Estudo e Simulação Métodos de Newton e Secante

Prof.ª Dr.ª Silvia Maria Simões de Carvalho
UFSCAR - Universidade Federal de São Carlos - Campus Sorocaba

Francisco Augusto Cesar De Camargo Bellaz Guiraldelli - RA 379840
Rafael Câmara Pereira - RA 380431

27/04/2015

Sumário

Introdução

O presente trabalho visa o estudo e a análise dos métodos de *Newton* e da *Secante*, através da implementação desses métodos em linguagem de programação específica utilizando Matlab.

Para conseguirmos estudar tais métodos, foi necessário a utilização de funções matemáticas, seus respectivos gráficos e após a execução dos métodos, as iterações necessárias para a conclusão de sua convergência ou, caso contrário, as causas de sua não convergência. Para tanto, estruturamos nosso relatório de uma forma clara, descrita abaixo.

Inicialmente no capítulo 1, descrevemos os objetivos, metodologias e as respectivas funções utilizadas e seus respectivos gráficos. Após, no capítulo 2 há a obtenção dos dados de convergência das funções através da execução dos algoritmos e análises das iterações executadas.

Finalmente no capítulo 3 refletimos sobre o comportamento das funções e efetuamos a conclusão dos métodos e das funções executadas por eles, seguido das referências bibliográficas e de um apêndice que mostra o código implementado e seus respectivos comentários para melhor entendimento.

Capítulo 1

Objetivos, Metodologia e Funções Utilizadas

1.1 Objetivos

O Objetivo do estudo é utilizar algumas funções polinomiais, logarítmicas, exponenciais e trigonométricas a fim de se observar o comportamento dos métodos de *Newton* e da *Secante* implementados. Dessa forma, serão feitas análises verificando-se as iterações, tabelas de valores e o tipo de gráfico que as funções apresentam.

1.2 Metodologia

Para implementar os métodos de *Newton* e da *Secante*, utilizamos três arquivos distintos descritos abaixo:

1 - `main.m` é o arquivo principal. Nele estão contidos as entradas de dados que o usuário fará para escolher o tipo do método a ser utilizado, os dados iniciais para a execução desses métodos, a plotagem da função escolhida, alguns tratamentos de erro, visando uma melhor interação humano - computador e a exibição dos resultados para o usuário.

2 - `newt.m` é o arquivo onde está implementado o método de *Newton*. Nele estão contidos as funções de derivada e as fórmulas necessárias para a execução desse método, além disso há também outras variáveis como contagem de iterações, contagem de tempo de execução e a plotagem das iterações na função.

3 - `sect.m` é o arquivo onde está implementado o método da *Secante*. Nele estão contidos as fórmulas necessárias para a execução desse método, além disso há também outras variáveis como contagem de iterações, contagem de tempo de execução e a plotagem das iterações na função.

1.3 Funções Utilizadas

Para a análise dos métodos, foram escolhidas algumas funções utilizadas em lista de exercícios, prova, literatura e pesquisas na internet. A escolha das funções está ligada ao comportamento dos métodos e na esperança de encontrarmos falhas relativas, principalmente quando há escolhas ruins em se tratando de valores iniciais e eficácia do método.

Abaixo, mostraremos as funções e seus respectivos gráficos relacionados:

1.3.1 Função $x^2 + 9$

Selecionamos a função pelo simples fato de não ter raiz, poderíamos assim, verificar qual o comportamento dos métodos ao utilizarmos esta função. Abaixo, podemos ver o gráfico dessa função:

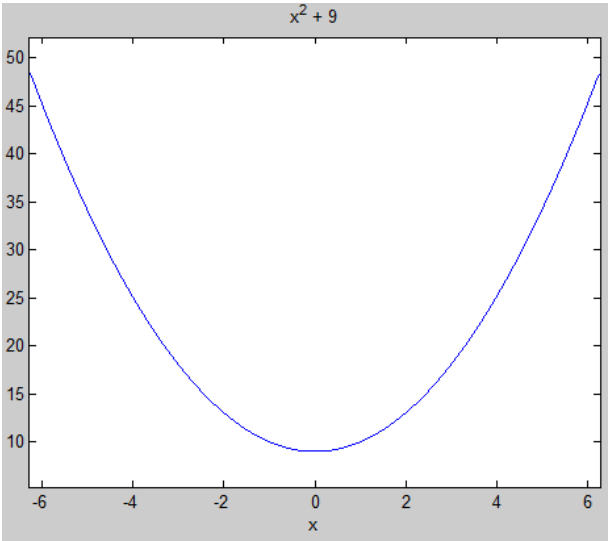

Figura 1.1: Gráfico da função $x^2 + 9$

1.3.2 Função $cos(x) - x$

Esta função foi escolhida pois, caso seja feita uma escolha ruim, o método de *Newton* pode não convergir por razões de *ciclagem*. Abaixo, podemos ver o gráfico dessa função:

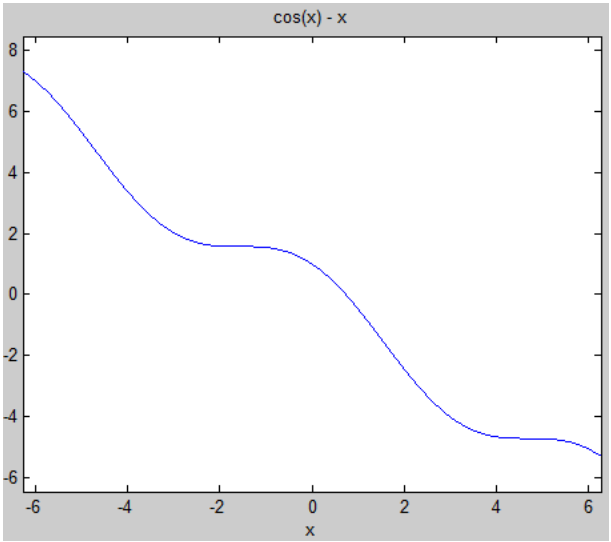

Figura 1.2: Gráfico da função $cos(x) - x$

1.3.3 Função $e^x - ln(x^2 + 1)$

Escolhemos esta função para analisar melhor os possíveis resultados da execução dela com outros pontos de entrada de dados, diferentes do ponto aplicado na prova de cálculo numérico. Vemos abaixo o gráfico da função:

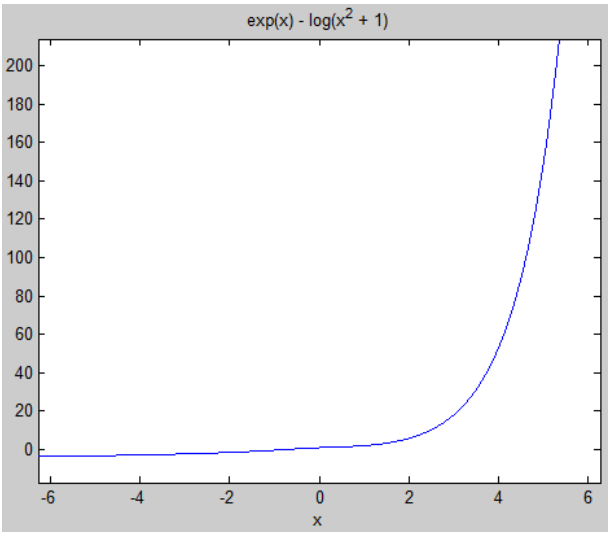

Figura 1.3: Gráfico da função $e^x - ln(x^2 + 1)$

1.3.4 Função $e^{-x^2} - cos(x)$

Optamos por esta função pois ela é uma função simétrica, ou par $f(x) = f(-x)$. Sua representação gráfica é:

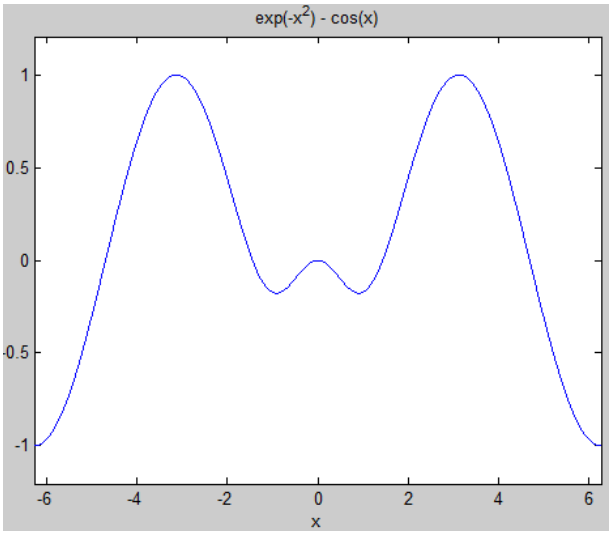

Figura 1.4: Gráfico da função $e^{-x^2} - cos(x)$

1.3.5 Função $x^3 - x - 1$

Selecionamos esta função porque a região próxima da raiz forma um pequeno intervalo quase horizontal, então nos interessamos em analisar os valores obtidos com a proximidade de 0. Representamos graficamente:

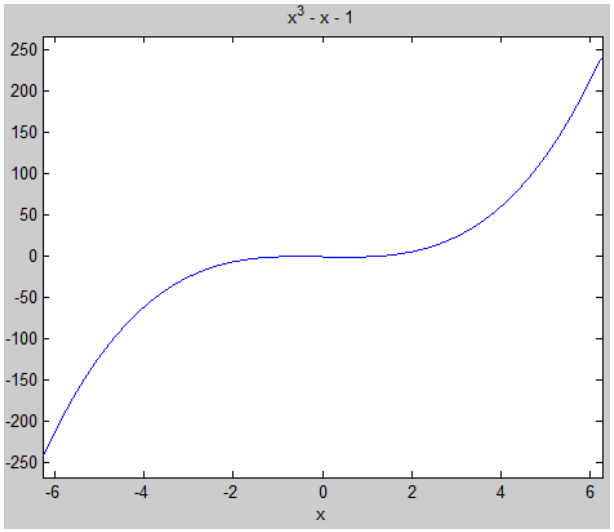

Figura 1.5: Gráfico da função $x^3 - x - 1x$

1.3.6 Função $4sen(x) - e^x$

Esta função foi escolhida pois temos raízes mais próximas entre si. Vemos o gráfico:

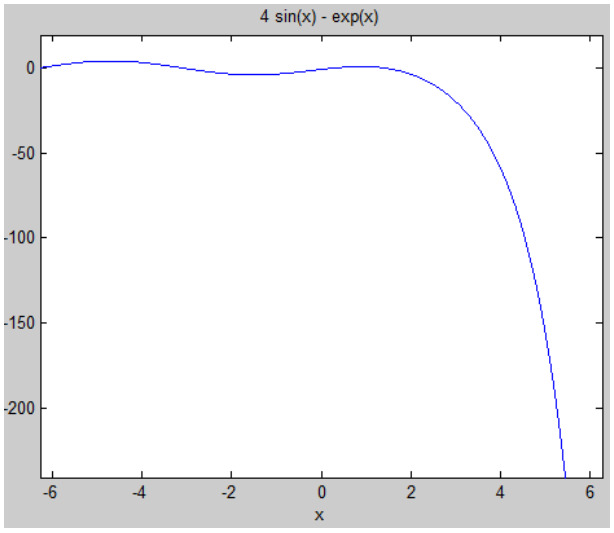

Figura 1.6: Gráfico da função $4sen(x) - e^x$

1.3.7 Função $xlog(x) - 1$

Escolhemos uma função logarítmica simples para analisar seu comportamento. Temos o gráfico abaixo:

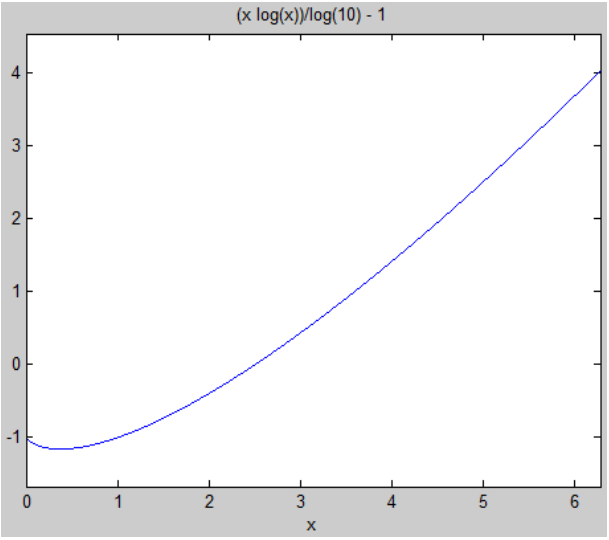

Figura 1.7: Gráfico da função $xlog(x) - 1$

Capítulo 2

Obtenção e Análise dos dados

2.1 Execução do algoritmo e dados obtidos

Para testar o programa utilizamos as funções mencionadas acima, alterando os dados de entrada e obtendo os seguintes resultados:

2.1.1 Função $x^2 + 9$

Erro	-4
x0 Newton	0.1
x0 Secante	-5
x1 Secante	3
Iterações Newton	-
Iterações Secante	-
Resultado Newton	-
Resultado Secante	-
Tempo Newton	-
Tempo Secante	-

2.1.2 Função $cos(x) - x$

Erro	-5	-5	-5	-5
x0 Newton	$\pi/4$	-1.69	-1.695	-1.7
x0 Secante	$\pi/4$	-5	-5	-1.7
x1 Secante	π	5	80	-1.6
Iterações Newton	3	12	310	15
Iterações Secante	5	7	7	9
Resultado Newton	0.7390851	0.7390851	0.7390851	0.7390851
Resultado Secante	0.7390851	0.7390851	0.7390851	0.7390851
Tempo Newton	0.23211	0.65497	15.0209	0.79858
Tempo Secante	0.38275	0.44576	0.44472	0.59067

2.1.3 Função $e^x - ln(x^2 + 1)$

Erro	-2	-5
x0 Newton	-1	1
x0 Secante	-1	1
x1 Secante	0	2
Iterações Newton	2	5
Iterações Secante	4	8
Resultado Newton	-0.7682198	-0.7682215
Resultado Secante	-0.7682215	-0.7682215
Tempo Newton	0.1826	0.32004
Tempo Secante	0.26449	0.47438

2.1.4 Função $e^{-x^2} - cos(x)$

Erro	-4	-4
x0 Newton	1.5	0.5
x0 Secante	1	-0.5
x1 Secante	2	0.5
Iterações Newton	3	12
Iterações Secante	7	-
Resultado Newton	1.447414	0.00008007797
Resultado Secante	1.447414	-
Tempo Newton	0.29201	0.71833
Tempo Secante	0.44571	-

2.1.5 Função $x^3 - x - 1$

Erro	-6	-6
x0 Newton	0	1.2
x0 Secante	0	1.2
x1 Secante	0.5	1.4
Iterações Newton	21	4
Iterações Secante	28	6
Resultado Newton	1.324718	1.324718
Resultado Secante	1.324718	1.324718
Tempo Newton	1.1184	0.34549
Tempo Secante	1.4854	0.42759

2.1.6 Função $4sen(x) - e^x$

Erro	-5	-5
x0 Newton	0.5	4.7
x0 Secante	0	0
x1 Secante	1	-2.5
Iterações Newton	4	68
Iterações Secante	8	11
Resultado Newton	0.3705581	1.364958
Resultado Secante	0.3705581	1.364958
Tempo Newton	0.23714	3.5099
Tempo Secante	0.47792	0.67573

2.1.7 Função $xlog(x) - 1$

Erro	-7
x0 Newton	2.5
x0 Secante	2.3
x1 Secante	2.7
Iterações Newton	3
Iterações Secante	5
Resultado Newton	2.506184
Resultado Secante	2.506184
Tempo Newton	0.21587
Tempo Secante	0.3394

2.2 Análise dos dados obtidos

Após as execuções, analisando os dados obtidos nas tabelas anteriores e os gráficos, podemos estudar os valores obtidos e discorrer sobre a dependência de certas escolhas para um bom funcionamento dos algoritmos. Mostramos as análises a seguir:

2.2.1 Função $e^{-x^2} - cos(x)$

Ao analisar esta função graficamente vemos que ela é uma função par $f(x) = f(-x)$ e achamos interessante efetuar um segundo teste no método da secante com valores $x0$ e $x1$ com paridade distinta $x0 = -x1$.

Partindo destes pontos, notamos que independentemente do número de iterações, a reta secante não é alterada, nunca chegando a um resultado final, como podemos mostrar nos gráficos a seguir:

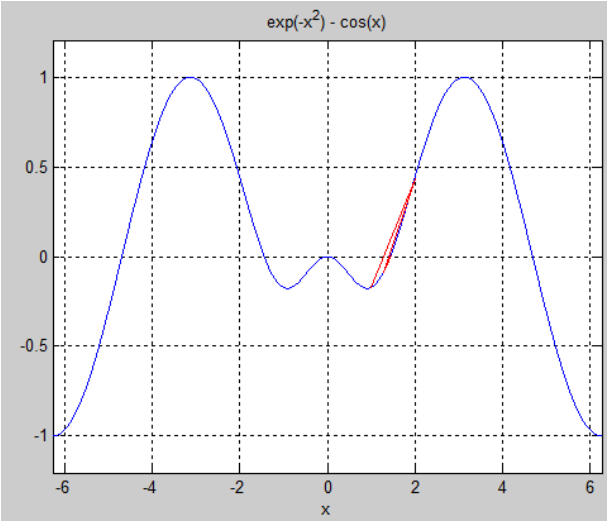

Figura 2.1: Método da secante com pontos distintos

2.2.2 Função $x^3 - x - 1$

Esta função nos mostra a importância da boa execução da fase 1, onde achamos o ponto $x0$ e $x1$.

Com um ponto mais afastado da raiz da função, temos cinco vezes mais execuções dos métodos em média, do que com uma boa fase 1.

Podemos notar também que algumas retas secantes estão indo para fora do gráfico, aumentando demasiadamente o valor de um dos pontos, isto se dá pela aproximação de 0 de $f(x_k) - f(x_{k-1})$, a parte divisora do método, que ocasiona um número que tende ao infinito.

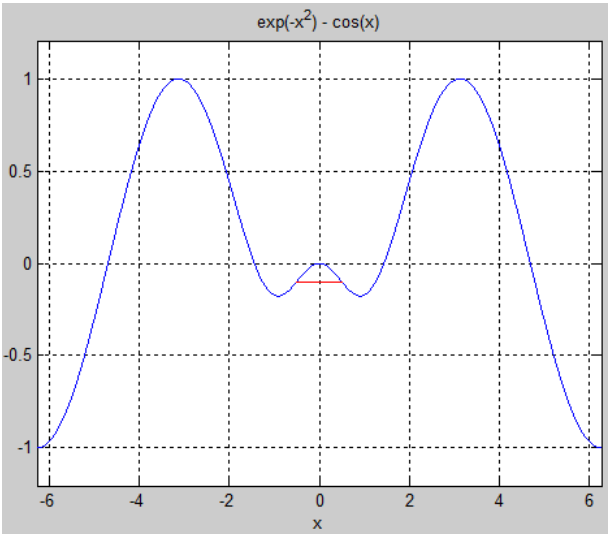

Figura 2.2: Método da secante com pontos opostos

```
O método executou em 0.34449 segundos
A quantidade de iteracoes efetuadas: 4
O valor do erro dado foi: 1e-06
O valor do erro calculado e: 7.158e-08
O valor de zero da raiz e: 1.324718
Pressione qualquer tecla para fechar
```

Figura 2.3: Fase 1 boa para o método de Newton

```
O método executou em 1.1534 segundos
A quantidade de iteracoes efetuadas: 21
O valor do erro dado foi: 1e-06
O valor do erro calculado e: 8.3137e-07
O valor de zero da raiz e: 1.324718
Pressione qualquer tecla para fechar
```

Figura 2.4: Fase 1 ruim para o método de Newton

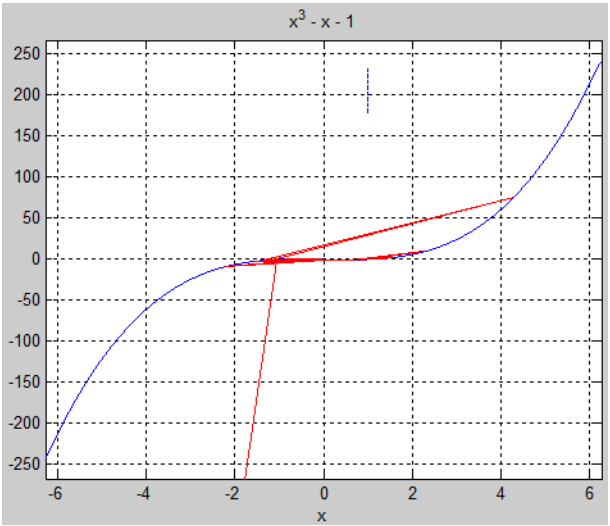

Figura 2.5: Explosão no cálculo da reta secante

2.2.3 Função $x^2 + 9$

Neste caso temos apenas uma função cuja $f''(x)$ existe, porém ela não possui raiz. Neste caso independentemente do número de iterações, a função nunca chegará a uma raiz aceitável também, sendo assim um desperdício computacional tentar executar o algoritmo com qualquer número de iterações.

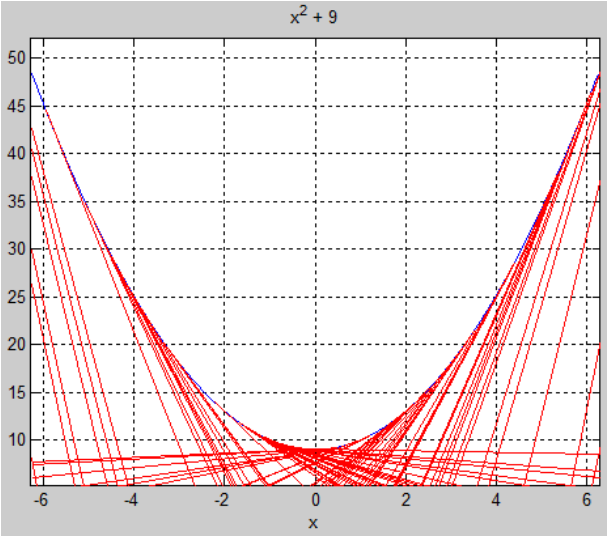

Figura 2.6: Cálculo do método de Newton em 100 iterações

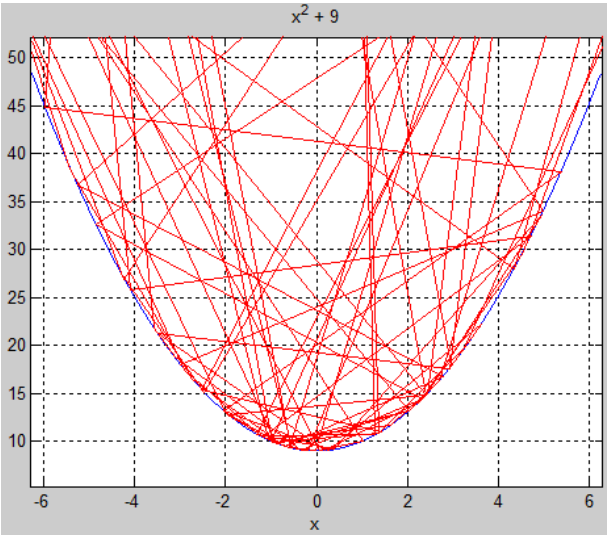

Figura 2.7: Cálculo do método da Secante em 100 iterações

2.2.4 Função $cos(x) - x$

Analisando os dados obtidos na execução desta função, não conseguimos pensar em um bom ponto para testar a ciclagem como queríamos inicialmente. Porém notamos que é uma função periódica com pequenos intervalos em que $f(x)$ tende a ser constante, graficamente a função apresenta pequenas retas horizontais, em que a inclinação da reta tangente tende a 0.

Assim sendo escolhemos três valores iniciais para o método de Newton, no meio de um destes intervalos e com uma diferença de $+-0.05$ entre eles.

No segundo e quarto casos, com $x0$ igual a -1.69 e -1.70, temos 12 e 15 iterações, respectivamente, para chegar ao mesmo resultado de raiz em 0.7390851, e uma diferença de 14 milésimos de segundo na execução destes.

Já no terceiro caso, escolhendo-se o ponto inicial -1.695, o método precisa de 310 iterações para encontrar exatamente a mesma raiz, demorando, para um problema pequeno, 15 segundos para encontrar um resultado. Isto nos mostrou a importância da análise da função no ponto encontrado na fase 1 a fim de verificar se ele pode causar computação excessiva para retornar algo fácil de ser calculado com outro ponto.

```
O metodo executou em 15.3705 segundos
A quantidade de iteracoes efetuadas: 310
O valor do erro dado foi: 1e-05
O valor do erro calculado e: 6.3118e-06
O valor de zero da raiz e: 0.7390851
Pressione qualquer tecla para fechar
```

Figura 2.8: Resultado do método de Newton com $x0 = -1.695$

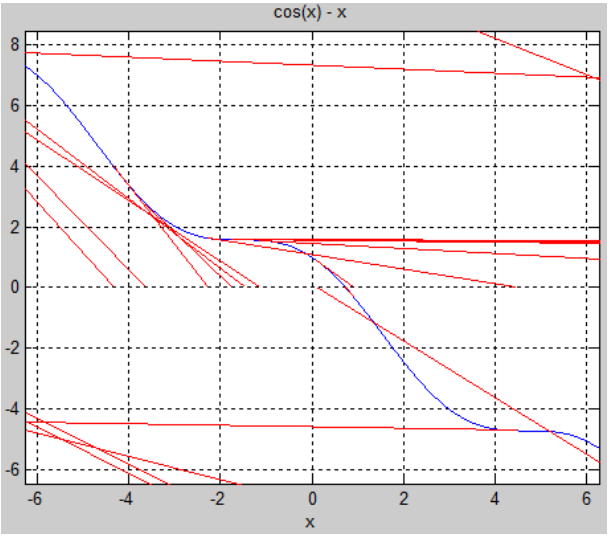

Figura 2.9: Gráfico do método de Newton com $x0 = -1.695$

2.2.5 Função $4sen(x) - e^x$

Temos aqui uma função com várias raízes, e nesta função utilizamos pontos diferentes em cada iteração dos métodos justamente para encontrar raízes distintas.

Como podemos alterar o ponto inicial para demonstrar uma escolha ruim de ponto inicial, na segunda iteração do método de *Newton* utilizamos o ponto 4.7, que ocasiona 68 iterações para chegar no mesmo resultado de raiz que a aplicação do método da *Secante* nos pontos 0 e -2.5.

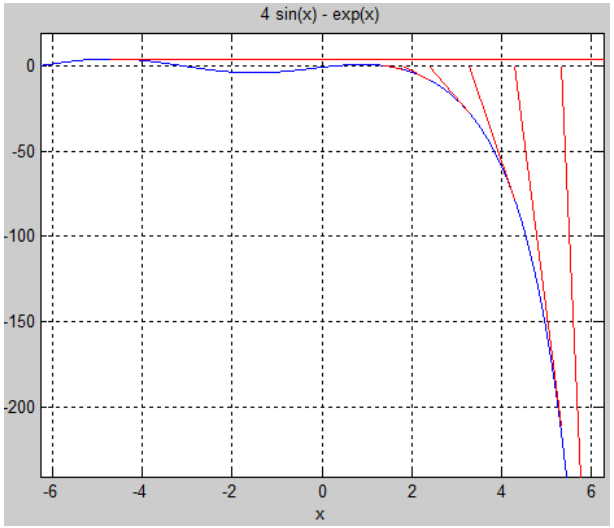

Figura 2.10: Gráfico do método de Newton com $x0 = 4.7$

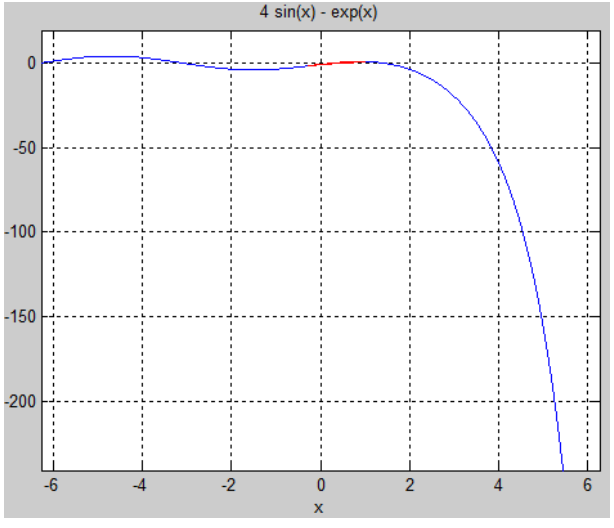

Figura 2.11: Gráfico do método da Secante com $x0 = 0$ e $x1 = 1$

Capítulo 3

Conclusão

Podemos notar que os métodos de *Newton* e da *Secante* são robustos para encontrar raízes de funções em problemas grandes, esparsos ou com muitos números que tendem a dar 0 ao serem truncados em outros métodos.

Notamos rapidez na resolução dos problemas, visto que eram pequenos e não estávamos muito preocupados com o tempo de execução no momento, já que não estávamos aplicando o método em campo.

Notamos também que alguns casos particulares, como funções pares, exigem cuidado na escolha da fase 1, principalmente para o método da secante, e que uma boa fase inicial pode diminuir significativamente o número de execuções do algoritmo.

Referências Bibliográficas

[1] Márcia A. Gomes Ruggiero, Vera Lúcia da Rocha Lopes.'Cálculo Numérico Aspectos Teóricos e Computacionais'.2ª Edição.Editora Makron Books.

[2] http://www.math.pitt.edu/ sussmanm/2070/lab_04/lab_04.html#NonQuadraticConvergence

[3] Prof.ª Dr.ª Silvia Maria Simões de Carvalho. Exercício da primeira avaliação de Cálculo Numérico - Abril 2015

[4] Prof.ª Dr.ª Silvia Maria Simões de Carvalho. Lista 2 de Exercícios de Cálculo Numérico.

Apêndice A

Apêndice

A.1 main.m

```
%
% Francisco Guiraldelli - 379840 - francisco.guiraldelli@gmail.com
% Rafael Camara Pereira - 380431 - rafael_c_pereira@hotmail.com
%
% Implementacao da rotina de execucao principal com entrada
% de dados pelo usuario, tratamento de erros, chamada das
% rotinas principais e exibicao de resultados
%

%Limpa as variaveis
clear all;
close all;
%Limpa a Tela
clc;
%Elimina mostra na tela de detalhes de tratamento de erro
warning off backtrace;

%Forcar formatacao dos numeros
format long;
format compact;

%Instancia as variaveis simbolicas
syms x x0 x1;
%Variavel booleana auxiliar
xpto = false;

%Entrada de Dados pelo usuario
%Tratamento de Erro
while xpto == false
    method = input('Digite 1 para utilizar o metodo de Newton ou 2
        para utilizar o metodo da Secante: ');
    if (method == 1) || (method == 2)
        xpto = true;
    else
        clc;
        warning('Valor nao corresponde a nenhum metodo');
```

```
36      end
37  end
38  fx = input('Digite a funcao: ');
39  x0 = input('Digite o valor de x0: ');
40  %Dado exclusivo do metodo da secante
41  if method == 2
42      x1 = input('Digite o valor de x1: ');
43  end
44  epsilon = input('Digite a precisao do erro que deseja: ');
45  iter = input('Digite a quantidade maxima de iteracoes: ');
46  %Tratamento de Erro
47  while xpto == true
48      precision=input('Digite a precisao em casas decimais(3 a 32): ');
49      if (precision>=3)&&(precision<=32)&&(precision>=abs(epsilon))
50          xpto = false;
51      else
52          clc;
53          if (precision < abs(epsilon))
54              warning('O valor de precisao deve ser maior ou igual a
                    precisao do erro');
55          end
56          if (precision < 3)||(precision > 32)||(isempty(precision))
57              warning('O valor de precisao deve estar entre 3 e 32');
58          end
59      end
60  end
61  %Tratamento especial de erro na mostra de precisao em unix
62  precision = abs(precision)-1;
63  epsilon = 10^epsilon;
64  %Plotagem do grafico da funcao
65  figure;
66  ezplot(fx);
67  hold on
68  grid on
69
70  %Chamada para o metodo escolhido
71  if method == 2
72      [iteration, calc_ep, final_result, time] = sect(fx, x0, x1,
            epsilon, precision, iter);
73  elseif method == 1
74      [iteration, calc_ep, final_result, time] = newt(fx, x0, epsilon,
            precision, iter);
75  end
76
77  %Mostra dados ao usuario
78  clc;
79  if iteration < iter
80      V=['O metodo executou em ', num2str(time), ' segundos'];
81      W=['A quantidade de iteracoes efetuadas: ', num2str(iteration)];
82      X=['O valor do erro dado foi: ', num2str(epsilon)];
83      Y=['O valor do erro calculado e: ', num2str(double(calc_ep))];
```

```
84      Z=['O valor de zero da raiz e: ', final_result];
85      clc
86      disp(V);
87      disp(W);
88      disp(X);
89      disp(Y);
90      disp(Z);
91  else
92      warning('Funcao ultrapassou o numero maximo de iteracoes');
93  end
94
95  disp('Pressione qualquer tecla para fechar');
96  pause();
97  clear all;
98  close all;
99  clc;
```

A.2 newt.m

```
%
% Francisco Guiraldelli - 379840 - francisco.guiraldelli@gmail.com
% Rafael Camara Pereira - 380431 - rafael_c_pereira@hotmail.com
%
% Implementacao das iteracoes do metodo de Newton, calculo de erro e
% resultado final com numero de iteracoes
%

function[iteration, calc_ep, final_result, time] = newt(fx, x0,
    epsilon, precision, iter)
    %Derivada da funcao
    dfx = diff(fx);
    %Variavel booleana auxiliar
    xpto = true;
    %Contador de iteracoes efetuadas
    iteration = 0;
    %Repeticao enquanto epsilon calculado for menor que
    %o pedido e nao ultrapassou o limite de iteracoes
    tic;
    while xpto && iteration < iter
        %Substituicao de x por valores de x0 na f(x) e f'(x)
        fa = subs(fx, x0);
        dfa = subs(dfx, x0);
        %Calculo do resultado utilizando a formula de Newton
        %com valor absoluto (positivo) e a precisao pedida
        result = vpa((x0 - (fa / dfa)), precision);
        %Calculo do erro na iteracao atual
        calc_ep = vpa(abs(x0 - result), precision);
        %Verifica a necessidade de mais iteracoes
        if (calc_ep <= epsilon)
            xpto = false;
        end
        %Plota a tangente da iteracao atual no grafico da funcao
        y = line([x0, result], [fa, 0], 'Color', [1, 0, 0]);
        plot(y);
        %Atualiza x0
        x0 = result;
        %Somador de iteracoes
        iteration = iteration + 1;
    end
    time = toc;
    %Tratamento do resultado final para exibicao
    final_result = char(result);
end
```

A.3 sect.m

```
%
% Francisco Guiraldelli - 379840 - francisco.guiraldelli@gmail.com
% Rafael Camara Pereira - 380431 - rafael_c_pereira@hotmail.com
%
% Implementacao das iteracoes do metodo da secante, calculo de erro e
% resultado final com numero de iteracoes
%

function[iteration, calc_ep, final_result, time] = sect(fx, x0, x1,
    epsilon, precision, iter)
    %Variavel booleana auxiliar
    xpto = true;
    %Contador de iteracoes efetuadas
    iteration = 0;
    %Repeticao enquanto epsilon calculado for menor que
    %o pedido e nao ultrapassou o limite de iteracoes
    tic;
    while xpto && iteration < iter
        %Substituicao de x por valores de x0 e x1 na f(x)
        fa = subs(fx, x0);
        fb = subs(fx, x1);
        %Calculo do resultado utilizando a formula da secante
        %com valor absoluto (positivo) e a precisao pedida
        result = vpa(((x0*fb - x1*fa)/(fb-fa)), precision);
        %Calculo do erro na iteracao atual
        calc_ep = vpa(abs(x0 - result), precision);
        %Verifica a necessidade de mais iteracoes
        if (calc_ep <= epsilon)
            xpto = false;
        end
        %Plota a tangente da iteracao atual no grafico da funcao
        y = line([x0, x1], [fa, fb], 'Color', [1, 0, 0]);
        plot(y);
        %Atualiza x0 e x1
        x0 = x1;
        x1 = result;
        %Somador de iteracoes
        iteration = iteration + 1;
    end
    time = toc;
    %Tratamento do resultado final para exibicao
    final_result = char(result);
end
```