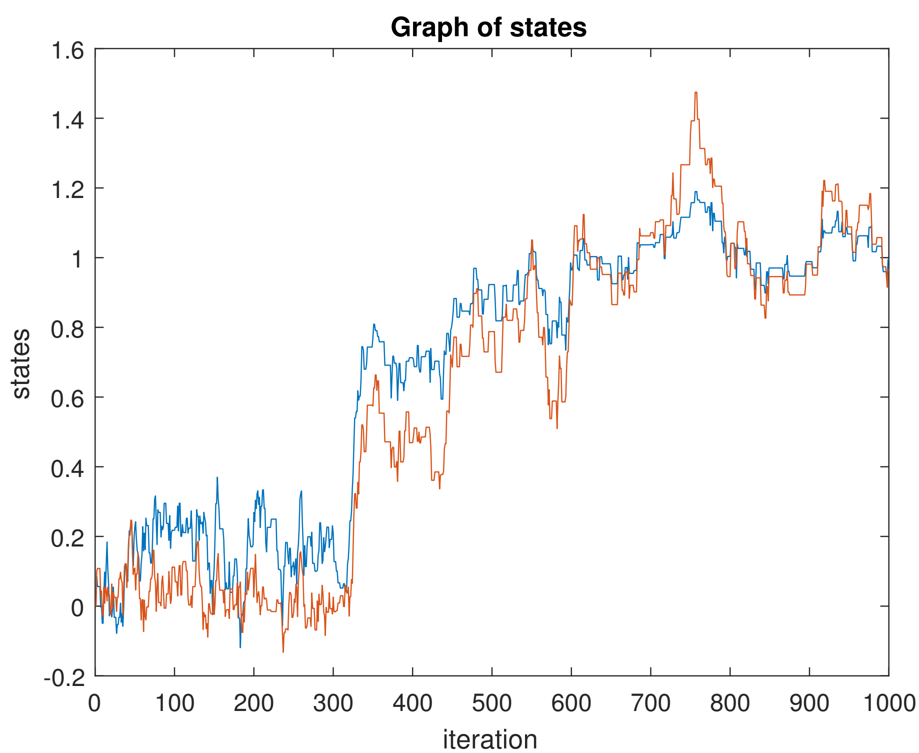
Exercício 5. A função que queremos minimizar é a Rosenbrock function, descrita com:

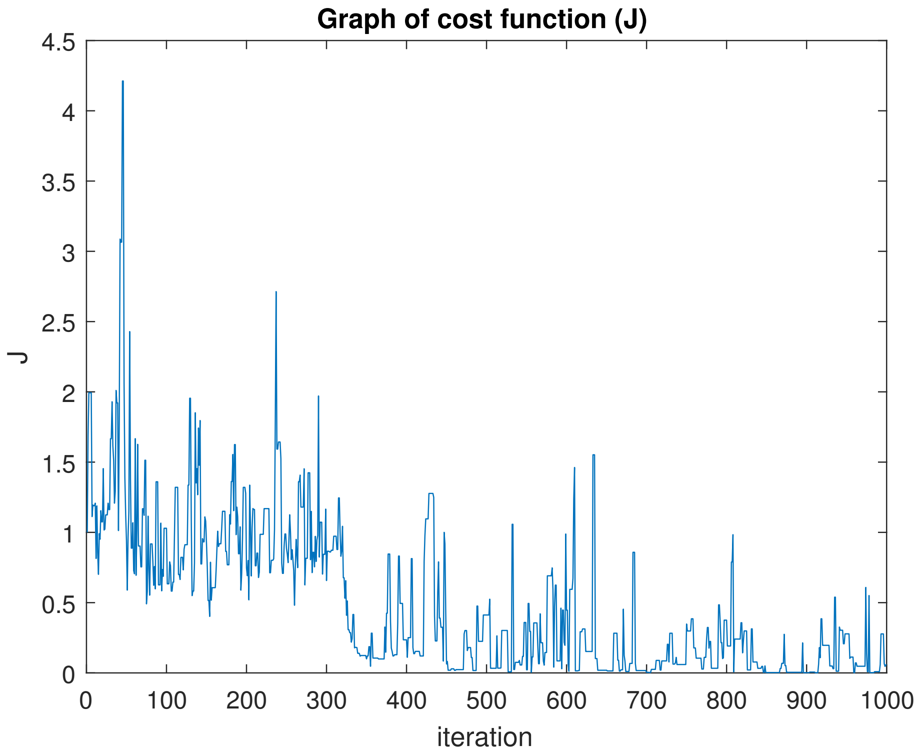
$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

O mínimo da função se encontra em $(x, y) = (1, 1)$ e $f(x, y) = 0$. Modificamos o algoritmo SA para vetores de duas dimensões, como pode ser visto abaixo:

```
% Exercicio 05

clear all
clc

x0 = [0, 0];
number_of_iteration = 100;
number_of_temperature_iteration = 10;
epsilon = 0.1;
T0 = 1;
T = T0;

x = zeros(number_of_iteration * number_of_temperature_iteration, 2);
x(1,:) = x0;

J = @(n) (1 - n(:,1)).^2 + 100 * (n(:,2) - n(:,1).^2).^2;

B = @(n,t) exp(-n/t);

counter = 1;
counter_temperatures = 1;
Jmin = J(x(counter,:));
xmin = x(counter,:);
while counter_temperatures <= number_of_temperature_iteration
    while counter < number_of_iteration * counter_temperatures
        r = 2 * rand(1,2) - 1;
        xk = x(counter,:) + epsilon * r;
        Jxk = J(xk);
        dJ = Jxk - J(x(counter,:));
        counter = counter + 1;
        if dJ < 0
            x(counter,:) = xk;
        else
            a = rand;
            if B(dJ,T) > a
                x(counter,:) = xk;
            else
                x(counter,:) = x(counter - 1,:);
            end
        end
        if Jxk < Jmin
            Jmin = Jxk;
```

```
            xmin = xk;
        end
    end
    counter_temperatures = counter_temperatures + 1;
    T = T0 / log2(counter_temperatures + 1);
end

figure
plot(1:length(x),x)
title('Graph of states')
xlabel('iteration')
ylabel('states')
path_e0 = strcat('../figs/ex5_','states','.eps');
print(path_e0,'-depsc2','-painters')

figure
plot(1:length(x),J(x))
title('Graph of cost function (J)')
xlabel('iteration')
ylabel('J')
path_e0 = strcat('../figs/ex5_','j','.eps');
print(path_e0,'-depsc2','-painters')
```





Obtivemos $x_{\min} = (1.0168, 1.0311)$ e $J(x_{\min}) = 0.001$.