

Exercício 04

Renan Salles de Freitas
CPE 723 - Otimização Natural

23 de abril de 2018

Exercício 1. O objetivo deste exercício é escrever um programa de algoritmo genético simples. O código de MatLab está abaixo:

```
1 clear all
2 clc
3
4 number_of_iteration = 100;
5 n = 1;
6 number_of_bits = 16;
7 population_size = 30;
8 xbimax = 2 ^ number_of_bits - 1;
9 pc = 0.7;
10 pm = 0.001;
11 jmax = 0;
12 xmax = 0;
13 xdemin = -2;
14 xdemax = 2;
15
16 pop = randi(2, population_size, number_of_bits) - 1;
17
18
19 while n < number_of_iteration
20     xr = mapbi2func(pop, xdemin, xdemax, xbimax);
21     j = -f(xr);
22
23     [jm, index] = max(j);
24     if jm > jmax
25         jmax = jm;
26         xmax = xr(index);
27     end
28
29     mating_pool = sus_selection(pop, j);
30     children = zeros(population_size, number_of_bits);
31
32     % Recombinacao
```

```

33     i = 1;
34     while i <= population_size
35         if rand < pc && i < population_size - 1
36             [children(i,:), children(i+1,:)] = ...
37                 crossover(mating_pool(i,:), mating_pool(i+1,:));
38             i = i + 2;
39         else
40             children(i,:) = mating_pool(i,:);
41             i = i + 1;
42         end
43     end
44
45     % Mutacao
46     for i = 1:population_size
47         for j = 1:length(children(i))
48             if rand < pm
49                 children(i,:) = mutation(children(i,:), j);
50             end
51         end
52     end
53
54     pop = children;
55     n = n + 1;
56
57 end

```

Como o código apresenta diversas funções e há várias maneiras de desenvolver o código, é necessário explicar como cada etapa do GA foi implementada. Como parâmetros iniciais do algoritmo, foi escolhida uma população inicial de 10 indivíduos de 16 bits, com bit de maior valor significativo na esquerda. O fenótipo é a conversão do genótipo em números reais entre -2 e 2 e avalia-se a função que se quer minimizar: $y = x^2 - 0.3\cos(10\pi x)$. O código MatLab para o fenótipo está abaixo:

```

1 function r = mapbi2func(x, xdemin, xdemax, xbimax)
2 xde = bi2dec(x);
3 r = (xdemax - xdemin) * xde / xbimax + xdemin;

```

```

1 function y = bi2dec(x)
2 x = flip(x);
3 y = bi2de(x);

```

```

1 function y = f(x)
2 y = x.^2 - 0.3 * cos(10 * pi * x);

```

Como é natural nos algoritmos genéticos, busca-se a minimização, portanto o valor da função é negativado (linha 21 do código do exercício 1). Na próxima etapa, guarda-se a melhor aptidão e o valor de x real correspondente.

Na seleção de pais, optou-se por utilizar o algoritmo *Amostragem Estocástica Universal* (SUS). O ranking é feito com probabilidade, de acordo com a aptidão. A implementação está abaixo:

```

1 function mating_pool = sus_selection(x, j)
2 [xsort, psort] = ranking(x, j);
3 number_of_bits = size(xsort, 2);
4 population_size = size(xsort, 1);
5 current_member = 1;
6 i=1;
7
8 mating_pool = zeros(population_size, number_of_bits);
9 psort = cumsum(psort);
10
11 r = rand / population_size;
12 while current_member <= population_size
13     while r <= psort(i)
14         mating_pool(current_member,:) = xsort(i,:);
15         r = r + 1/population_size;
16         current_member = current_member + 1;
17     end
18     i = i + 1;
19 end

```

```

1 function [xsort, psort] = ranking(x, j)
2 j = j + max(abs(j));
3 sumj = sum(j);
4 if sumj == 0
5     p = ones(length(j), 1) / length(j);
6 else
7     p = j / sum(j);
8 end
9 [psort, psort_index]=sort(p, 'descend');
10 xsort = x(psort_index,:);

```

A partir da geração de pais, a recombinação é feita se um valor randômico for menor que $p_c = 0.7$. Se isso acontecer, dois pais geram dois filhos por recombinação do tipo 1 ponto (onde há troca da cauda dos bits). Aqui, optou-se por representação binária Gray. Código abaixo:

```

1 function [nx1, nx2] = crossover(x1, x2)
2 % One point crossover
3 x1g = bi2gray(x1);
4 x2g = bi2gray(x2);
5 index = randi(length(x1));
6 temp = x1g(index:end);
7 x1g(index:end) = x2g(index:end);
8 x2g(index:end) = temp;
9 nx1 = gray2bi(x1g);

```

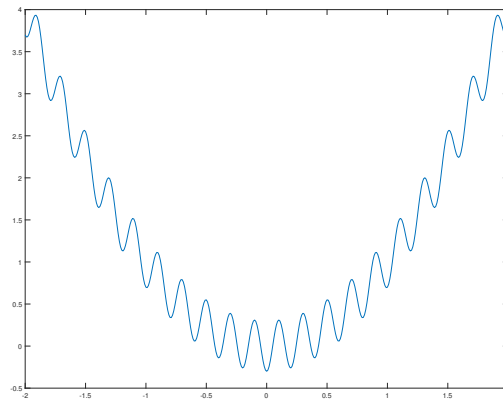
```
10 nx2 = gray2bi(x2g);
```

Caso o valor randômico não seja maior que p_c , o filho é uma cópia do pai.

Por fim, a mutação ocorre caso um valor randômico seja menor que $p_m = 0.001$. Aqui também optou-se pela codificação Gray, já que alterações de um bit representa números próximos em decimais. Código abaixo:

```
1 function y = mutation(x, index)
2 xg = bi2gray(x);
3 xg(index) = xor(xg(index), xg(index));
4 y = gray2bi(xg);
```

A função pode ser vista abaixo:



Com uma população inicial de 10 indivíduos e realizando 100 iterações, obtemos população final: $x = 0.0023$ e $f(x) = -0.2992$.

Exercício 2. Exercício semelhante ao anterior, porém mudando parâmetros de população inicial para 100, número de bits 25, função custo a ser maximizada (não minimizada) e, por se tratar de uma função que usa a variável binária, não é necessário utilizar a codificação Gray. Os códigos MatLab e gráficos são apresentados abaixo:

```
1 clear all
2 clc
3
4 number_of_iteration = 100;
5 number_of_bits = 25;
6 population_size = 100;
7 xbimax = 2 ^ number_of_bits - 1;
8 pc = 0.7;
9 pm = 1/number_of_bits;
10 jmax = [];
11 jmin = [];
12 jmean = [];
13
```

```

14 pop = randi(2, population_size, number_of_bits) - 1;
15
16 n = 1;
17 while n < number_of_iteration+1
18     j = f(pop);
19
20     jmax = [jmax ; max(j)];
21     jmin = [jmin ; min(j)];
22     jmean = [jmean ; mean(j)];
23
24     mating_pool = sus_selection(pop, j);
25     children = zeros(population_size, number_of_bits);
26
27     % Recombinacao
28     i = 1;
29     while i <= population_size
30         if rand < pc && i < population_size - 1
31             [children(i,:), children(i+1,:)] = ...
32                 crossover(mating_pool(i,:), mating_pool(i+1,:));
33             i = i + 2;
34         else
35             children(i,:) = mating_pool(i,:);
36             i = i + 1;
37         end
38     end
39
40     % Mutacao
41     for i = 1:population_size
42         for j = 1:length(children(i))
43             if rand < pm
44                 children(i,:) = mutation(children(i,:), j);
45             end
46         end
47     end
48
49     pop = children;
50     n = n + 1;
51
52 end

```

```

1 function y = f(x)
2 y = sum(x,2);

```

```

1 function mating_pool = sus_selection(x, j)
2 [xsort, psort] = ranking(x, j);
3 number_of_bits = size(xsort, 2);
4 population_size = size(xsort, 1);
5 current_member = 1;

```

```

6 i=1;
7
8 mating_pool = zeros(population_size, number_of_bits);
9 psort = cumsum(psort);
10
11 r = rand / population_size;
12 while current_member <= population_size
13     while r <= psort(i)
14         mating_pool(current_member,:) = xsort(i,:);
15         r = r + 1/population_size;
16         current_member = current_member + 1;
17     end
18     i = i + 1;
19 end

```

```

1 function [xsort, psort] = ranking(x, j)
2 sumj = sum(j);
3 if sumj == 0
4     p = ones(length(j), 1) / length(j);
5 else
6     p = j / sum(j);
7 end
8 [psort, psort_index]=sort(p, 'descend');
9 xsort = x(psort_index,:);

```

```

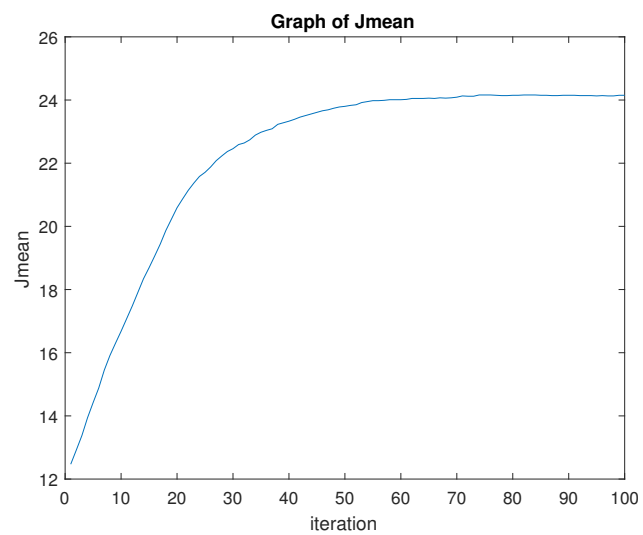
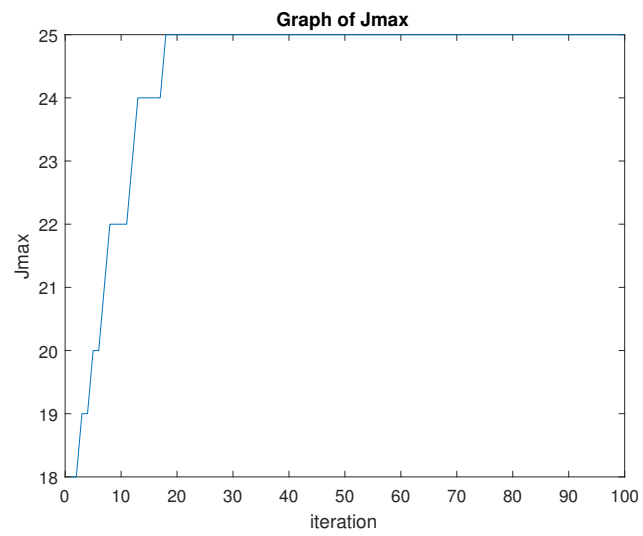
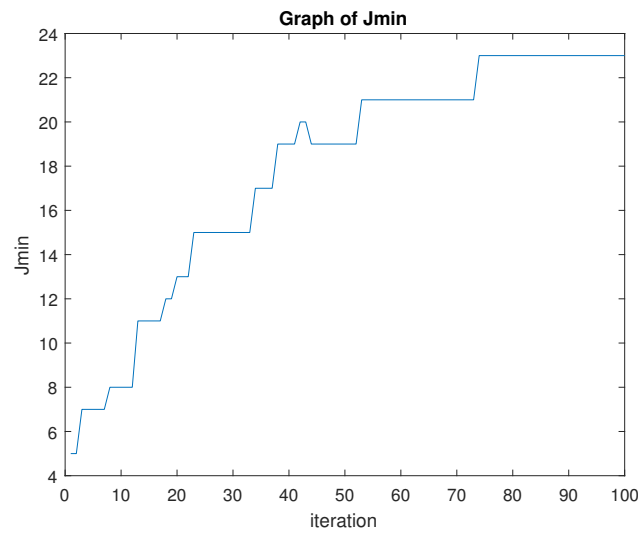
1 function [x1, x2] = crossover(x1, x2)
2 % One point crossover
3 index = randi(length(x1));
4 temp = x1(index:end);
5 x1(index:end) = x2(index:end);
6 x2(index:end) = temp;

```

```

1 function y = mutation(x, index)
2 x(index) = xor(x(index), x(index));
3 y = x;

```



A segunda parte do problema é rodar o algoritmo 10 vezes e verificar a média e desvio padrão de iterações para o programa achar o ótimo global. Executou-se o código MatLab abaixo:

```
1 clear all
2 clc
3
4 time_to_optimum = [];
5 for program_iteration = 1:10
6     number_of_iteration = 100;
7     number_of_bits = 25;
8     population_size = 100;
9     xbimax = 2 ^ number_of_bits - 1;
10    pc = 0.7;
11    pm = 1/number_of_bits;
12    jmax = [0];
13    jmin = [];
14    jmean = [];
15
16    pop = randi(2, population_size, number_of_bits) - 1;
17
18
19    n = 0;
20    while max(jmax) < 25
21
22        n = n + 1;
23        j = f(pop);
24
25        jmax = [jmax ; max(j)];
26        jmin = [jmin ; min(j)];
27        jmean = [jmean ; mean(j)];
28
29        mating_pool = sus_selection(pop, j);
30        children = zeros(population_size, number_of_bits);
31
32        % Recombinacao
33        i = 1;
34        while i <= population_size
35            if rand < pc && i < population_size - 1
36                [children(i,:), children(i+1,:)] = ...
37                    crossover(mating_pool(i,:), mating_pool(i+1,:));
38                i = i + 2;
39            else
40                children(i,:) = mating_pool(i,:);
41                i = i + 1;
42            end
43        end
44    end
```



```

45         % Mutacao
46         for i = 1:population_size
47             for j = 1:length(children(i))
48                 if rand < pm
49                     children(i,:) = mutation(children(i,:), j);
50                 end
51             end
52         end
53
54         pop = children;
55
56         end
57         time_to_optimum = [time_to_optimum;n];
58     end
59
60     mean(time_to_optimum)
61     std(time_to_optimum)

```

E foi encontrada média 19.2 e desvio padrão 2.7406.