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: 10/05/2016

Lista 5 - Resolução

Questão 1

***Capítulo 5, Exercício 7*:**

Implement an EP for the Ackley function with $n = 30$. Make 100 runs, storing the best value found in each, and then calculate the mean and standard deviation of these values. Compare your results with those from exercises 5 and 6 in Chap. 4.

O código, feito em MATLAB, que implementa a solução da questão, encontra-se abaixo.

```
clear all; close all; clc

N_pop = 200; % Tamanho da população
N_avaliacoes = 200000; % Quantidade de avaliações da função de
    Ackley
N_ger = N_avaliacoes/N_pop; % Número de gerações avaliadas
n = 30; % Dimensões da função de Ackley
epsilon = 0.02; % Tamanho mínimo do passo de alteração
eta = 6; % Passo de alteração inicial
N_exec = 100; % Número de execuções do algoritmo
alpha = 0.2; % Constante de alteração do passo
q = 10; % Quantidade de vezes que uma solução participará do
    torneio
melhor_fitness_execucao = ones(1, N_exec); % Melhores fitness
    encontradas em cada execução do algoritmo
geracao_otima_execucao = zeros(1, N_exec); % Primeira geração em
    que apareceu a melhor fitness em cada execução do algoritmo

for t = 1:N_exec

    ger = 1; % Geração atual
    melhores_fitness = ones(1, N_ger); % Melhores fitness por geraç
        ão
    P = [unifrnd(-30,30,n,N_pop); eta*ones(n,N_pop)]; % Inicializaç
        ão aleatória da população

    % Cálculo do fitness

    fitness = -20 * exp(-0.2 * sqrt((1/n) * sum(P(1:n,:).^2, 1))) -
         exp((1/n) * sum(cos(2 * pi * P(1:n,:)), 1)) + 20 + exp(1);
         % Fitness por indivíduos

    melhores_fitness(ger) = min(fitness);

    while (ger <= N_ger) && (melhores_fitness(ger) > 1e-7)

        % Mutação

        P_filhos = zeros(size(P));

        passos = P((n+1):end, :);
```

```
        passos = passos .* (1 + alpha * (ones(n, 1) * randn(1,
            N_pop))); % Mutação dos passos;
        passos(passos < epsilon) = epsilon;

        P_filhos((n+1):end, :) = passos;

        P_filhos(1:n, :) = P(1:n, :) + passos .* randn(n, N_pop);

        % Seleção de sobreviventes

        P_torneio = [P P_filhos]; % Monta a população de filhos +
            pai para o torneio
        fitness = -20 * exp(-0.2 * sqrt((1/n) * sum(P_torneio(1:n
            ,:).^2, 1))) - exp((1/n) * sum(cos(2 * pi * P_torneio
            (1:n,:)), 1)) + 20 + exp(1); % Fitness dos
            participantes
        resultados = zeros(1,size(P_torneio, 2)); % Resultados do
            torneio;
        qtd_participacoes = zeros(1,size(P_torneio, 2)); % Contagem
             de participações de cada indivíduo no torneio

        for p1 = 1:length(resultados)
            while (qtd_participacoes(p1) ~= q)

                qtd_participacoes(p1) = qtd_participacoes(p1) + 1;

                competidores_disponiveis = find(qtd_participacoes ~
                    = q);
                p2 = unidrnd(size(P_torneio, 2));

                if length(competidores_disponiveis) > 1 % Há mais
                    de um competidor disponível
                    while (p2 == p1) || (qtd_participacoes(p2) == q
                        ) % Evita sortear o mesmo participante ou
                        um participante que já competiu o número má
                        ximo de vezes no torneio
                        p2 = unidrnd(size(P_torneio, 2));
                    end

                    qtd_participacoes(p2) = qtd_participacoes(p2) +
                         1;

                    if fitness(p1) < fitness(p2)
                        resultados(p1) = resultados(p1) + 3;
                    else if fitness(p1) == fitness(p2)
                            resultados(p1) = resultados(p1) + 1;
                            resultados(p2) = resultados(p2) + 1;
                        else
                            resultados(p2) = resultados(p2) + 3;
                        end
                    end

                else if length(competidores_disponiveis) == 1 %
                    Sobrou somente 1 competidor; aceita sortear
                    outro que já competiu 'q' vezes, porém não o
                    pontua
                    while (p2 == p1) % Sorteia outro competidor,
```

```
                        que não o próprio
                        p2 = unidrnd(size(P_torneio, 2));
                    end

                    if fitness(p1) < fitness(p2)
                        resultados(p1) = resultados(p1) + 3;
                    else if fitness(p1) == fitness(p2)
                            resultados(p1) = resultados(p1) + 1;
                            resultados(p2) = resultados(p2) + 1;
                        end
                    end
                    end
                end
            end
        end

        idx = zeros(1, N_pop); % Índice que identifica os indiví
            duos melhor classificados

        for i = 1:N_pop % Salva os N_pop indivíduos melhores
            classificados no torneio para a próxima geração
            idx(i) = find(resultados == max(resultados), 1);
            P(:,i) = P_torneio(:,idx(i));
            resultados(idx(i)) = -Inf;
        end

        ger = ger + 1;

        fitness = fitness(idx);
        melhores_fitness(ger) = min(fitness);

    end

    melhor_fitness_execucao(t) = min(melhores_fitness);
    geracao_otima_execucao(t) = find(melhores_fitness == min(
        melhores_fitness), 1);
end
```

A Figura 1 exibe um gráfico das melhores fitness encontradas em cada execução independente do algoritmo, enquanto que a Figura 2 mostra o seu histograma. Vale ressaltar que, embora a melhor fitness corresponda a 0 (que é o valor mínimo da função de Ackley), devido a erros de aproximação do programa, o critério de parada não incluiu o ponto $(x) = 0$, mas sim, uma distância máxima, da qual a solução, considerada ótima, deve se encontrar de tal ponto (no caso, abaixo de 1×10^{-7}). Embora o histograma sugira que as soluções ótimas tenham sido encontradas diversas vezes, o gráfico revela que a melhor fitness (ou seja, a de menor valor) é muito maior do que o valor estipulado na condição de parada. De fato, o menor valor encontrado ao longo das 100 execuções foi de 0.0659. Nesse sentido, considera-se que em nenhuma das execuções a solução "ótima" foi encontrada (apesar que a distância considerada foi arbitrária; se tivesse sido considerado, por exemplo, que valores abaixo de 0.1 já seriam suficientemente ótimos, então o programa teria obtido êxito). É interessante ressaltar que na

maioria dos casos, a melhor solução teve valor acima de 1, o qual é distante do ótimo global. Em média, a melhor fitness (a de menor valor) encontrada é de 1.2818, com desvio-padrão de 0.8135.

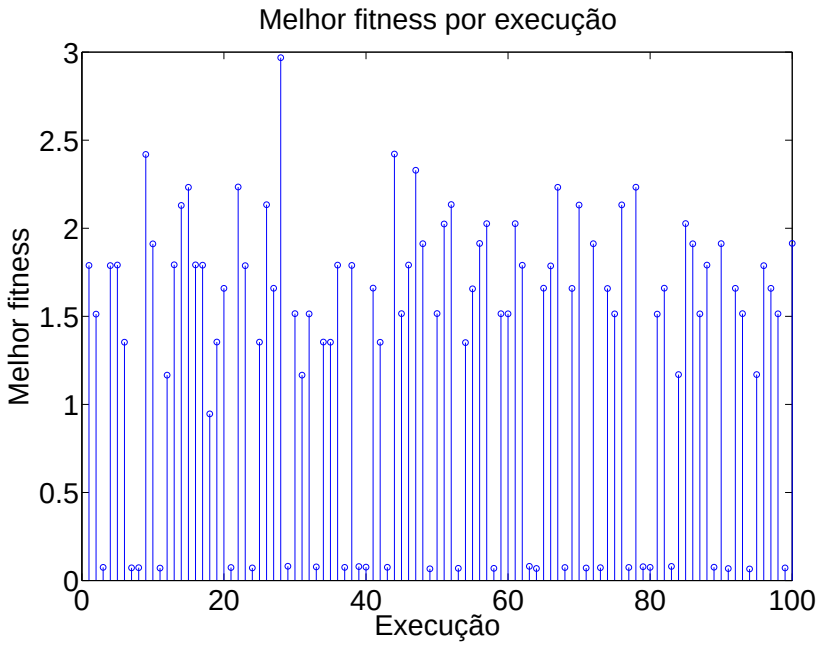

Figura 1: Gráfico das melhores fitness ao longo de 100 execuções do algoritmo

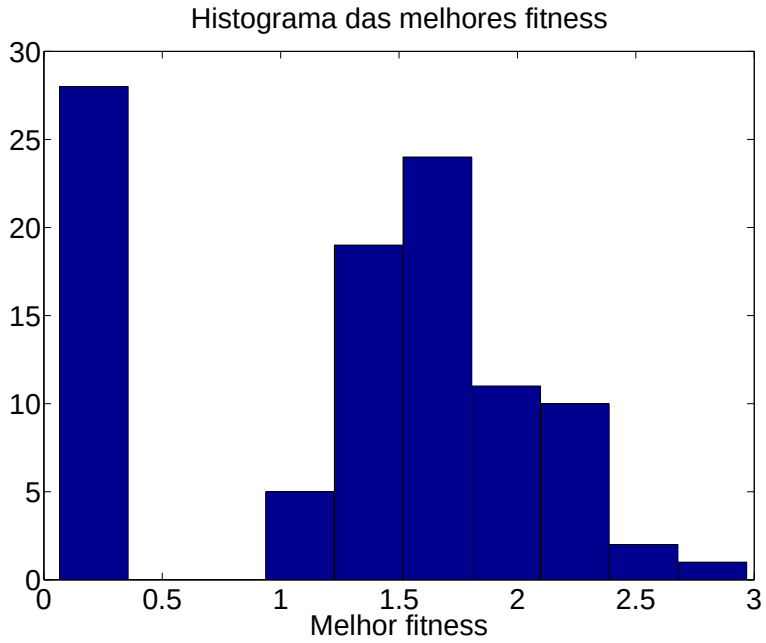

Figura 2: Histograma das melhores fitness encontradas ao longo de 100 execuções do algoritmo

A Figura 3 mostra as gerações em que as melhores soluções de cada execução apareceram pela primeira vez e a Figura 4 exibe seu histograma. A análise do histograma mostra que em grande parte das execuções, a melhor solução foi alcançada após muitas gerações terem sido consideradas (inclusive, próximas do limite imposto de 1000 gerações). O gráfico mostra que a quantidade de gerações mínima, necessária para se encontrar a melhor solução, foi de aproximadamente 600 (mais precisamente, 579). Em média, foram consideradas 841 gerações, com desvio padrão de 112. Considerando que nenhuma das soluções encontradas ficou abaixo do limiar estabelecido e que um número relativamente elevado de gerações foi necessário para se alcançar a melhor solução da execução (que não é a ótima), indica-se que um número maior de gerações deva ser considerado, na busca por solução, de modo a se tentar obter mais soluções próximas do mínimo buscado.

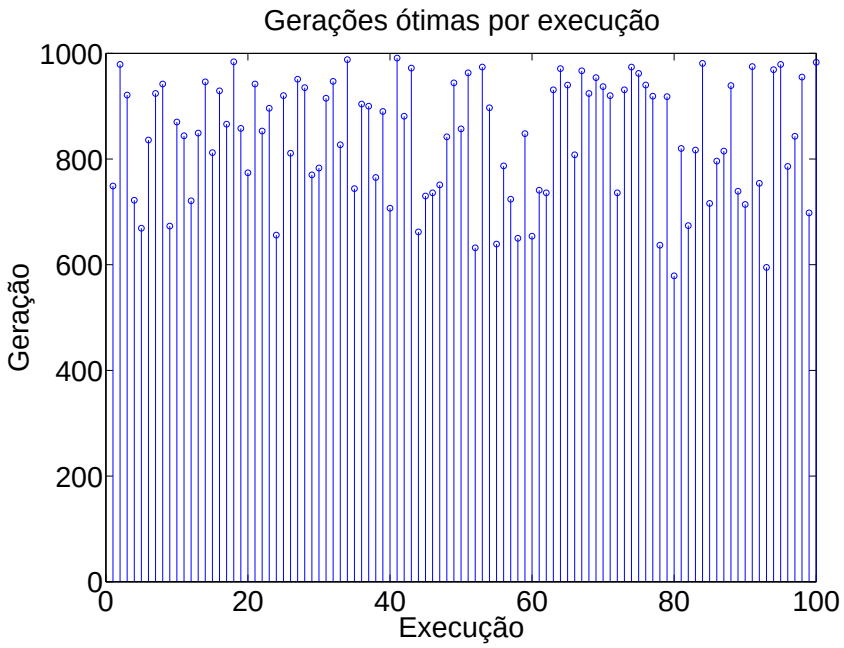

Figura 3: Gráfico das gerações, nas quais as melhores soluções da execução em questão foram encontradas pela primeira vez

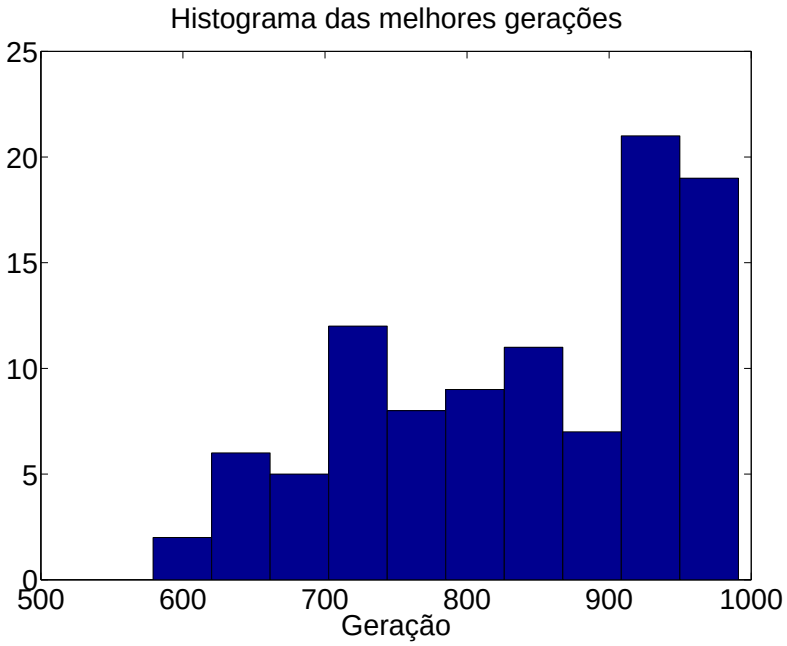

Figura 4: Histograma das gerações nas quais a melhor solução foi encontrada pela primeira vez ao longo das 100 execuções do algoritmo

Questão 2

***Capítulo 14, Exercício 1*:**

Consider using the number of generations as a measure to establish the speed of an EA. Compare the use of this measure with using the number of fitness evaluations.

Embora ambas sejam métricas válidas para se fazer uma avaliação da velocidade de execução de um algoritmo evolucionário, é preciso tomar cuidado em sua utilização na hora de se fazer comparações entre dois algoritmos. Considere, por exemplo, dois algoritmos, os quais utilizam como critério de parada a obtenção da solução ótima. O primeiro tem um processamento por geração mais lento (por conta, por exemplo, dos critérios de recombinação ou mutação selecionados), porém, em contrapartida, resulta em soluções bem próximas à ótima, ao passo que o segundo tem um processamento por geração mais rápido e resulta em soluções que se aproximam mais devagar da ótima. Devido à implementação dos algoritmos, supondo que ambos tenham alcançado a solução ótima, se o número de gerações fosse utilizado como critério para avaliar a velocidade de

cada um, ter-se-ia a falsa impressão de que o primeiro é mais rápido que o segundo, por utilizar um número menor de gerações. Entretanto, na prática, por ter um processamento mais lento por geração, o tempo de execução de ambos pode ter sido semelhante, embora o segundo tenha precisado de mais gerações (compensado pelo fato de o processamento por geração ser mais rápido). Portanto, diversos outros fatores dos algoritmos podem impactar nessa medida e devem ser considerados.

A utilização do número de avaliações da função de fitness tem a vantagem de retratar a capacidade do algoritmo em avaliar diferentes soluções para uma dada configuração do problema. Quanto mais rápido ele conseguir realizar cada avaliação, por exemplo, para um dado tamanho de população, mais soluções serão analisadas e, consequentemente, maiores são suas chances de encontrar a ótima. Por outro lado, caso a solução do problema não seja encontrada, essa métrica pode ser enganosa, tendo sido observado um alto número de avaliações, sugerindo uma alta velocidade, porém, não tendo êxito em resolver o problema. Nesse sentido, a utilização do número de gerações daria uma noção melhor do tempo de execução do algoritmo, já que um número elevado de gerações sugeriria uma demora do algoritmo em obter (caso alcance) a solução ótima. Entretanto, o número de gerações também tem a desvantagem de depender muito de outras etapas do algoritmo. Por exemplo, supondo que a inicialização da população tenha sido tal, que a solução ótima se encontrasse já na primeira geração. Essa métrica sugeriria uma velocidade do algoritmo muito distante da real, já que, circunstancialmente, outros fatores influenciaram essa convergência. Essas são algumas vantagens e desvantagens de cada uma dessas métricas. Vale lembrar que, por conta da estocacidade dos algoritmos genéticos, ambas as métricas só fazem sentido levando-se em conta suas propriedades estatísticas ao longo de diversas execuções independentes dos algoritmos a serem testados. Em todo caso, essas são algumas vantagens e desvantagens associadas a cada uma, as quais devem estar em mente, quando uma ou outra for escolhida como medida de desempenho.

Questão 3

***Capítulo 8, Exercício 1*:**

Give arguments why mutation strength (e.g., p_m or σ) should be increased during a run. Give arguments why it should be decreased.

A adaptação da taxa e da magnitude do passo de mutação pode favorecer na busca de soluções, feita pelo algoritmo genético, melhorando seu desempenho, não só em relação ao tempo de execução do algoritmo (quanto tempo ele

leva para encontrar uma solução boa), mas também em relação à qualidade das soluções encontradas. Com isso, cabe a pergunta de quando (e como) tais taxas devem variar. Sendo assim, analisa-se, primeiro, quando elas devem mudar.

Aumentar esses parâmetros podem ser feitos, quando o resultado das mutações tem um efeito mais positivo do que degenerativo na população final, ou seja, a maioria dos indivíduos gerados por meio de mutação apresentam aptidões melhores do que os indivíduos anteriores. O argumento para tal é que se esse fato é observado, então, no espaço de soluções, os indivíduos da população atual encontram-se distantes das soluções ótimas (tanto global quanto locais). Sendo assim, aumentar o valor desses parâmetros, isto é, aumentar a taxa de mutação e/ou a magnitude do passo faz com que os indivíduos deixem essas regiões subótimas mais rapidamente (analogamente, que se aproximem das regiões melhores mais rapidamente), evitando-se gastar muito tempo em regiões, nas quais as soluções estão longe das desejadas.

Valores elevados desses parâmetros podem ter efeitos contrários, quando a população se situa próxima a um ótimo, que pode ser local ou global. Nesse caso, é interessante que tais valores sejam reduzidos, de forma a se realizar um ajuste mais fino dos indíviduos sobre a região do espaço de soluções em que se encontram, permitindo que a solução ótima seja, eventualmente, atingida. Estes são os casos, nos quais a solução ótima está próxima das soluções representadas pelos indivíduos em questão. Por um lado, a redução no valor de tais parâmetros é válida, pois, supondo que a população está convergindo para a solução ótima, mutações frequentes ou passos largos tenderiam a gerar mais indivíduos menos aptos do que indivíduos mais aptos, o que não é desejado. Por outro lado, caso não se esteja no ótimo global, o algoritmo ficará preso na região do ótimo local. Sendo assim, ainda que a redução do valor dos parâmetros de mutação seja benéfica, quando a mutação rende, em sua maioria, indivíduos menos aptos, é preciso garantir uma certa diversidade da população, de modo a permitir que o algoritmo escape de ótimos locais (e possa, eventualmente, atingir a região do ótimo global). Garantindo essa diversidade, pode-se aplicar a regra proposta para aumento/redução nos valores dos parâmetros de mutação: maiores, para indivíduos longe da região ótima, e menores, para indivíduos próximos da região ótima.

Questão 4

Considere o problema básico de *clustering* em que colunas $\mathbf{x}(n), n = 1, ..., N$ da matriz de dados X devem ser representadas, de forma aproximada, pelas colunas $\mathbf{y}(k), k = 1, ..., K$ do dicionário Y de forma que

seja minimizado o erro médio quadrático:

$$D = \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{x}(n) - \mathbf{y}(k(n))||^2$$

onde $k(n) = argmin_i ||\mathbf{x}(n) - \mathbf{y}(\mathbf{i})||$. Utilizando pseudo-código, escreva um algoritmo genético simples que, operando sobre uma população de dicionários $\mathbf{Y}$, leve à obtenção de uma solução $\mathbf{Y}^*$ localmente ótima para este problema. Defina todos os parâmetros que você julgar necessários.

Antes de se exibir o pseudo-código pensado para o problema, serão definidos alguns elementos básicos, que fazem parte do algoritmo genético simples.

1. Representação

De modo a facilitar a representação do problema, assume-se que os dados $\mathbf{x}(n)$ correspondem a vetores no $\mathbb{R}^2$, tendo, portanto, somente duas coordenadas x_1 e x_2, e que $x_1,\ x_2 \in \mathbb{Z}$. Os centróides $\mathbf{y}(i)$, por sua vez, também correspondem a vetores no $\mathbb{R}^2$, com coordenadas inteiras. O número K de centróides é um parâmetro livre, o qual pode ser definido pelo usuário (o parâmetro K poderia ser adaptado pelo próprio algoritmo, pensando em um algoritmo de parâmetros auto-adaptativo, por exemplo; entretanto, de modo a simplificar o algoritmo, determinou-se que tal parâmetro é estático, ou seja, definido antes da execução do algoritmo). Para esse problema, escolhe-se que $K = 3$.

Uma solução para o problema corresponde, então, a um conjunto de 3 centróides, que representariam os dados em questão. Dessa forma, o fenótipo das possíveis soluções são matrizes $\mathbf{Y}$, cujas colunas correspondem aos centróides escolhidos. Nesse caso, a dimensão dessa matriz seria 2×3 (já que arbritrou-se que os vetores estão no $\mathbb{R}^2$ e que $K = 3$). O genótipo $\mathbf{g}$ que codifica tais indivíduos é dado por uma string de bits, em que cada coordenada de um centróide é representada de forma binária. Sendo assim, supondo que sejam necessários B bits para representar uma coordenada de um centróide, seriam necessários $2B$ bits para definir totalmente um centróide. Como a matriz $\mathbf{Y}$ abriga 3 centróides, o genótipo final apresenta $6B$ bits. Cada segmento de $2B$ bits define, respectivamente, o centróide localizado na coluna 1, 2 e 3 dessa matriz.

Antes de se efetuar o cálculo da função custo (erro médio quadrático), encontra-se para cada vetor de dados $\mathbf{x}(n)$, o centróide, do indivíduo $\mathbf{Y}$ em questão, mais próximo dele, ou seja, aquele que minimiza a distância euclidiana $||\mathbf{x}(n)-\mathbf{y}(i)||_2$ (com $i = 1, ..., K$). Esse procedimento é repetido para cada indivíduo $\mathbf{Y}$ da população. Uma vez definidos os centróides de cada indíviduo adequados para cada dado, calcula-se a função custo de cada matriz $\mathbf{Y}$.

2. Seleção de pais

A seleção de pais é feita por meio do algoritmo "SUS" (*Stochastic Universal Sampling*), de modo a garantir que os indivíduos com melhores fitness tenham mais chances de se reproduzir e gerar filhos melhores. O tamanho da população reprodutora é igual ao tamanho da população original. Além disso, cada par de pais é selecionado aleatoriamente da população reprodutora e ele gerará dois filhos.

3. Recombinação

O operador de recombinação a ser aplicado a cada par de pais é conhecido como *"N-point crossover"*. Nele, são sorteados, aleatoriamente, N posições no genótipo dos pais, dividindo-o em N+1 segmentos, os quais, por sua vez, são copiados alternadamente para os filhos (por exemplo, no caso de N = 1, o filho 1 recebe o segmento 1 do pai 1 e o filho 2 recebe o segmento 2 do pai 1; analogamente, o pai 2 cede os segmentos 1 e 2 aos filhos 2 e 1, respectivamente). Como são 3 centróides por indivíduo, define-se, para esse problema, N = 2. Desse modo, o filho 1 recebe os segmentos 1 e 3 do pai 1 e o segmento 2 do pai 2, ao passo que o filho 2 recebe os segmentos 1 e 3 do pai 2 e o segmento 2 do pai 1. É definida, também, uma probabilidade com a qual a recombinação deve ocorrer (ou seja, dos pais selecionados, de fato, reproduzirem). Caso não ocorra recombinação, os filhos gerados serão, somente, cópias do par de pais selecionados. Arbitra-se que tal probabilidade seja de 90%.

4. Mutação

Admite-se, também, uma probabilidade de cada bit de cada um dos filhos gerados sofrer mutação. Caso ocorra mutação, o bit em questão troca de valor ($0 \rightarrow 1$ ou $1 \rightarrow 0$). Essa taxa é fixada em 5%.

5. Seleção de sobreviventes

Uma vez finalizada a população de filhos, realiza-se a seleção de sobreviventes. Supondo que a população original tenha tamanho μ, como cada par de pais gerou um par de filhos, existem 2μ indivíduos, dos quais somente μ passarão para a geração seguinte. Para isso, calcula-se a função de fitness para cada indivíduo e a seleção feita por ranqueamento, ou seja, somente os indivíduos mais aptos, levando-se em conta pais e filhos, sobreviverão (alternativamente, os μ indivíduos menos aptos são substituídos). Essa abordagem também é conhecida como *GENITOR*.

6. Inicialização

A população é inicializada de forma aleatória e o tamanho da população μ é definido como $\mu = 100$ indivíduos.

7. Condição de parada

A condição de parada do algoritmo é dada por duas condições: até que se alcance um número pré-definido de gerações (aqui, definido como 1000 gerações), ou até que o melhor fitness encontrado seja menor do que 10^{-5} (o valor mínimo da função fitness, que se busca minimizar, é de 0, porém, dados os erros de aproximação computacionais, admite-se uma tolerância para esse mínimo).

O pseudo-código é exibido a seguir:

$tam\_pop \leftarrow 100$
$x \leftarrow$ matriz contendo os vetores de dados
$P \leftarrow$ inicialização aleatória da população, sendo P uma matriz contendo os genótipos de cada indivíduo

$melhor\_fitness \leftarrow 10000$
$n \leftarrow 1$
$p_{rec} \leftarrow 0.9$
$p_{mut} \leftarrow 0.05$

Enquanto $(n < 1000)$ e $(melhor\_fitness > 10^{-5})$ faça

Para cada indivíduo i da população P, encontra-se o centróide mais próximo de cada vetor de dado x

```
F ← vetor contendo o valor da função de fitness (
    erro médio quadrático) de cada indivíduo da
    população

melhor_fitness ← menor valor armazenado em F

Calcula a probabilidade de cada indivíduo ser
    selecionado como pai, dividindo a sua fitness
    pela soma de todas as fitness da população

a ← vetor contendo a distribuição cumulativa,
    baseada nas probabilidades calculadas
    anteriormente, onde cada posição desse vetor
    corresponde ao i-ésimo indivíduo de P

membro_atual ← 1
i ← 1
r ← sorteia uma amostra da distribuição uniforme
    entre [0, 1/μ]

Enquanto (membro_atual ≤ μ) faça
        Enquanto (r ≤ a[i]) faça

                reprodutores[membro_atual] ← vetor
                    contendo cópias dos pais da
                    população P selecionados para
                     reprodução

                r ← r + 1/μ
                membro_atual ← membro_atual + 1
        Fim Enquanto
        i ← i + 1
Fim Enquanto

i ← 1
Enquanto (i < tam_pop) faça
        Seleciona aleatoriamente dois pais da
            população de reprodutores
        r ← sorteia um número da distribuição
            uniforme entre [0, 1]
        Se (r < p_rec) faça
                Seleciona aleatoriamente 2 pontos
                     no genótipo dos pais,
                    segmentando-o em 3 partes
```

```
                        Filho i ← segmentos 1 e 3 do pai
                            1 e segmento 2 do pai 2
                        Filho i+1 ← segmentos 1 e 3 do
                            pai 2 e segmento 2 do pai 1
                Senão
                        Filho i ← pai 1
                        Filho i+1 ← pai 2
                Fim Se

                i ← i+2
        Fim Enquanto

        Para cada filho gerado faça
                Para cada bit do filho em questão faça
                        r ← sorteia um número da
                            distribuição uniforme entre
                            [0, 1]
                        Se (r < p_mut) faça
                                Troca o valor do bit em
                                    questão
                        Fim Se
                Fim Para
        Fim Para

        Calcula a função de fitness da população total
            filhos + pais

        P ← os tam_pop indivíduos mais aptos para a pró
            xima geração (ou seja, aqueles que apresentam
            os menores erros médios quadráticos)

        n ← n+1
Fim Enquanto
```