UNIVERSIDADE
FEDERAL DO CEARÁ

CURSO DE ENGENHARIA DE COMPUTAÇÃO

INTELIGÊNCIA COMPUTACIONAL APLICADA

PROFESSOR MÁRCIO AMORA

ALGORITMO GENÉTICO

ARTHUR SOUSA DE SENA MATRÍCULA: 345750

VICTÓRIA TOMÉ OLIVEIRA MATRÍCULA: 366333

VITORIA ALMEIDA BERTAIA MATRICULA: 356741

SOBRAL – CE

20/06/2016

SUMÁRIO

1. INTRODUÇÃO

Nos últimos anos, pesquisas sobre modelos computacionais veem buscando inspirações na natureza, pois nela existem alguns processos que podem ser considerados "perfeitamente inteligentes". Análises complexas com soluções muito interessantes foram identificadas em métodos naturais, especialmente nos seres vivos.

Cientistas , matemáticos e engenheiros estão sempre em busca de ideias e métodos para que possam ser aplicadas em problemas complexos, retornando resultados satisfatórios e eficientes.

A Computação Evolucionária (CE) tem como base métodos evolutivos da natureza. Relacionada diretamente ao Darwinismo, vê tal teoria como um processo de adaptação e seleção natural, onde somente os melhores e mais bem aptos poderão sobreviver e se reproduzir. Em outras palavras, a CE visa analisar a adequação das estruturas computacionais, mantendo as que melhor se adaptam ao meio evoluído. Existem vários métodos e paradigmas para uma CE, o mais importante atualmente são os Algoritmos Genéticos (GAs, do inglês Genetic Algorithms).

GAs é uma das técnicas mais estudadas de Computação Evolucionária, tanto pela sua flexibilidade como por sua eficiência em realizar buscas global em diversos tipos de ambientes.

Este trabalho tem como objetivo introduzir conceitos de GAs, bem como alguns pontos que ajudam a aumentar sua eficiência. Um exemplo de aplicação, bem como detalhes de sua implementação.

2. ALGORITMO GENÉTICO

Os Algoritmos Genéticos, são métodos de otimização e busca baseados nos mecanismos de evolução de populações de seres vivos. Foram introduzidos por John Holland em 1975 e introduzido por um dos seus alunos, David Goldberg em 1989. Estes algoritmos se baseiam no princípio da seleção natural, pronunciado por Charles Darwin em 1859 no livro A Origem das Espécies. Desde então os Algoritmos Genéticos são utilizados para solucionar problemas de otimização e aprendizado de maquinas.

O Algoritmo Genético processa populações de cromossomos, um cromossomo é uma estrutura de dados, que representa uma possível resolução do problema a ser otimizado. Em suma, um cromossomo retrata um conjunto de parâmetros da função objetivo cuja resposta será maximizada ou minimizada. O conjunto de todas as configurações que o cromossomo pode assumir se constitui no seu espaço de busca. Se o cromossomo representa n parâmetros de uma função, então o espaço de busca é um espaço que contêm n dimensões.

O primeiro passo de um Algoritmo Genético típico é a criação de uma população inicial de cromossomos, que é constituída por um conjunto aleatório de cromossomos que simbolizam possíveis soluções do problema a ser resolvido. Durante o processo evolutivo, esta população é avaliada e cada cromossomo recebe uma nota de aptidão, representando a qualidade da solução que ele representa. Seguindo a teoria do Darwinismo, os cromossomos mais aptos são selecionados e os menos aptos são descartados. Os cromossomos mais aptos selecionados podem sofrer alterações em suas características fundamentais através dos operadores de crossover e mutação, assim gerando descendentes para a próxima geração. Este processo é repetido até que uma solução satisfatória seja encontrada.

Existem três principais critérios de parada de um Algoritmo Genético, são eles:

- Atingir um número de gerações;
- Chegada de um valor ótimo da função objetivo;
- Convergência.

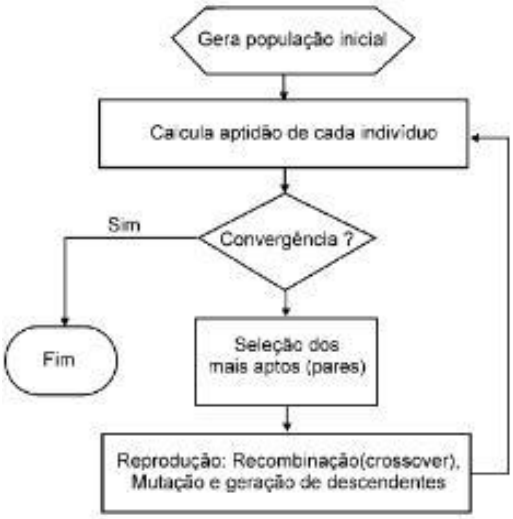

Figura 1. Algoritmo Gênetico Básico

2.1 Otimização

A otimização é a busca da solução mais adequada para um certo problema. Consiste em tentar várias soluções possíveis e utilizar a informação conquistada neste processo de forma a encontrar soluções cada vez melhores e mais adequadas.

As técnicas de busca e otimização, em geral, retratam:

- Um espaço de busca, onde se encontram todas as possíveis soluções do problema;
- Uma função objetivo, ou seja, uma função de aptidão, que é utilizada para avaliar as soluções obtidas, associando a cada uma delas uma nota.

Em termos matemáticos, a otimização consiste em encontrar a solução que corresponda ao ponto de máximo ou mínimo da função objetivo.

2.2 Seleção

O processo de seleção é baseado no processo de seleção natural dos seres vivos, o Algoritmo Genético seleciona os melhores cromossomos da população inicial, ou seja, aqueles com maior aptidão para gerar cromossomos filhos através dos operadores crossover e mutação.

Existem várias formas de seleção, entre elas há o método de seleção por Roleta e o método de seleção por Torneio.

2.2.1 Seleção por Roleta

No método de seleção por Roleta, cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Consequentemente, para indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos indivíduos de aptidão mais baixa, é dada uma porção menor. Um dos maiores problemas existentes se dá pelo tempo de processamento, já que o método obriga duas passagens por todos os indivíduos da população.

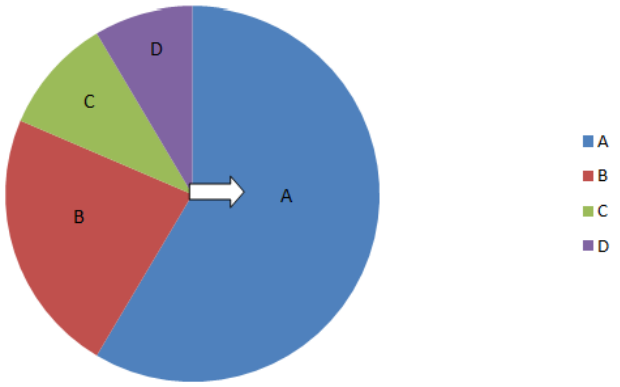

Figura 2. Método da Roleta

Em 1997, Mitchell descreveu um algoritmo baseado no método da roleta, mostrado abaixo:

```
Inicio
    T = soma dos valores de aptidão de todos os indivíduos da população
    Repita N vezes para selecionar n indivíduos
r = valor aleatório entre 0 e T
Percorra seqüencialmente os indivíduos da população, acumulando em S o valor de aptidão dos indivíduos já percorridos
Se S >= r então
        Selecione o indivíduo corrente
Fim se
    Fim Repita
Fim
```

2.2.2 Seleção por Torneio

No método de seleção por Torneio, um número n de indivíduos da população é selecionado aleatoriamente para constituir uma sub-população temporária. Nesta sub-população, o indivíduo selecionado dependerá de uma probabilidade k definida previamente. O método de Torneio é mais utilizado, pois oferece a vantagem de não exigir que a comparação seja feita entre todos os indivíduos da população.

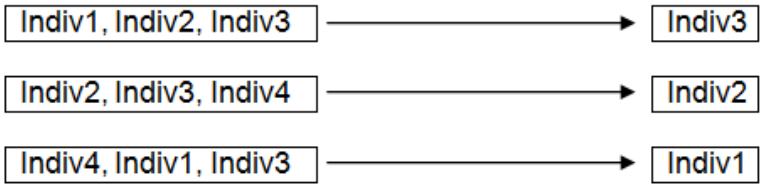

Figura 3. Método de Torneio

Em 1997, Mitchell descreveu um algoritmo baseado no método de Torneio mostrado abaixo, com n=2:

```
Inicio
k = 0.75
    Repita N vezes
    Escolha 2 indivíduos da população aleatoriamente
    r = valor aleatório entre 0 e 1
    Se r < k
O melhor indivíduo é escolhido
    Senão
O pior indivíduo é escolhido
    Fim se
    Fim Repita
Fim
```

2.3 Elitismo

O Elistimo é quando ocorre a tranferencia do melhor cromossomo de uma geração para a outra sem alterações, pois se isso não ocorrer o melhor cromossomo pode ser perdido de uma geração para outra devido ao corte do crossover ou a ocorrência de mutação.

2.4 Operadores Genéticos

Os operadores genéticos têm como principio básico modificar a população através de sucessivas gerações, expandindo a busca até chegar ao melhor resultado, ou seja, o resultado mais satisfatório. Os operadores genéticos são necessários para que a população se diversifique e mantenha características de adaptação adquiridas pelas gerações anteriores. Os operadores de cruzamento e de mutação têm um papel fundamental em um algoritmo genético.

2.3.1 Crossover (Cruzamento)

O crossover é considerado o operador genético predominante. Através do cruzamento são criados indivíduos novos, recebendo características misturadas de dois indivíduos "pais". Esta mistura é feita tentando imitar a reprodução de genes em células. Partes das características de um indivíduo "pai" são trocadas por parte equivalentes da do outro indivíduo "pai". O resultado dessa operação é um indivíduo "filho" que potencialmente combine as melhores características dos indivíduos usados como base, ou seja, os indivíduos "pais".

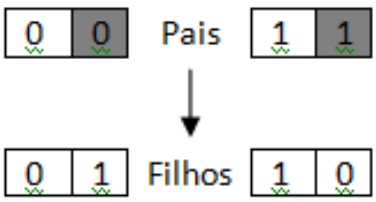

Figura 4. Cruzamento

2.3.2 Mutação

O método de Mutação, simplesmente modifica aleatoriamente alguma característica do indivíduo sobre o qual é aplicada. A mutação é necessária para a introdução e manutenção da diversidade genética da população. Mutações neurais são definidas como mutações cujos efeitos não influenciam a aptidão dos indivíduos.

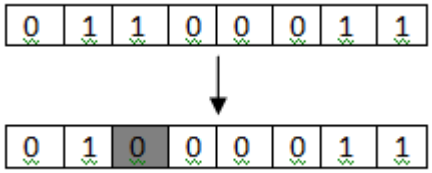

Figura 4. Mutação

2.4 Parâmetros Genéticos

Alguns parâmetros influenciam no comportamento dos Algoritmos Genéticos e podem ser escolhidos para melhorar o seu desempenho, adaptando-se às características particulares de determinadas classes de problemas.

- Tamanho da População – O tamanho da população afeta o desempenho e a eficiência do Algoritmo Genético. Quando a população é pequena o desempenho diminui, pois a população fornece uma pequena cobertura do espaço de busca do problema. Portanto, uma grande população geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior.
- Taxa de Cruzamento - A taxa de cruzamento e proporcional a criação de novas estruturas sendo introduzidas na população. Mas se esta for muito alta, estruturas com boas aptidões poderão ser retiradas mais rapidaum valor alto, a maior parte da população será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.
- Taxa de Mutação - Uma baixa taxa de mutação previne que uma dada posição fique estagnada em um valor, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Com uma taxa muito alta a busca se torna essencialmente aleatória.

2.5 Representação Binária x Real

A representação binária é historicamente importante, dado que foi utilizada Holland em 1975. É a representação mais tradicional, sendo fácil de utilizar e manipular, como também é simples de analisar teoricamente. Porém, se um problema tem parâmetros contínuos e o usuário quer trabalhar com uma maior precisão numérica, ele precisará armazenar cromossomos longos na memória e isso acarretará em uma menor otimização. Porém, a representação real gera cromossomos menores e é compreendida mais naturalmente pelo ser humano do que a cadeira de binário, e tem desempenho melhor que a binária para parâmetros contínuos.

3. DESCRIÇÃO DO PROBLEMA

O objetivo do trabalho é utilizar algoritmo genético com representação real pra encontrar o ponto de máximo da função $z = \left(1 + \sqrt{x} + 0.5 * y\right)^2$ mostrada na Figura 5.

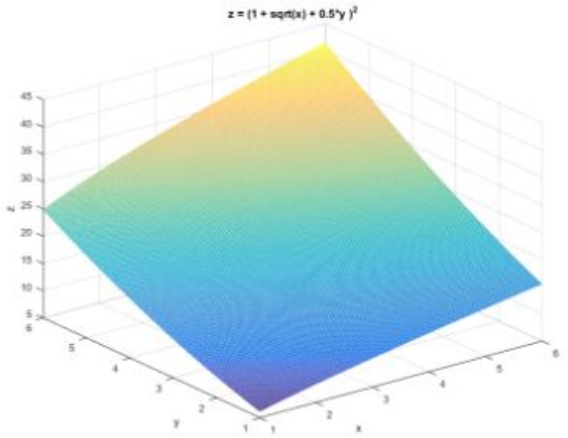

Figura 5. Mutação

Para propósito deve ser realizado utilizando dois métodos de seleção, Roleta e Torneio, dois métodos de cruzamentos, Simples e Flat, e mutação aleatória. Tais métodos devem ser comparados para verificar a eficácia de cada um.

4. IMPLEMENTAÇÃO EM MATLAB

O algoritmo implementado trabalha em três modos distintos. No primeiro modo, o algorítimo realiza o objetivo principal do trabalho, que é encontrar o ponto de máximo da função dada. Nesse modo, são exibidos basicamente dois gráficos, o gráfico do ponto de máximo encontrado e o gráfico de evolução da aptidão do melhor cromossomo durante as gerações. Como já dito anteriormente, foi implementado dois métodos de seleção e dois métodos de cruzamento, dessa forma, existem quatro possíveis combinações de seleção e cruzamento: (roleta, simples), (roleta, flat), (torneio, simples) e (torneio, flat). A execução do algoritmo é realizada para cada um dos quatro casos. No primeiro modo, o tamanho da população é fixado em 200 cromossomos, o número de gerações é 200, a taxa de cruzamento de 80% e probabilidade de mutação de 30%.

No segundo modo de operação, o algoritmo exibe uma comparação entre os métodos implementados de seleção e cruzamento. Os parâmetros do algoritmo genético são os mesmo do modo anterior, contudo, a execução do algoritmo é repetida 50 vezes para cada uma das quatro combinações. Ao final das 50 repetições, são exibidos dois gráficos, o primeiro exibe o tempo gasto para cada repetição de cada método, e o segundo, a comparação de erros.

No terceiro modo, o algoritmo também é executado 50 vezes com os mesmos parâmetros anteriores, exceto o tamanho da população, que é variado linearmente com o número de execuções. Esse modo foi implementado para verificar a influência do tamanho da população na eficiência do algoritmo. O tamanho inicial da população é setado para 10, então, a cada repetição esse número é incrementado em 10 até que se atinja as 50 repetições, resultando assim em uma população final de tamanho 500. Ao final das execuções, são exibidos os gráficos de erro e de tempo para cada combinação de seleção e cruzamento.

Abaixo temos código implementado em MATLAB:

```
function [parent1,parent2]=crossFlat(parent1,parent2,crossProb)
    if(crossProb>rand)
        chromoSize = length(parent1); % Tamanho do cromossomo
        for i = 1: chromoSize
            interval = [parent1(i),parent2(i)];
            raffle1 = (min(interval) + (max(interval)-min(interval))*rand); % Sorteio do elemento i
do parent1
            raffle2 = (min(interval) + (max(interval)-min(interval))*rand); % Sorteio do elemento i
do parent2

            % o crossover é realizado para cada elemento i;
            parent1(i) = raffle1;
            parent2(i) = raffle2;
        end
    end
end
```

```
function [parent1,parent2]=crossSimples(parent1,parent2,crossProb)
    if(crossProb>rand)
        chromoSize = length(parent1); % Tamanho do cromossomo
        cut = round((1 + (chromoSize-1)*rand)); % Ponto de corte é definido aleatoriamente

        % O crossover é realizado
        temp = parent1(cut); % Elemento do parent1 é armazenado temporariamente
        parent1(cut) = parent2(cut); % Parent1 recebe elemento de parent2
        parent2(cut) = temp; % Parent2 recebe elemento de parent1
    end
end

function child=mutRand(child,interval,mutProb)
    if(mutProb>rand)
        chromoSize = length(child); % Tamanho do cromossomo
        mutPosition = round((1 + (chromoSize-1)*rand)); % A posição da mutação é definida
aleatoriamente

        mutation = interval(1) +(interval(2)-interval(1))*rand;

        % A mutação é realizada
        child(mutPosition) = mutation;
    end
end
% Loop de gerações, e executado até que o número máximo de gerações seja atingido
        for gen=1:numGenerations

            F = funcApt(pop(:,1),pop(:,2)); % Aptidão é calculada para todos os valores da atual
polulação

            [~, maxFi] = max(F);        % Melhor cromossomo é guardado
            currentMax = pop(maxFi,:);  % para garantir o elitismo

            % Seleção Roleta ------------------------------------------------------
            if (strcmp(tipoSelecao,'Roleta'))
                prob = zeros(length(F),3); % Vetor de probabilidades
                for i=1:length(pop');
                    % Cada probabilidade é calculada
                    prob(i,:) = [funcApt(pop(i,1),pop(i,2)) ./ sum(F),pop(i,1),pop(i,2)];
                end

                % O vetor de probabilidades é ordenado para ser utilizado na roleta
                prob = sortrows(prob,1);

                % Vetor utilizado para guardar as posições da roleta e
                % utilizado pra evitar que tal posição seja selecionada novamente
                positions = zeros(sizePop/2,1);

                i=1;
                while i<=sizePop/2
                    % Uma posição é gerada aleatoriamente, simulando a roleta
                    roulette =round(1 + (sizePop-1)*rand);
                    if ~any(positions==roulette) % Verifica se a posição já foi retirada
                        positions(i)=roulette; % Salva a posição gerada
                        % O cromossomo é guardado na nova população
                        newPop(i,:)=[prob(positions(i),2),prob(positions(i),3)];
                        i = i+1;
                    end
                end
                newPop((sizePop/2+1:end),:) = flipud(newPop((1:sizePop/2),:));

            end
            %--------------------------------------------------------------------------


            % Seleção Torneio -----------------------------------------------------
            if (strcmp(tipoSelecao,'Torneio'))
                numTour = 5; % Número de cromossomos sorteados
                positions = zeros(numTour,1); % Vetor utilizado para evitar repetição de torneio

                for i=1:sizePop/2
                    t = 1;
                    while t<numTour+1
                        tournament = round(1 + (sizePop - 1)*rand); % Torneio é realizado
```

```
                        if ~any(positions==tournament) % Verifica se o cromossomo já foi sorteado
                            positions(t)=tournament;
                            tempPop(t,:)=[pop(positions(t),1),pop(positions(t),2)]; % Cromossomos
sorteados é armazenado em vetor temporário
                            t = t+1;
                        end
                    end

                    F = funcApt(tempPop(:,1),tempPop(:,2));
                    [~,maxTourI] = max(F); % O melhor cromossomo é selecionado dentre daqueles
sorteados
                    newPop(i,:) = [tempPop(maxTourI,1),tempPop(maxTourI,2)]; % Cromossomo é salvo
na nova população
                end
                newPop((sizePop/2+1:end),:) = flipud(newPop((1:sizePop/2),:));
            end


            %---------------------------------------------------------------------


            % Crossover simples ---------------------------------------------------
            if(strcmp(tipoCrossover,'Simples'))
                for i=1:2:sizePop
                    [newPop(i,:), newPop(i+1,:)]=crossSimples(newPop(i,:),
newPop(i+1,:),crossProb);
                end
            end
            %---------------------------------------------------------------------

            % Crossover flat ------------------------------------------------------
            if(strcmp(tipoCrossover,'Flat'))
                for i=1:2:sizePop
                    [newPop(i,:), newPop(i+1,:)]=crossFlat(newPop(i,:), newPop(i+1,:),crossProb);
                end
            end
            %---------------------------------------------------------------------


            % Mutação aleatória ---------------------------------------------------
            for i=1:sizePop
                newPop(i,:)=mutRand(newPop(i,:),[limInf limSup],mutProb);
            end
            %---------------------------------------------------------------------


            % Elitismo ------------------------------------------------------------
            pop = newPop;
            pop(maxFi,:) = currentMax; % O melhor cromossomo é passado para a geração seguinte.
            % ---------------------------------------------------------------------


            % Os gráficos são plotados ----------------------------------

            F = funcApt(pop(:,1),pop(:,2));
            [zMax(gen), iMax] = max(F);
            xMax(gen) = pop(iMax,1);
            yMax(gen) = pop(iMax,2);
            if OPMode==1
                if grafTempoReal
                    passo = 0.03;
                    x = limInf:passo:limSup;
                    y = limInf:passo:limSup;
                    [x,y]=meshgrid(x,y);
                    z = funcApt(x,y);
                    mesh(x,y,z);
                    hold on;
                    plot3(pop(:,1),pop(:,2),F,'b+');
                    plot3(xMax(gen),yMax(gen),zMax(gen),'r*');
                    xlabel(sprintf('X_{max} = %f',xMax(gen)));
                    ylabel(sprintf('Y_{max} = %f',yMax(gen)));
                    zlabel(sprintf('Z_{max} = %f',zMax(gen)));
                    hold off;
```

```
                    title(sprintf('Método de seleção: %s, Método de cruzamento: %s, Geração = %d',
tipoSelecao, tipoCrossover, gen));
                    pause(0);
                end
            end
        end
```

5. RESULTADOS

Através dos resultados obtidos, pôde-se perceber interessantes características de cada método implementado. Verificou-se que o método de seleção Torneio gera resultados bem mais precisos que o método Roleta. Quanto aos métodos de cruzamento, quando se utiliza seleção Roleta, o cruzamento mais eficaz é o método Simples, já o Flat gera erros bastante altos. Já com o método de seleção Torneio, o cenário é diferente, nesse caso ambos os métodos de cruzamento geram erros muito baixos, mas o Flat na maioria das vezes se sobressai em relação ao simples, embora que por uma diferença pequena.

Abaixo temos os resultados para o primeiro modo de operação. No gráfico da Figura 6, está exposto em vermelho o ponto de máximo encontrado, e em azul o restante da população da geração. Esse primeiro gráfico é referente ao método de seleção Roleta e método de cruzamento Simples. Através dos gráficos da Figura 7, nota-se que na geração 20, o algoritmo já apresentava uma boa aptidão, e essa aptidão se manteve sendo a melhor durante aproximadamente 120 gerações, só então, surgiu um cromossomo com aptidão melhor.

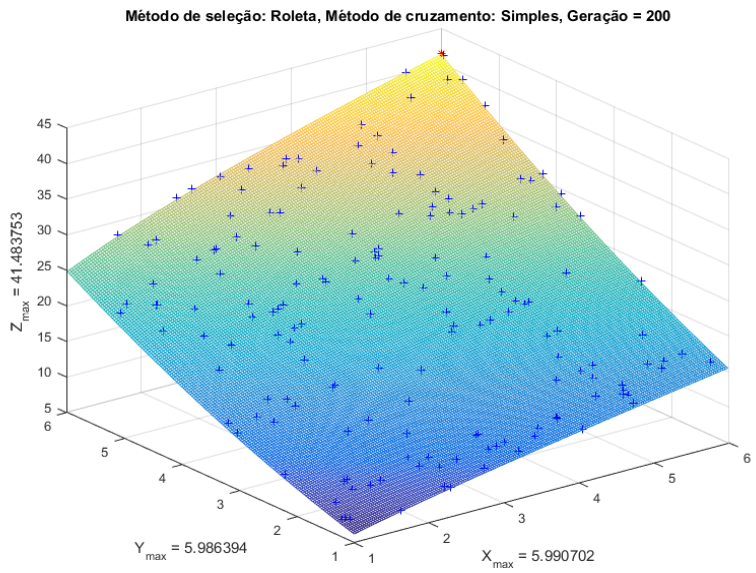

Figura 6. Gráfico de aptidões da população da geração de número 200, com seleção Roleta e cruzamento Simples.

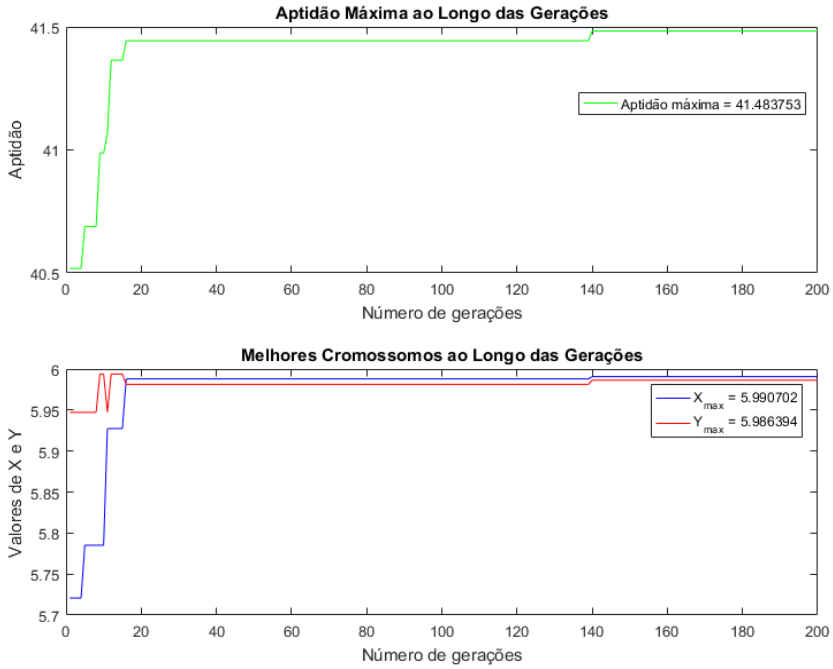

Figura 7. Gráficos de aptidões máximas ao longo das gerações, com seleção Roleta e cruzamento Simples.

No gráfico da Figura 8, temos os resultados para o método de seleção Roleta e método de cruzamento Flat. Essa combinação foi a que se mostrou menos eficiente nos testes realizados. Quando a comparação entre os métodos for realizada mais a frente, isso ficará mais evidente. Analisando o gráfico da Figura 8, vemos que a população geral é bem mais espalhada e distante do ponto de máximo. Nos gráficos da Figura 9, vemos que a partir da geração 50, não há mais melhoria na aptidão máxima da população, passando cerca de 130 gerações sem nenhuma evolução.

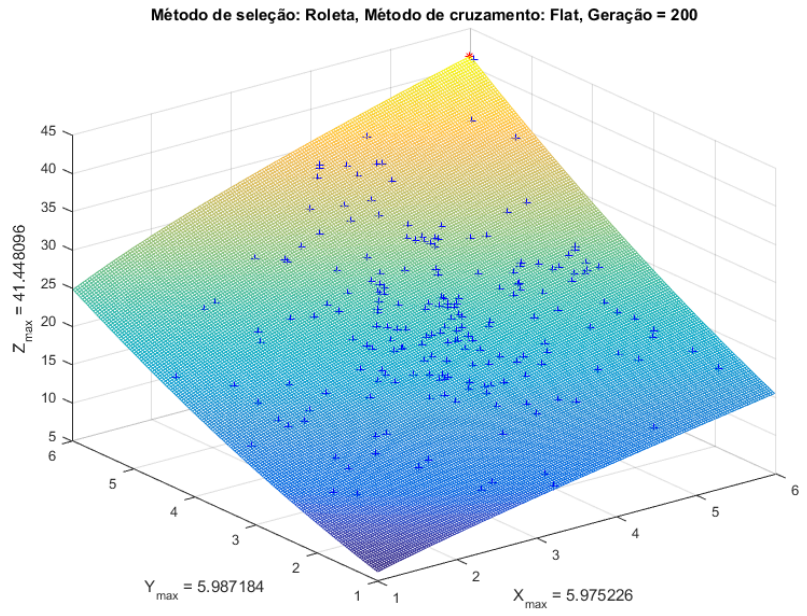

Figura 8. Gráfico de aptidões da população da geração de número 200, com seleção Roleta e cruzamento Flat.

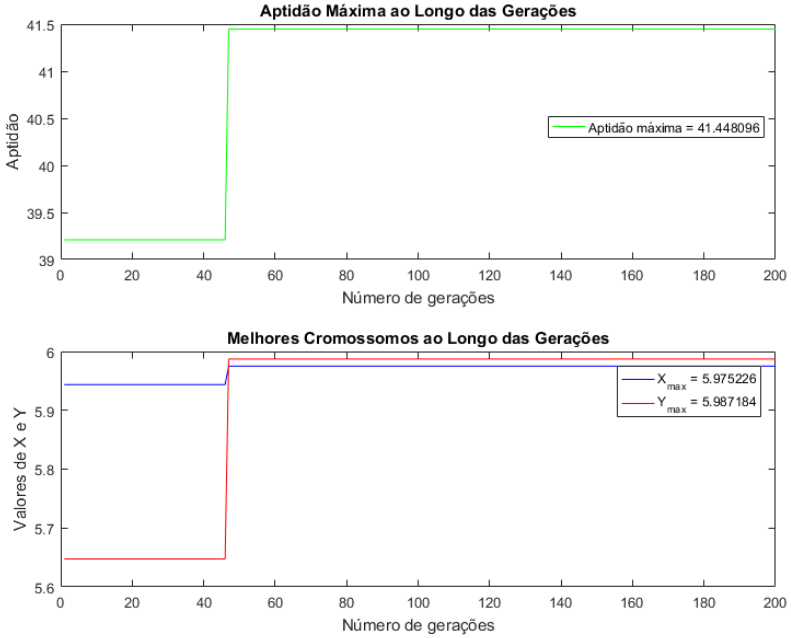

Figura 9. Gráficos de aptidões máximas ao longo das gerações, com seleção Roleta e cruzamento Flat.

Os resultados obtidos através do método de seleção Torneio foram muito precisos em ambos cruzamentos, Simples e Flat. Todos os erros foram bem mais baixos que a Roleta. No gráfico da Figura 10, temos o resultado para a configuração de seleção Torneio e cruzamento Simples. Como pode ser notado na Figura 11, a melhora na aptidão máxima da população acontece em intervalos de geração bem curtas, se comparado com o método de seleção Roleta, proporcionando assim erros muito baixos.

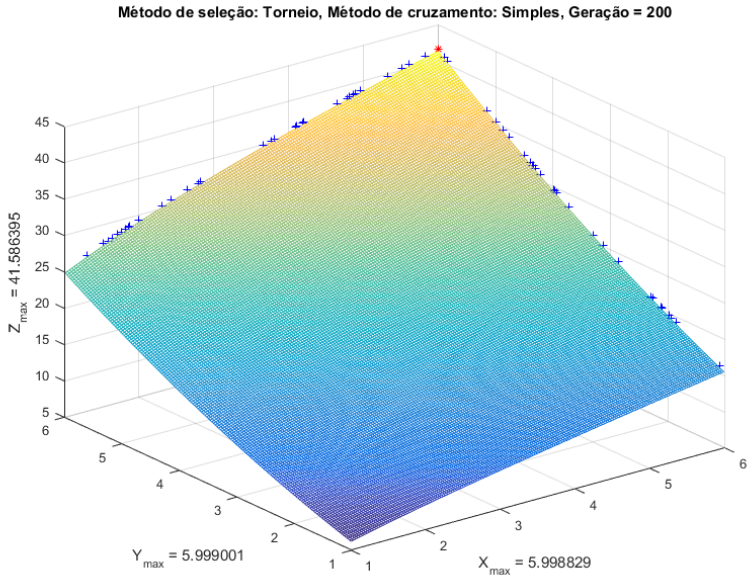

Figura 10. Gráfico de aptidões da população da geração de número 200, com seleção Torneio e cruzamento Simples.

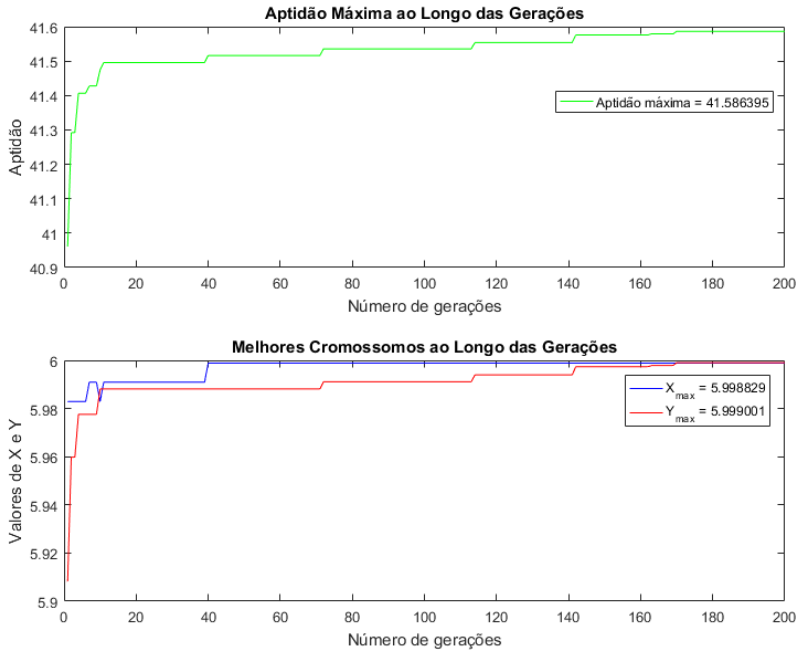

Figura 11. Gráficos de aptidões máximas ao longo das gerações, com seleção Torneio e cruzamento Simples.

A combinação de seleção Torneio e cruzamento Flat se mostrou tão eficiente quanto o caso anterior, proporcionando erros ainda mais baixos, o que não aconteceu quando aplicamos no método de seleção Roleta. No gráfico da Figura 12 temos a situação final da população para essa configuração. Vemos na Figura 13 que a aptidão máxima já fica muito próxima do valor final na geração 60, e nas gerações seguintes é aumentada a precisão do ponto de máximo encontrado.

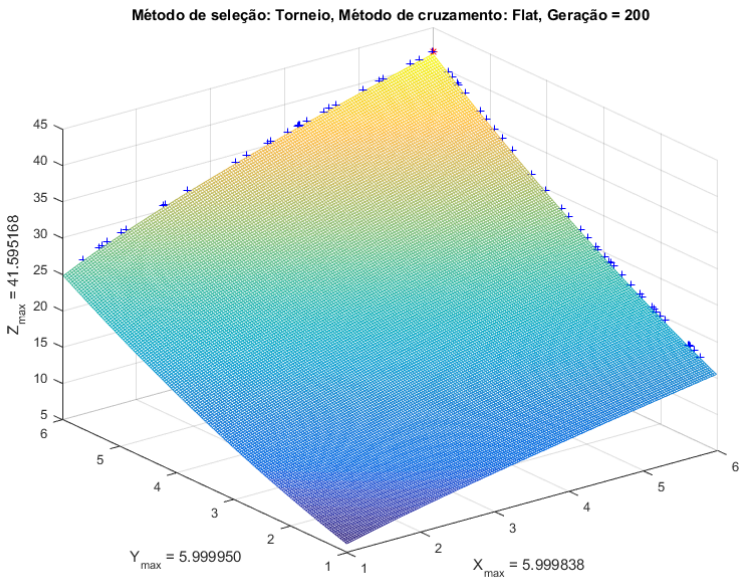

Figura 12. Gráfico de aptidões da população da geração de número 200, com seleção Torneio e cruzamento Flat.

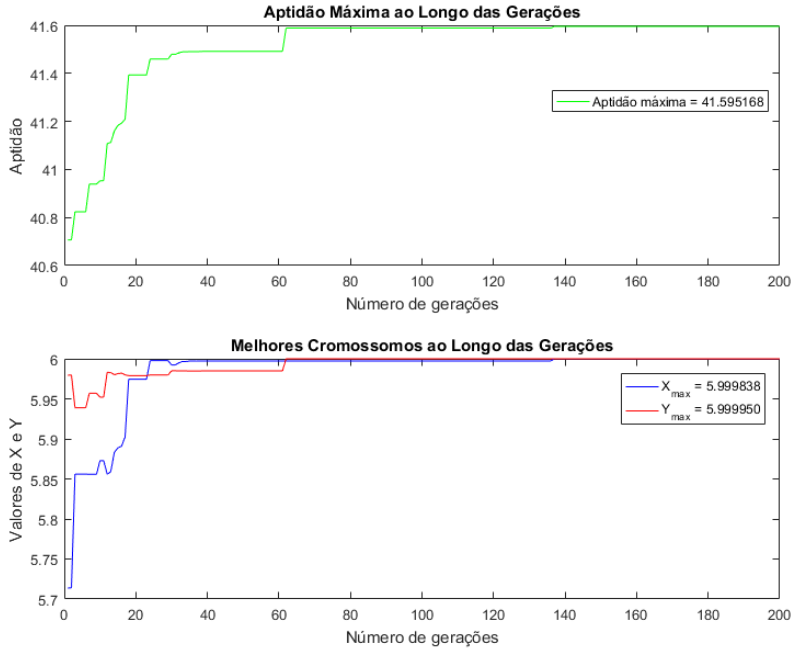

Figura 13. Gráficos de aptidões máximas ao longo das gerações, com seleção Torneio e cruzamento Flat.

Abaixo, temos os relatórios de resultados mostrados no MATLAB. É possível verificar quais métodos geram erros mais baixos. Aqui, fica evidente que o método de seleção Roleta em conjunto com o cruzamento Flat geram um erro bem mais alto do que as demais configurações. Além disso, percebe-se que o método de seleção Torneio é superior ao método Roleta, no quesito precisão. Contudo, no quesito tempo, o método Roleta de sobressai, resultando tempos mais baixos que o torneio. O cruzamento Flat e seleção Torneio, proporcionou erro extremamente baixo, de apenas 0,0008, porém o tempo gasto foi o mais alto, atingindo 0,45 s.

```
============================================================
       Algorítmo Genético com Representação Real
          para obtenção do ponto de máximo da
          função Z = (1 + sqrt(x) + 0.5*y )^2

==================== Parâmetros do GA ====================
Número de cromossomos: 200
Número de gerações: 200
Método de seleção utilizado: Roleta
Método de cruzamento utilizado: Simples


------------------- Resultados --------------------

Tempo gasto: 0.21 s

Valores obtidos
X = 5.965327     Y = 5.992189     Z = 41.454234

Ponto de máximo real
X = 6            Y = 6            Z = 41.595918

Erros
e_x = 0.578%     e_y = 0.130%     e_z = 0.341%

Erro absoluto: 0.1461

============================================================
```

```
============================================================
       Algorítmo Genético com Representação Real
          para obtenção do ponto de máximo da
          função Z = (1 + sqrt(x) + 0.5*y )^2

==================== Parâmetros do GA ====================
Número de cromossomos: 200
Número de gerações: 200
Método de seleção utilizado: Roleta
Método de cruzamento utilizado: Flat


------------------- Resultados --------------------

Tempo gasto: 0.29 s

Valores obtidos
X = 5.854520     Y = 5.991565     Z = 41.157278

Ponto de máximo real
X = 6            Y = 6            Z = 41.595918

Erros
e_x = 2.425%     e_y = 0.141%     e_z = 1.055%

Erro absoluto: 0.4622

============================================================
```

```
=========================================================
       Algorítmo Genético com Representação Real
          para obtenção do ponto de máximo da
          função Z = (1 + sqrt(x) + 0.5*y )^2

=================== Parâmetros do GA ====================
Número de cromossomos: 200
Número de gerações: 200
Método de seleção utilizado: Torneio
Método de cruzamento utilizado: Simples


------------------- Resultados --------------------

Tempo gasto: 0.32 s

Valores obtidos
X = 5.998829     Y = 5.999001     Z = 41.586395

Ponto de máximo real
X = 6            Y = 6            Z = 41.595918

Erros
e_x = 0.020%     e_y = 0.017%     e_z = 0.023%

Erro absoluto: 0.0096

=========================================================
```

```
=========================================================
       Algorítmo Genético com Representação Real
          para obtenção do ponto de máximo da
          função Z = (1 + sqrt(x) + 0.5*y )^2

=================== Parâmetros do GA ====================
Número de cromossomos: 200
Número de gerações: 200
Método de seleção utilizado: Torneio
Método de cruzamento utilizado: Flat


------------------- Resultados --------------------

Tempo gasto: 0.45 s

Valores obtidos
X = 5.999838     Y = 5.999950     Z = 41.595168

Ponto de máximo real
X = 6            Y = 6            Z = 41.595918

Erros
e_x = 0.003%     e_y = 0.001%     e_z = 0.002%

Erro absoluto: 0.0008

=========================================================
```

Agora são apresentados os resultados obtidos no segundo modo de operação do algoritmo. O número de repetições da execução do algoritmo foi definido para 50, tamanho da população de 200 e número de gerações 200. Nesse modo, é possível verificar a média de tempo e o comportamento de erro para cada combinação de seleção e cruzamento. Vemos através do gráfico da Figura 14 que a configuração que proporciona resultados mais rápidos é a de seleção Roleta e cruzamento Simples, com média de tempo de 0,18s. No outro extremo, temos os métodos de seleção Torneio e cruzamento Flat como os mais lentos.

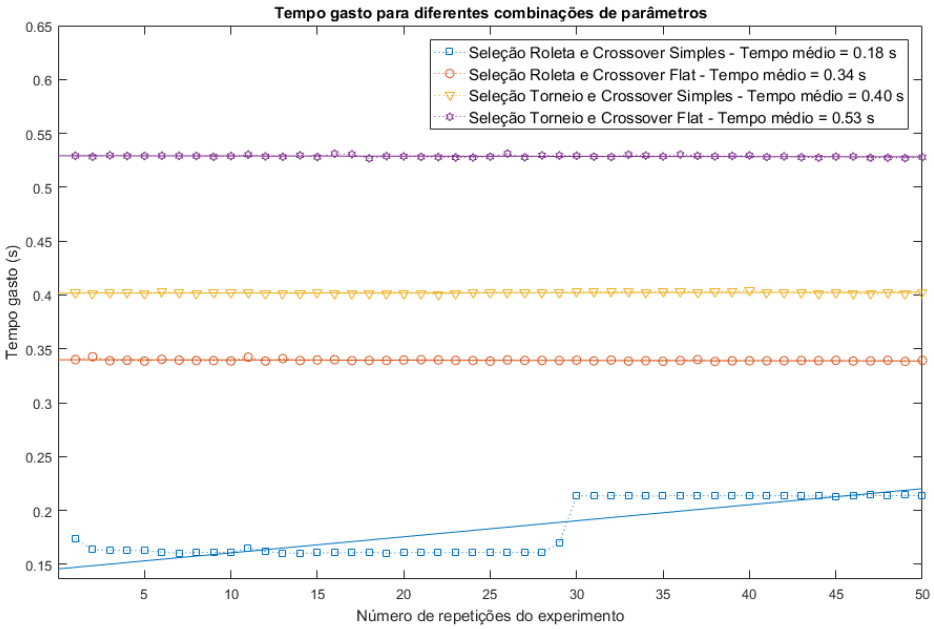

Figura 14. Gráfico de tempos gastos para cada combinação dos métodos de cruzamento e seleção implementados.

No gráfico de erros, exibido na Figura 15, podemos perceber o quão eficiente é cada método. Percebe-se aqui com clareza que o método de seleção Roleta e cruzamento Flat, possui a menor eficiência, gerando erros muito altos, se comparados as outras configurações. Embora a média de erro da Roleta Simples seja baixo, os métodos de seleção Torneio são bem mais precisos.

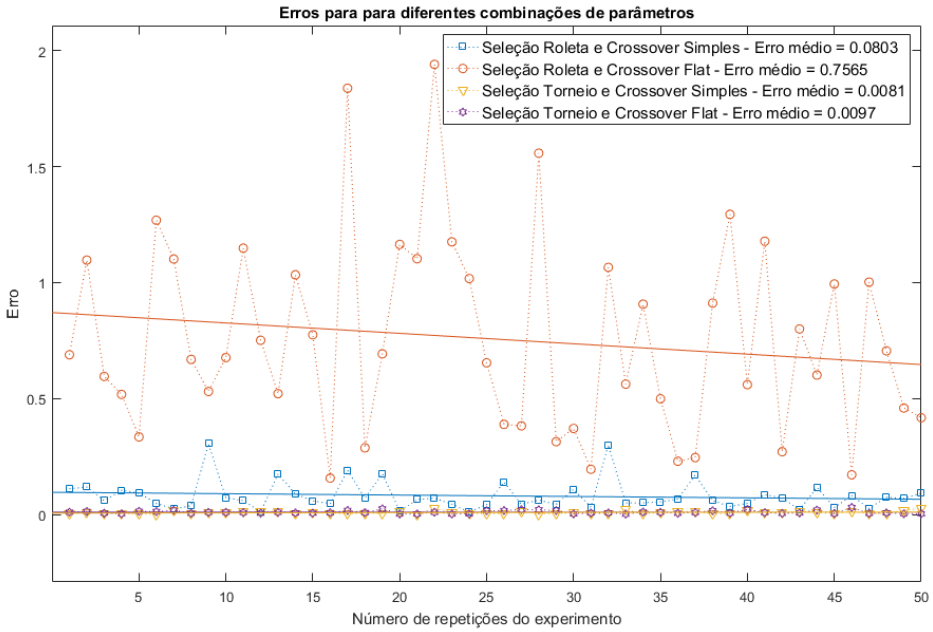

Figura 15. Gráfico de erros para cada combinação dos métodos de cruzamento e seleção implementados.

Abaixo temos o relatório exibido no MATLAB. São exibidos os valores dos parâmetros do algoritmo, o tempo total gasto com todas as repetições e o tempo e erro para cada combinação de dos métodos implementados.

```
Número de cromossomos: 200
Número de gerações: 200

Número de repetições para cada configuração: 50
Tempo total gasto: 72.70 s


======================== Resultados dos testes ========================

----- Método de seleção: ROLETA   Método de cruzamento: SIMPLES -----
Tempo total gasto nessa configuração: 9.19 s
Tempo médio de cada repetição: 0.18 s
Erro médio: 0.0803
-------------------------------------------------------------------------


------ Método de seleção: ROLETA   Método de cruzamento: FLAT -------
Tempo total gasto nessa configuração: 16.97 s
Tempo médio de cada repetição: 0.34 s
Erro médio: 0.7565
-------------------------------------------------------------------------


---- Método de seleção: TORNEIO   Método de cruzamento: SIMPLES -----
Tempo total gasto nessa configuração: 20.11 s
Tempo médio de cada repetição: 0.40 s
Erro médio: 0.0081
-------------------------------------------------------------------------


------ Método de seleção: TORNEIO   Método de cruzamento: FLAT ------
Tempo total gasto nessa configuração: 26.44 s
Tempo médio de cada repetição: 0.53 s
Erro médio: 0.0097
```

Abaixo temos os resultados do terceiro modo de operação do algoritmo. É possível perceber a influência do número de cromossomos tanto no tempo de execução, quanto no erro gerado. Nos gráficos da Figura 16, vemos que o tempo aumenta linearmente com o número de cromossomos, independentemente dos métodos de seleção e cruzamento utilizados. Contudo, como já sabemos, o tempo gasto em cada configuração é diferente.

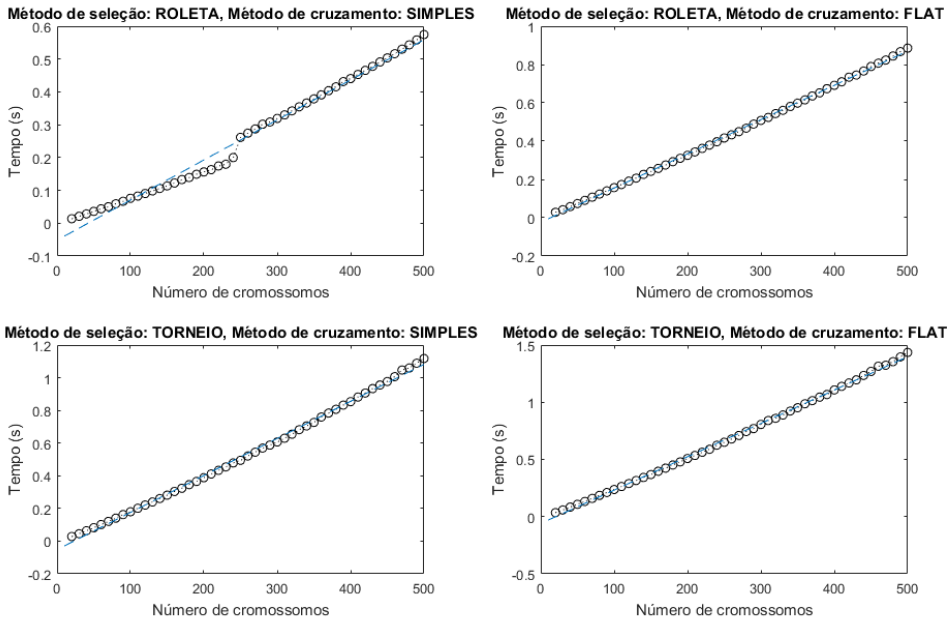

Figura 16. Influência do número de cromossomos no tempo de execução do algoritmo.

Nos gráficos de erro da Figura 17, vemos que o erro médio decai com o número de cromossomos. E percebemos que mesmo com 500 cromossomos, o erro no método de seleção Roleta com cruzamento Flat, ainda é alto se comparado aos demais. No método de seleção Torneio, o erro já começa muito baixo e cai rapidamente para zero, sendo assim, aumentar o número de cromossomos para um valor maior que 200 não gera nenhuma melhoria considerável, na verdade, apenas aumenta o tempo de processamento.

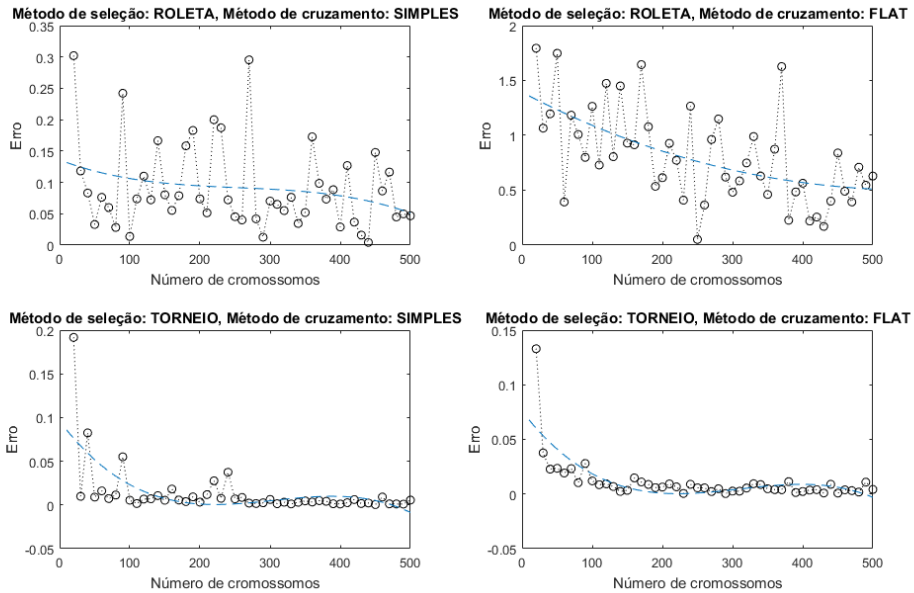

Figura 17. Influência do número de cromossomos no erro gerado pelo algoritmo.

6. CONCLUSÃO

Os Algoritmos Genéticos podem não ser a melhor solução de otimização, mas não há dúvidas que é uma técnica útil e eficiente para obter soluções de problemas complexos com multivariáveis.

Combinados com outros métodos ou até mesmo puro, GAs têm sido implementados com grande sucesso em muitos problemas. Empresas têm investido bastante na técnica, o que leva a se acreditar que há um futuro promissos para os GAs.

REFERÊNCIAS

Algoritmo Genético. Disponível em <https://pt.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico>. Acesso em 28 de junho de 2016.

Genetic Algorithm. Disponível em <http://www.icmc.usp.br/~andre/research/genetic/>. Acesso em 28 de junho de 2016

Fundamentos de Algoritmos Genéticos. Disponível em <http://www.nce.ufrj.br/GINAPE/VIDA/alggenet.htm>. Acesso em 29 de junho de 2016

.