

UMA ANÁLISE COMPARATIVA E IMPLEMENTAÇÃO DO MÉTODO DE NEWTON-RAPHSON PARA SOLUÇÃO DE EQUAÇÕES DE UMA VARIÁVEL

Métodos Numéricos

Professora: Prof^a Dr^a Claudia Mazza Dias
Alunos: Alexsander Andrade de Melo e Ygor de Mello Canalli

Universidade Federal Rural do Rio de Janeiro
Instituto Multidisciplinar

Nova Iguaçu - RJ, 19 de março de 2013

Resumo

Neste trabalho, apresentaremos um breve resumo teórico alguns dos principais métodos numéricos para solução de equações de uma variável. O Método principal deste estudo é o método de Newton-Raphson. Além de explicarmos o funcionamento de cada método, apresentaremos uma implementação em C e um estudo comparativo dos demais métodos com Newton-Raphson.

Sumário

1	Soluções de equações de uma variável	2
1.1	Método da bissecção	2
1.2	Método de Newton-Raphson	3
1.3	Método da secante	4
1.4	Método da Falsa Posição	4
2	Análise Comparativa	5
3	Implementações	8
	Referências Bibliográficas	37

1 Soluções de equações de uma variável

Os métodos numéricos aqui discutidos são utilizados para aproximar solução de equações que não são possíveis de ser solucionadas com valor exato através de método algébricos. Em especial, nos deteremos ao problema de encontrar a raiz de uma determinada função, isto é, encontrar x tal que $f(x) = 0$, denominado *zero* da função.

1.1 Método da bissecção

O primeiro método se baseia no Teorema do Valor Intermediário, a saber,

Teorema 1.1 (Teorema do Valor Intermediário). *Se $f \in C[a, b]$ e k for qualquer número entre $f(a)$ e $f(b)$ (isto é, $f(a) \leq k \leq f(b)$), então existe $c \in (a, b)$ para o qual*

$$f(c) = k.$$

O método divide iterativamente o intervalo em subintervalo $[a, b]$ e a cada passo localizando a metade do intervalo p . Encontramos o ponto médio p_1 de $[a, b]$, dado por

$$p_1 = a_1 + \frac{b_1 - a_1}{2} = \frac{a_1 + b_1}{2}.$$

Se $f(p_1) = 0$, temos a solução exata de nosso problema. Caso $\frac{a_1 + b_1}{2}$ seja menor que a tolerância, significa que temos uma solução p_1 dentro desta tolerância de erro.

Segue abaixo o método da bissecção:

Algoritmo 1.1 Método da bissecção

Entrada: Função f ; Extremidades a, b ; Tolerância $TOL \in \mathbb{R}$; Número máximo de iterações $N \in \mathbb{Z}$.

Saída: Solução aproximada p ou erro, onde $f(p) = 0$.

```
1: função BISSECÇÃO( $f, a, b, TOL, N$ )
2:    $i = 1$ 
3:    $FA = f(a)$ 
4:   enquanto  $i \leq N$  faça
5:      $p = a + (b - a)/2$ 
6:      $FP = f(p)$ 
7:     se  $FP = 0$  ou  $(b - a)/2 < TOL$  então
8:       retorne  $p$ 
9:     fim se
10:    se  $FA \cdot FP > 0$  então
11:       $a = p$ 
12:       $FA = FP$ 
13:    senão
14:       $b = p$ 
15:    fim se
16:     $i = i + 1$ 
17:  fim enquanto
18:  retorne ERRO
19: fim função
```

1.2 Método de Newton-Raphson

O método de Newton-Raphson é um dos métodos numéricos mais eficientes para o cálculo de raízes de uma equação. Seu funcionamento se baseia nos polinômios de Taylor, e é explicado abaixo.

Teorema 1.2. *Seja $f : \mathbb{R} \rightarrow \mathbb{R}$ com derivadas contínuas até ordem n :*

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \cdots + \frac{h^n}{n!}f^{(n)}(x) + R_n,$$

onde $h = x_n - x_{n-1}$ e

$$R_n = \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(\xi)$$

para algum $\xi \in [x, x+h]$.

Logo, se $x_n = x_{n-1} + \alpha$ e $f(x_{n-1} + \alpha) = 0$, então

$$0 = f(x_{n-1} + \alpha) = f(x_{n-1}) + \alpha f'(x_{n-1}) + \frac{\alpha^2}{2}f''(x_{n-1}) + \cdots$$

$$0 = f(x_{n-1} + \alpha) \approx f(x_{n-1}) + \alpha f'(x_{n-1}).$$

Portanto,

$$\alpha \approx -\frac{f(x_{n-1})}{f'(x_{n-1})} \implies x_n \approx x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}. \quad (1.1)$$

Teorema 1.3. *Seja $f \in C^2[a, b]$. Se $p \in [a, b]$ é tal que $f(p) = 0$ e $f'(p) \neq 0$, então, existe um $\delta > 0$ de forma que o método de Newton-Raphson gera uma sequência $\{p_n\}_{n=0}^\infty$ que converge para p para qualquer aproximação inicial $p_0 \in [p - \delta, p + \delta]$.*

Algoritmo 1.2 Método de Newton-Raphson

Entrada: Função f ; Aproximação inicial p_0 ; Tolerância $TOL \in \mathbb{R}$; Número máximo de iterações $N \in \mathbb{Z}$.

Saída: Solução aproximada p ou erro, onde $f(p) = 0$.

```
1: função NEWTON-RAPHSON( $f, p_0, TOL, N$ )
2:    $i = 1$ 
3:   enquanto  $i \leq N$  faça
4:      $p = p_0 - f(p_0)/f'(p_0)$ 
5:     se  $|p - p_0| < TOL$  então
6:       retorne  $p$ 
7:     fim se
8:      $p = p_0$ 
9:      $i = i + 1$ 
10:  fim enquanto
11:  retorne ERRO
12: fim função
```

1.3 Método da secante

Apesar do método de Newton-Raphson ser extremamente eficiente e possuir uma ótima convergência, nem sempre é possível calcular a derivada da função que se deseja encontrar o zero. Para isso, apresentamos o método da secante, que é uma alternativa para contornar tal problema.

Temos que

$$f'(x_{n-1}) = \lim_{x \rightarrow x_{n-1}} \frac{f(x) - f(x_{n-1})}{x - x_{n-1}}.$$

Tomando $x = x_{n-2}$, temos que

$$\begin{aligned} f'(x_{n-1}) &\approx \frac{f(x_{n-2}) - f(x_{n-1})}{x_{n-1} - x_{n-2}} \\ &= \frac{f(x_{n-2}) - f'(x_{n-1})}{x_{n-1} - x_{n-2}}. \end{aligned}$$

Substituindo f' na fórmula de Newton (1.1), obtemos

$$x_n = x_{n-1} - \frac{f(x_{n-1}) - f(x_{n-2})}{f(x_{n-1}) - f'(x_{n-1})}. \quad (1.2)$$

Algoritmo 1.3 Método da secante

Entrada: Função f ; Aproximações inicial p_0 e p_1 ; Tolerância $TOL \in \mathbb{R}$; Número máximo de iterações $N \in \mathbb{Z}$.

Saída: Solução aproximada p ou erro, onde $f(p) = 0$.

```
1: função SECANTE( $f, p_0, p_1, TOL, N$ )
2:    $i = 2$ 
3:    $q_0 = f(p_0)$ 
4:    $q_1 = f(p_1)$ 
5:   enquanto  $i \leq N$  faça
6:      $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$ 
7:     se  $|p - p_1| < TOL$  então
8:       retorne  $p$ 
9:     fim se
10:     $p_0 = p_1$ 
11:     $q_0 = q_1$ 
12:     $p_1 = p$ 
13:     $p_0 = p_1$ 
14:     $q_i = f(p)$ 
15:     $i = i + 1$ 
16:  fim enquanto
17:  retorne ERRO
18: fim função
```

1.4 Método da Falsa Posição

O método da falsa funciona da mesma maneira que o método da secante. A única modificação é que ele inclui um teste que garante que a aproximação

gerada não extrapolará o intervalo definido em cada iteração. O método da falsa posição geralmente não é indicado para fins práticos, pois sua maior segurança faz com que sejam necessários mais iterações que no método da secante. Entretanto, o apresentamos com fins teóricos, para ilustrar como podemos delimitar o intervalo no qual a solução aproximada será gerada.

Algoritmo 1.4 Método da falsa posição

Entrada: Função f ; Aproximações inicial p_0 e p_1 ; Tolerância $TOL \in \mathbb{R}$; Número máximo de iterações $N \in \mathbb{Z}$.

Saída: Solução aproximada p ou erro, onde $f(p) = 0$.

```

1: função FALSA-POSIÇÃO( $f, p_0, p_1, TOL, N$ )
2:    $i = 2$ 
3:    $q_0 = f(p_0)$ 
4:    $q_1 = f(p_1)$ 
5:   enquanto  $i \leq N$  faça
6:      $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$ 
7:     se  $|p - p_1| < TOL$  então
8:       retorne  $p$ 
9:     fim se
10:     $q = f(p)$ 
11:    se  $q \cdot q_1 < 0$  então
12:       $p_0 = p_1$ 
13:       $q_0 = q_1$ 
14:    fim se
15:     $p_1 = p$ 
16:     $q_1 = q$ 
17:     $i = i + 1$ 
18:  fim enquanto
19:  retorne ERRO
20: fim função

```

2 Análise Comparativa

Dos métodos apresentados acima, um dos mais utilizados por, de forma geral, possuir uma melhor convergência, como talvez já esperado, é o método de Newton-Rapson. No entanto, como vimos, nem sempre sua utilização é viável devido ao agravante da dificuldade de determinar a derivada de uma função, e quando isso ocorre, é preferível a utilização dos demais métodos. Dessa forma, não existe um método mais apropriado para o contexto geral, mas sim um método que é preferível de ser utilizado para um determinado problema, cabendo, então, uma análise detalhada do problema antes de se aplicar um método. Além disso, em muitos momentos é feita a utilização de mais de um método sucessivamente, por exemplo, é de comum ocorrência a utilização do método da Bisseção para uma aproximação inicial e após isso a utilização do método de Newton-Rapson usando os valores obtidos do método da Bisseção.

Exemplo 2.1. *Considere:*

1. $f(x) = 2x^3 + x^2 - 2$

2. Para $f(x) = 0, x \simeq 0.858094$
3. $TOL = 10^{-3}$
4. $N = 30$
5. Bisseção: $[-5, 5]$
6. Secante/Falsa Posição: $p_0 = -5; p_1 = 5$
7. Newton-Raphson: $p_0 = 5$

Iterações	Newton-Raphson	Bisseção	Secante	Falsa Posição
1	3.293750	0.000000	-0.460000	-0.460000
2	2.173285	2.500000	-0.420625	-0.420625
3	1.461878	1.250000	6.537428	-0.381752
4	1.056359	0.625000	-0.397814	-0.343281
5	0.889074	0.937500	-0.375127	-0.305134
6	0.859017	0.781250	15.538132	-0.267246
7	0.858095	0.859375	-0.371089	-0.229571
8		0.820312	-0.367054	-0.192076
9		0.839844	24.449435	-0.154741
10		0.849609	-0.365420	-0.117560
11		0.854492	-0.363786	-0.080536
12		0.856934	28.346145	-0.043687
13		0.858154	-0.362569	-0.007037
14		0.857544	-0.361353	0.029377
15			31.224187	0.065510
16			-0.360350	0.101311
17			-0.359348	0.136720
18			33.940445	0.171671
19			-0.358499	0.206096
20			-0.357650	0.239922
21				0.273077
22				0.305486
23				0.337079
24				0.367787
25				0.397547
26				0.426301
27				0.453998
28				0.480597
29				0.506063
30				0.506063
31				ERRO!

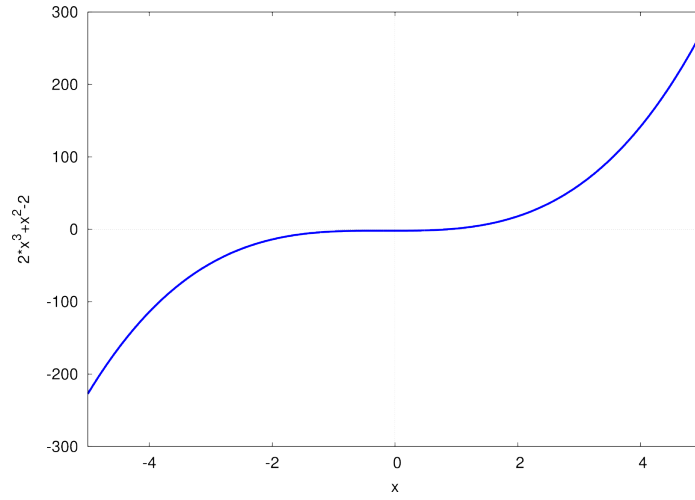


Figura 1: $f(x) = 2x^3 + x^2 - 2$

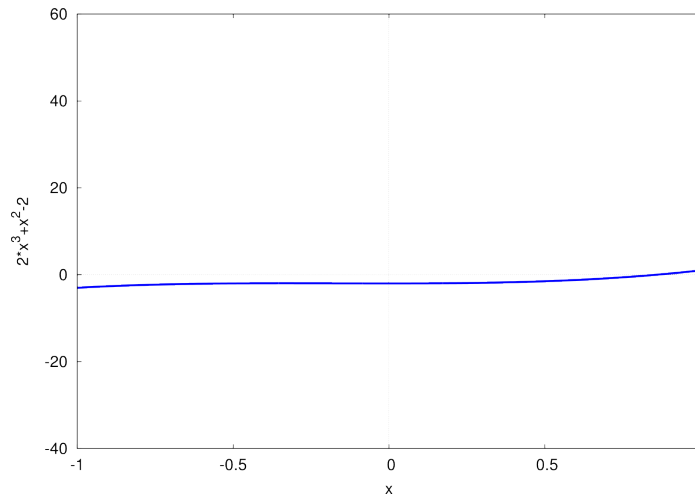


Figura 2: $f(x) = 2x^3 + x^2 - 2$

Como podemos notar pelos gráficos expostos acima e pela Tabela 2.1, que embora o método da secante tenha convergido para uma solução, o resultado retornando por este está incorreto. Isso ocorre, pois o Teorema 1.3 garante apenas que, sob as hipóteses aceitáveis, o método de Newton-Raphson (aqui, equivalentemente, estendido para o método da secante) convergirá se a aproximação inicial escolhida for suficientemente precisa.

Por outro lado, através ainda de uma análise da Tabela 2.1, percebemos nitidamente que, de modo geral, o método de Newton-Raphson converge muito mais rápido do que os demais métodos apresentados. Isso deve-se ao fato de que o método de Newton-Raphson é, na maioria dos casos, *quadraticamente convergente*, enquanto os outros não o são. Os conceitos de *quadraticamente convergente* e *linearmente convergente* são definidos abaixo.

Definição 2.1. Seja $\{p_n\}_{n=0}^{\infty}$ uma sequência que convirja para p , com $p_n \neq p$ para todo n . Se existem constantes positivas λ e α com

$$\lim_{n \rightarrow \infty} \frac{|p_{n+1} - p|}{|p_n - p|^\alpha} = \lambda,$$

então $\{p_n\}_{n=0}^{\infty}$ converge para p com ordem α .

Definição 2.2. Uma técnica iterativa da forma $p_n = g(p_{n-1})$ é dita da ordem α se a sequência $\{p_n\}_{n=0}^{\infty}$ convergir para a solução $p = g(p)$ com ordem α .

De modo geral, uma sequência com alta ordem de convergência converge mais rápido do que uma com menor ordem de convergência (ou seja, se $\alpha_1 > \alpha_2$, onde α_1 é a ordem de convergência uma determinada sequência $\{p_n\}_{n=0}^{\infty}$ e α_2 é a ordem de convergência de uma outra sequência $\{q_n\}_{n=0}^{\infty}$, então espera-se que $\{p_n\}_{n=0}^{\infty}$ convirja mais rapidamente do que $\{q_n\}_{n=0}^{\infty}$). Temos dois casos especiais de ordem de convergência, os quais são expostos abaixo:

- Se $\alpha = 1$, dizemos que a sequência é *linearmente convergente*.
- Se $\alpha = 2$, dizemos que a sequência é *quadraticamente convergente*.

Já o método da secante possui *convergência supralinear*, isto é,

$$1 < \alpha < 2.$$

Mais precisamente, neste caso,

$$\alpha = \frac{1}{2} \sqrt{1 + \sqrt{5}} \approx 1.6818.$$

3 Implementações

Por fim, apresentaremos nesta seção uma implementação dos algoritmos expostos acima. Optamos utilizar a *linguagem C* para este feito, devido a mesma ser uma linguagem de mais baixo nível (se comparado com as demais linguagens usuais como *Java*, *Python*, etc.), o que interfere diretamente, de forma positiva, no desempenho da execução do programa.

Tais implementações foram feitas seguindo a risca (com pequenas “modificações” apenas por conta da linguagem utilizada e da apresentação gráfica dos resultados) os algoritmos expostos ao longo deste trabalho. Cada implementação possui os respectivos métodos para determinar $f(x) = 0$, dada a função $f(x)$. Além desta determinação, a cada passo da execução dos métodos é gerado um gráfico da função $f(x)$ e o respectivo ponto $(x, f(x))$ para o qual $f(x) = 0$ (caso este seja determinado pelo método executado).

Ademais, para cada método foram feitas duas implementações, uma para quaisquer $f(x)$ que são funções polinomiais, onde o usuário entra via teclado com o valor de $f(x)$ e outra implementação para quaisquer funções $f(x)$ aceita pela biblioteca do C *math.h*. No entanto, se nesta última implementação $f(x)$ for alterado, então esta implementação deve ser sempre recompilada com a nova função $f(x)$ e, no caso do Newton-Raphson, também com a função $f'(x)$ correspondente. Segue abaixo cada uma dessas implementações.

Implementação 3.1: newtonRaphson.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "gnuplot.i.h"
6
7  #define MAX_STRING 80
8  #define SLEEP_LGTH 2
9
10
11 double polyValue(double* poly, int order, double value);
12 double* polyDerivative(double* poly, int order);
13 double newtonRaphson(double* poly, int polyOrder, double p0, double TOL, int N,
    int plot);
14 double doubleAbs(double a);
15 char* linePlotStringPoly(double x0, double y0, double x1, double y1);
16 double charToDouble(char* doubleInChar);
17 double* stringToDoublePoly(char* poly, int order);
18 char* doubleToStringPoly(double* poly, int order);
19
20 int main(int argc, char *argv[])
21 {
22
23     double zero;
24     double *poly;
25     double tol;
26     double p0;
27     int order;
28     int n;
29     int plot;
30
31     order = (int) strtoul(argv[2], 0, 10);
32     p0 = charToDouble(argv[3]);
33     tol = charToDouble(argv[4]);
34     n = (int) strtoul(argv[5], 0, 10);
35     plot = (int) strtoul(argv[6], 0, 10);
36
37     poly = stringToDoublePoly(argv[1], order);
38
39     zero = newtonRaphson(poly, order, p0, tol, n, plot);
40     printf("\n\nf(x) = 0; x = %f\n", zero);
41
42
43     return 0;
44 }
45
46 double doubleAbs(double a)
47 {
48     return (a < 0) ? (-1*a) : a;
49 }
50
51 double polyValue(double* poly, int order, double value)
52 {
53     int i;
54     double sum = 0;
55     for (i = 0; i < order+1; i++)
56         sum += poly[i]*pow(value,(double) i);
57     return sum;
58 }
59
60 double* polyDerivative(double* poly, int order)

```

```

61 {
62     int i;
63     double* derivative = malloc(order*sizeof(double));
64     for (i = 0; i < order; i++)
65         derivative[i] = (i+1)*poly[i+1];
66     derivative[order] = 0;
67     return derivative;
68 }
69
70 double newtonRaphson(double* poly, int polyOrder, double p0, double TOL, int N,
    int plot)
71 {
72     double* derivative = polyDerivative(poly, polyOrder);
73     int i = 0;
74     double p = 0;
75     double f = 0;
76     double fLinha = 0;
77     char* functLine;
78     gnuplot_ctrl *h1;
79
80
81     if (plot)
82     {
83         h1 = gnuplot_init();
84         gnuplot_plot_slope(h1, 1.0, 0.0, "f(x)" );
85         gnuplot_plot_slope(h1, 2.0, 0.0, "f'(x)" );
86
87         gnuplot_setstyle(h1, "lines");
88         gnuplot_cmd(h1, "set style line 1 lt 2 lw 15");
89
90         gnuplot_resetplot(h1);
91         gnuplot_plot_equation(h1, doubleToStringPoly(poly, polyOrder), "f(x)");
92         //sleep(SLEEP_LGTH);
93     }
94
95     while (i < N)
96     {
97
98         f = polyValue(poly, polyOrder, p0);
99         fLinha = polyValue(derivative, polyOrder, p0);
100         p = p0 - (f/fLinha);
101
102         if (plot)
103         {
104             functLine = linePlotStringPoly(p0, f, p, 0.00);
105             gnuplot_setstyle(h1, "lines");
106             gnuplot_plot_equation(h1, functLine, "");
107             sleep(SLEEP_LGTH);
108         }
109
110         if (doubleAbs(p - p0) < TOL)
111         {
112             if (plot)
113             {
114                 gnuplot_resetplot(h1);
115                 gnuplot_cmd(h1, "set label '(%f,0)' at %f,%f point
                    pointtype 2", p, p, 0.00);
116                 gnuplot_plot_equation(h1, doubleToStringPoly(poly, polyOrder), "f(
                    x)");
117                 getchar();
118                 gnuplot_close(h1);
119             }

```

```

120             printf("\nSolucao encontrada em %d iteracoes", i+1);
121             return p;
122         }
123
124         if (plot)
125             printf("\n%F", p);
126
127         p0 = p;
128         i++;
129     }
130     printf("\nLimite de %d iterecoes atingido. Nao foi possivel determinar o zero da
        funcao!\n", N);
131
132     getchar();
133     if (plot)
134         gnuplot_close(h1);
135
136     return p;
137 }
138
139 //return f(x) = ax + b
140 char* linePlotStringPoly(double x0, double y0, double x1, double y1)
141 {
142     char* line;
143     double* functDouble = (double*) malloc (sizeof(double) * 2);
144
145     functDouble[1] = (y1 - y0) / (x1 - x0); //a
146     functDouble[0] = y1 - (functDouble[1]*x1); //b
147
148     line = doubleToStringPoly(functDouble, 2);
149
150     return line;
151 }
152
153 double* stringToDoublePoly(char* poly, int order)
154 {
155     double* polyInDouble;
156
157     int i;
158     int j;
159     int beginVector;
160
161     polyInDouble = (double*) malloc (sizeof(double) * (order+1));
162
163     i = beginVector = j = 0;
164
165     while (poly[i] != '\0')
166     {
167         if (poly[i] == ',')
168         {
169             poly[i] = '\0';
170
171             polyInDouble[j] = charToDouble(poly + beginVector);
172             i++; //ignora espaco vazio
173
174             beginVector = i + 1;
175             j++; //quantidade de virgulas
176         }
177         i++;
178     }
179 }
180

```

```

181
182     polyInDouble[j] = charToDouble(poly + beginVector);
183
184     return polyInDouble;
185 }
186
187 char* doubleToStringPoly(double* poly, int order)
188 {
189     int i;
190     char aux[MAX_STRING];
191     char* result;
192
193     result = (char*) malloc (sizeof(char) * MAX_STRING * (order + 1));
194
195     for(i = 0; i < order; i++)
196     {
197         sprintf(aux, "%f*(x**%d) + ", poly[i], i);
198         strcat(result, aux);
199     }
200
201     sprintf(aux, "%f*(x**%d)", poly[i], i);
202     strcat(result, aux);
203
204     return result;
205 }
206
207 double charToDouble(char* doubleInChar)
208 {
209     int i = 0;
210     int j = 1;
211     int signal = 1;
212     double integer = 0;
213     double decimal = 0;
214
215     if (doubleInChar == NULL)
216         return 0;
217
218
219     if (doubleInChar[0] == '-') //Se o numero for negativo
220     {
221         signal = -1;
222         i = 1;
223     }
224
225     //Define a parte inteira
226     while ((doubleInChar[i] != '\0') && (doubleInChar[i] != ',') && (doubleInChar[i] !=
227         ','))
228     {
229         integer = (integer * 10) + (doubleInChar[i] - 48);
230         i++;
231     }
232
233     if (doubleInChar[i] != '\0')
234         i++; //pula a virgula
235
236     //Define a parte decimal
237     while (doubleInChar[i] != '\0')
238     {
239         decimal += (doubleInChar[i] - 48) / pow(10, j);
240         i++;
241         j++;
242     }

```

```

242         return signal*(integer + decimal);
243     }
244 }

```

Implementação 3.2: WithDefines/newtonRaphson.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "gnuplot.i.h"
6  #include "variables.h"
7
8  #define MAX_STRING 80
9  #define SLEEP_LGTH 2
10
11 double newtonRaphson(double p0, double TOL, int N, int plot);
12 double doubleAbs(double a);
13 char* linePlotStringPoly(double x0, double y0, double x1, double y1);
14 double charToDouble(char* doubleInChar);
15 char* doubleToStringPoly(double* poly, int order);
16
17 int main(int argc, char *argv[])
18 {
19
20     double zero;
21     double tol;
22     double p0;
23     int n;
24     int plot;
25
26     p0 = charToDouble(argv[1]);
27     tol = charToDouble(argv[2]);
28     n = (int) strtoul(argv[3], 0, 10);
29     plot = (int) strtoul(argv[4], 0, 10);
30
31     zero = newtonRaphson(p0, tol, n, plot);
32     printf("\n\nf(x) = 0; x = %f\n", zero);
33
34
35     return 0;
36 }
37
38 double doubleAbs(double a)
39 {
40     return (a < 0) ? (-1*a) : a;
41 }
42
43 double newtonRaphson(double p0, double TOL, int N, int plot)
44 {
45     int i = 0;
46     double p = 0;
47     double f = 0;
48     double fLinha = 0;
49     char* functLine;
50     gnuplot_ctrl *h1;
51
52
53     if (plot)
54     {
55         h1 = gnuplot_init();
56         gnuplot_plot_slope(h1, 1.0, 0.0, "f(x)");
57         gnuplot_plot_slope(h1, 2.0, 0.0, "f'(x)");

```

```

58         gnuplot_setstyle(h1, "lines");
59         gnuplot_cmd(h1, "set style line 1 lt 2 lw 15");
60
61         gnuplot_resetplot(h1);
62         gnuplot_plot_equation(h1, SFUNCTION(x), "f(x)");
63         //sleep(SLEEP_LGTH);
64     }
65
66     while (i < N)
67     {
68
69         f = FUNCTION(p0);
70         fLinha = DIFFUNCTION(p0);
71         p = p0 - (f/fLinha);
72
73
74
75         if (plot)
76         {
77             functLine = linePlotStringPoly(p0, f, p, 0.00);
78             gnuplot_setstyle(h1, "lines");
79             gnuplot_plot_equation(h1, functLine, "");
80             sleep(SLEEP_LGTH);
81         }
82
83         if (doubleAbs(p - p0) < TOL)
84         {
85             if (plot)
86             {
87                 gnuplot_resetplot(h1);
88                 gnuplot_cmd(h1, "set label '(%f,0)' at %f,%f point
89                     pointtype 2", p, p, 0.00);
90                 gnuplot_plot_equation(h1, SFUNCTION(x), "f(x)");
91                 getchar();
92                 gnuplot_close(h1);
93             }
94             printf("\nSolucao encontrada em %d iteracoes", i+1);
95             return p;
96         }
97
98         printf("\n%f", p);
99
100         p0 = p;
101         i++;
102     }
103     printf("\nLimite de %d iterecoes atingido. Nao foi possivel determinar o zero da
104         funcao!\n", N);
105
106     getchar();
107     if (plot)
108         gnuplot_close(h1);
109
110     return p;
111 }
112
113 //return f(x) = ax + b
114 char* linePlotStringPoly(double x0, double y0, double x1, double y1)
115 {
116     char* line;
117     double* functDouble = (double*) malloc (sizeof(double) * 2);
118
119     functDouble[1] = (y1 - y0) / (x1 - x0); //a

```

```

118         functDouble[0] = y1 - (functDouble[1]*x1); //b
119
120         line = doubleToStringPoly(functDouble, 2);
121
122         return line;
123     }
124
125
126     char* doubleToStringPoly(double* poly, int order)
127     {
128         int i;
129         char aux[MAX_STRING];
130         char* result;
131
132         result = (char*) malloc (sizeof(char) * MAX_STRING * (order + 1));
133
134         for(i = 0; i < order; i++)
135         {
136             sprintf(aux, "%f *(x**%d) + ", poly[i], i);
137             strcat(result, aux);
138         }
139
140         sprintf(aux, "%f *(x**%d)", poly[i], i);
141         strcat(result, aux);
142
143         return result;
144     }
145
146     double charToDouble(char* doubleInChar)
147     {
148         int i = 0;
149         int j = 1;
150         int signal = 1;
151         double integer = 0;
152         double decimal = 0;
153
154         if (doubleInChar == NULL)
155             return 0;
156
157
158         if (doubleInChar[0] == '-') //Se o numero for negativo
159         {
160             signal = -1;
161             i = 1;
162         }
163
164         //Define a parte inteira
165         while ((doubleInChar[i] != '\0') && (doubleInChar[i] != ',') && (doubleInChar[i] !=
166             ','))
167         {
168             integer = (integer * 10) + (doubleInChar[i] - 48);
169             i++;
170         }
171
172         if (doubleInChar[i] != '\0')
173             i++; //pula a virgula
174
175         //Define a parte decimal
176         while (doubleInChar[i] != '\0')
177         {
178             decimal += (doubleInChar[i] - 48) / pow(10, j);
179             i++;
180         }
181     }

```

```

179         j++;
180     }
181
182     return signal*(integer + decimal);
183 }

```

Implementação 3.3: bissecao.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "gnuplot.i.h"
6
7  #define MAX_STRING 80
8  #define SLEEP_LGTH 2
9
10
11 double polyValue(double* poly, int order, double value);
12 double* polyDerivative(double* poly, int order);
13 double bissecao(double* poly, int polyOrder, double a, double b, double TOL, int N,
    int plot);
14 double doubleAbs(double a);
15 char* linePlotStringPoly(double x0, double y0, double x1, double y1);
16 double charToDouble(char* doubleInChar);
17 double* stringToDoublePoly(char* poly, int order);
18 char* doubleToStringPoly(double* poly, int order);
19
20 int main(int argc, char *argv[])
21 {
22
23     double zero;
24     double *poly;
25     double tol;
26     double a,b;
27     int order;
28     int n;
29     int plot;
30
31     order = (int) strtoul(argv[2], 0, 10);
32     a = charToDouble(argv[3]);
33     b = charToDouble(argv[4]);
34     tol = charToDouble(argv[5]);
35     n = (int) strtoul(argv[6], 0, 10);
36     plot = (int) strtoul(argv[7], 0, 10);
37
38     poly = stringToDoublePoly(argv[1], order);
39
40     zero = bissecao(poly, order, a, b, tol, n, plot);
41     printf("\n\nf(x) = 0; x = %f\n", zero);
42
43
44     return 0;
45 }
46
47 double doubleAbs(double a)
48 {
49     return (a < 0) ? (-1*a) : a;
50 }
51
52 double polyValue(double* poly, int order, double value)
53 {
54     int i;

```



```

55     double sum = 0;
56     for (i = 0; i < order+1; i++)
57         sum += poly[i]*pow(value,(double) i);
58     return sum;
59 }
60
61 double* polyDerivative(double* poly, int order)
62 {
63     int i;
64     double* derivative = malloc(order*sizeof(double));
65     for (i = 0; i < order; i++)
66         derivative[i] = (i+1)*poly[i+1];
67     derivative[order] = 0;
68     return derivative;
69 }
70
71 double bissecao(double* poly, int polyOrder, double a, double b, double TOL, int N,
72     int plot)
73 {
74     int i = 0;
75     double p = 0;
76     double FA = 0;
77     double FP = 0;
78     gnuplot_ctrl *h1;
79
80     if (plot)
81     {
82         h1 = gnuplot_init();
83         gnuplot_plot_slope(h1, 1.0, 0.0, "f(x)" );
84
85         gnuplot_setstyle(h1, "lines");
86         gnuplot_cmd(h1, "set style line 1 lt 2 lw 15");
87     }
88
89     FA = polyValue(poly, polyOrder, a);
90
91     while (i < N)
92     {
93         p = a + (b-a)/2;
94         FP = polyValue(poly, polyOrder, p);
95
96         if ((FP == 0) || ((b-a)/2 < TOL))
97         {
98             if (plot)
99             {
100                 gnuplot_resetplot(h1);
101                 gnuplot_cmd(h1, "set label '(%f,0)' at %f,%f point
102                     pointtype 2", p, p, 0.00);
103                 gnuplot_plot_equation(h1, doubleToStringPoly(poly,
104                     polyOrder), "f(x)");
105                 getchar();
106                 gnuplot_close(h1);
107             }
108             printf("\nSolucao encontrada em %d iteracoes", i+1);
109             return p;
110         }
111
112         if (plot)
113             printf("\n%f", p);
114
115         if (FA*FP > 0)

```

```

114         {
115             a = p;
116             FA = FP;
117         }
118         else
119             b = p;
120         i++;
121     }
122     printf("\nLimite de %d iterecoes atingido. Nao foi possivel determinar o zero da
123         funcao!\n", N);
124
125     getchar();
126     if (plot)
127         gnuplot_close(h1);
128
129     return p;
130 }
131 //return f(x) = ax + b
132 char* linePlotStringPoly(double x0, double y0, double x1, double y1)
133 {
134     char* line;
135     double* functDouble = (double*) malloc (sizeof(double) * 2);
136
137     functDouble[1] = (y1 - y0) / (x1 - x0); //a
138     functDouble[0] = y1 - (functDouble[1]*x1); //b
139
140     line = doubleToStringPoly(functDouble, 2);
141
142     return line;
143 }
144
145 double* stringToDoublePoly(char* poly, int order)
146 {
147     double* polyInDouble;
148
149     int i;
150     int j;
151     int beginVector;
152
153     polyInDouble = (double*) malloc (sizeof(double) * (order+1));
154
155     i = beginVector = j = 0;
156
157     while (poly[i] != '\0')
158     {
159         if (poly[i] == ',')
160         {
161             poly[i] = '\0';
162
163             polyInDouble[j] = charToDouble(poly + beginVector);
164             i++; //ignora espaco vazio
165
166             beginVector = i + 1;
167             j++; //quantidade de virgulas
168         }
169
170         i++;
171     }
172
173     polyInDouble[j] = charToDouble(poly + beginVector);
174

```

```

175         return polyInDouble;
176     }
177 }
178
179 char* doubleToStringPoly(double* poly, int order)
180 {
181     int i;
182     char aux[MAX_STRING];
183     char* result;
184
185     result = (char*) malloc (sizeof(char) * MAX_STRING * (order + 1));
186
187     for(i = 0; i < order; i++)
188     {
189         sprintf(aux, "%f*(x**%d) + ", poly[i], i);
190         strcat(result, aux);
191     }
192
193     sprintf(aux, "%f*(x**%d)", poly[i], i);
194     strcat(result, aux);
195
196     return result;
197 }
198
199 double charToDouble(char* doubleInChar)
200 {
201     int i = 0;
202     int j = 1;
203     int signal = 1;
204     double integer = 0;
205     double decimal = 0;
206
207     if (doubleInChar == NULL)
208         return 0;
209
210
211     if (doubleInChar[0] == '-') //Se o numero for negativo
212     {
213         signal = -1;
214         i = 1;
215     }
216
217     //Define a parte inteira
218     while ((doubleInChar[i] != '\0') && (doubleInChar[i] != ',') && (doubleInChar[i] !=
219         ','))
220     {
221         integer = (integer * 10) + (doubleInChar[i] - 48);
222         i++;
223     }
224
225     if (doubleInChar[i] != '\0')
226         i++; //pula a virgula
227
228     //Define a parte decimal
229     while (doubleInChar[i] != '\0')
230     {
231         decimal += (doubleInChar[i] - 48) / pow(10, j);
232         i++;
233         j++;
234     }
235
236     return signal*(integer + decimal);

```

236 }

Implementação 3.4: WithDefines/bissecacao.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "gnuplot.i.h"
6
7  #define MAX_STRING 80
8  #define SLEEP_LGTH 2
9
10 #include "variables.h"
11
12 double bissecacao(double a, double b, double TOL, int N, int plot);
13 double doubleAbs(double a);
14 double charToDouble(char* doubleInChar);
15
16 int main(int argc, char *argv[])
17 {
18
19     double zero;
20     double tol;
21     double a,b;
22     int n;
23     int plot;
24
25     a = charToDouble(argv[1]);
26     b = charToDouble(argv[2]);
27     tol = charToDouble(argv[3]);
28     n = (int) strtoul(argv[4], 0, 10);
29     plot = (int) strtoul(argv[5], 0, 10);
30
31     zero = bissecacao(a, b, tol, n, plot);
32     printf("\n\nf(x) = 0; x = %f\n", zero);
33
34
35     return 0;
36 }
37
38 double doubleAbs(double a)
39 {
40     return (a < 0) ? (-1*a) : a;
41 }
42
43 double bissecacao(double a, double b, double TOL, int N, int plot)
44 {
45     int i = 0;
46     double p = 0;
47     double FA = 0;
48     double FP = 0;
49     gnuplot_ctrl *h1;
50
51     if (plot)
52     {
53         h1 = gnuplot_init();
54         gnuplot_plot_slope(h1, 1.0, 0.0, "f(x)");
55
56         gnuplot_setstyle(h1, "lines");
57         gnuplot_cmd(h1, "set style line 1 lt 2 lw 15");
58
59     }
```

```

60
61     FA = FUNCTION(a);
62
63     while (i < N)
64     {
65         p = a + (b-a)/2;
66         FP = FUNCTION(p);
67
68         if ((FP == 0) || ((b-a)/2 < TOL))
69         {
70             if (plot)
71             {
72                 gnuplot_resetplot(h1);
73                 gnuplot_cmd(h1, "set label '(%f,0)' at %f,%f point
74                     pointtype 2", p, p, 0.00);
75                 gnuplot_plot_equation(h1, SFUNCTION(p), "f(x)");
76                 getchar();
77                 gnuplot_close(h1);
78             }
79             printf("\nSolucao encontrada em %d iteracoes", i+1);
80             return p;
81         }
82
83         printf("\n%f", p);
84
85         if (FA*FP > 0)
86         {
87             a = p;
88             FA = FP;
89         }
90         else
91             b = p;
92         i++;
93     }
94     printf("\nLimite de %d iterecoes atingido. Nao foi possivel determinar o zero da
95         funcao!\n", N);
96
97     getchar();
98     if (plot)
99         gnuplot_close(h1);
100
101     return p;
102 }
103
104 double charToDouble(char* doubleInChar)
105 {
106     int i = 0;
107     int j = 1;
108     int signal = 1;
109     double integer = 0;
110     double decimal = 0;
111
112     if (doubleInChar == NULL)
113         return 0;
114
115     if (doubleInChar[0] == '-') //Se o numero for negativo
116     {
117         signal = -1;
118         i = 1;
119     }

```

```

120 //Define a parte inteira
121 while ((doubleInChar[i] != '\0') && (doubleInChar[i] != ',') && (doubleInChar[i] !=
    ','))
122 {
123     integer = (integer * 10) + (doubleInChar[i] - 48);
124     i++;
125 }
126
127 if (doubleInChar[i] != '\0')
128     i++; //pula a virgula
129
130 //Define a parte decimal
131 while (doubleInChar[i] != '\0')
132 {
133     decimal += (doubleInChar[i] - 48) / pow(10, j);
134     i++;
135     j++;
136 }
137
138 return signal*(integer + decimal);
139 }

```

Implementação 3.5: falsaPosicao.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "gnuplot.i.h"
6
7  #define MAX_STRING 80
8  #define SLEEP_LGTH 2
9
10
11 double polyValue(double* poly, int order, double value);
12 double falsaPosicao(double* poly, int polyOrder, double p0, double p1, double TOL,
    int N, int plot);
13 double doubleAbs(double a);
14 char* linePlotStringPoly(double x0, double y0, double x1, double y1);
15 double charToDouble(char* doubleInChar);
16 double* stringToDoublePoly(char* poly, int order);
17 char* doubleToStringPoly(double* poly, int order);
18
19 int main(int argc, char *argv[])
20 {
21
22     double zero;
23     double *poly;
24     double tol;
25     double p0;
26     double p1;
27     int order;
28     int n;
29     int plot;
30
31     order = (int) strtoul(argv[2], 0, 10);
32     p0 = charToDouble(argv[3]);
33     p1 = charToDouble(argv[4]);
34     tol = charToDouble(argv[5]);
35     n = (int) strtoul(argv[6], 0, 10);
36     plot = (int) strtoul(argv[7], 0, 10);
37
38     poly = stringToDoublePoly(argv[1], order);

```

```

39         zero = falsaPosicao(poly, order, p0, p1, tol, n, plot);
40         printf("\n\nf(x) = 0; x = %f\n", zero);
41
42
43
44         return 0;
45     }
46
47     double doubleAbs(double a)
48     {
49         return (a < 0) ? (-1*a) : a;
50     }
51
52     double polyValue(double* poly, int order, double value)
53     {
54         int i;
55         double sum = 0;
56         for (i = 0; i < order+1; i++)
57             sum += poly[i]*pow(value,(double) i);
58         return sum;
59     }
60
61
62     double falsaPosicao(double* poly, int polyOrder, double p0, double p1, double TOL,
63         int N, int plot)
64     {
65         int i;
66         double q0;
67         double q1;
68         double p;
69         double q;
70
71         char* functLine;
72         gnuplot_ctrl *h1;
73
74         i = 1;
75         q0 = polyValue(poly, polyOrder, p0);
76         q1 = polyValue(poly, polyOrder, p1);
77
78         if (plot)
79         {
80             h1 = gnuplot_init();
81             gnuplot_plot_slope(h1, 1.0, 0.0, "f(x)" );
82             gnuplot_plot_slope(h1, 2.0, 0.0, "f'(x)" );
83
84             gnuplot_setstyle(h1, "lines");
85             gnuplot_cmd(h1, "set style line 1 lt 2 lw 15");
86
87             gnuplot_resetplot(h1);
88             gnuplot_plot_equation(h1, doubleToStringPoly(poly, polyOrder), "f(x)");
89         }
90
91         while (i < N)
92         {
93             p = p1 - (q1*(p1 - p0)/(q1 - q0));
94             //printf("\n\n%f %f %f %f %f", p, p0, q0, p1, q1);
95
96             if (plot)
97             {
98                 functLine = linePlotStringPoly(p0, q0, p1, q1);
99                 gnuplot_setstyle(h1, "lines");

```

```

100         gnuplot_plot_equation(h1, functLine, "");
101         sleep(SLEEP_LGTH);
102     }
103
104     if (doubleAbs(p - p1) < TOL)
105     {
106         if (plot)
107         {
108             gnuplot_resetplot(h1);
109             gnuplot_cmd(h1, "set label '%f,0' at %f,%f point
110                             pointtype 2", p, p, 0.00);
111             gnuplot_plot_equation(h1, doubleToStringPoly(poly, polyOrder), "f(
112                                     x)");
113             getchar();
114             gnuplot_close(h1);
115         }
116
117         printf("\nSolucao encontrada em %d iteracoes", i+1);
118
119         return p;
120     }
121
122     if (plot)
123         printf("\n%f", p);
124
125     q = polyValue(poly, polyOrder, p);
126
127     if ((q * q1) < 0)
128     {
129         p0 = p1;
130         q0 = q1;
131     }
132
133     p1 = p;
134     q1 = q; //q1 = q
135
136     i++;
137 }
138
139 printf("\nLimite de %d iterecoes atingido. Nao foi possivel determinar o zero da
140 funcao!\n", N);
141
142 getchar();
143 if (plot)
144     gnuplot_close(h1);
145
146 return p;
147 }
148
149 //return f(x) = ax + b
150 char* linePlotStringPoly(double x0, double y0, double x1, double y1)
151 {
152     char* line;
153     double* functDouble = (double*) malloc (sizeof(double) * 2);
154
155     functDouble[1] = (y1 - y0) / (x1 - x0); //a
156     functDouble[0] = y1 - (functDouble[1]*x1); //b
157
158     line = doubleToStringPoly(functDouble, 2);
159
160     return line;
161 }

```



```

159 double* stringToDoublePoly(char* poly, int order)
160 {
161     double* polyInDouble;
162
163     int i;
164     int j;
165     int beginVector;
166
167     polyInDouble = (double*) malloc (sizeof(double) * (order+1));
168
169     i = beginVector = j = 0;
170
171     while (poly[i] != '\0')
172     {
173         if (poly[i] == ',')
174         {
175             poly[i] = '\0';
176
177             polyInDouble[j] = charToDouble(poly + beginVector);
178             i++; //ignora espaco vazio
179
180             beginVector = i + 1;
181             j++; //quantidade de virgulas
182         }
183         i++;
184     }
185
186     polyInDouble[j] = charToDouble(poly + beginVector);
187
188     return polyInDouble;
189 }
190
191 char* doubleToStringPoly(double* poly, int order)
192 {
193     int i;
194     char aux[MAX_STRING];
195     char* result;
196
197     result = (char*) malloc (sizeof(char) * MAX_STRING * (order + 1));
198
199     for(i = 0; i < order; i++)
200     {
201         sprintf(aux, "%f*(x**%d) + ", poly[i], i);
202         strcat(result, aux);
203     }
204
205     sprintf(aux, "%f*(x**%d)", poly[i], i);
206     strcat(result, aux);
207
208     return result;
209 }
210
211 double charToDouble(char* doubleInChar)
212 {
213     int i = 0;
214     int j = 1;
215     int signal = 1;
216     double integer = 0;
217     double decimal = 0;
218
219
220

```

```

221     if (doubleInChar == NULL)
222         return 0;
223
224
225     if (doubleInChar[0] == '-') //Se o numero for negativo
226     {
227         signal = -1;
228         i = 1;
229     }
230
231     //Define a parte inteira
232     while ((doubleInChar[i] != '\0') && (doubleInChar[i] != ',') && (doubleInChar[i] !=
        ':'))
233     {
234         integer = (integer * 10) + (doubleInChar[i] - 48);
235         i++;
236     }
237
238     if (doubleInChar[i] != '\0')
239         i++; //pula a virgula
240
241     //Define a parte decimal
242     while (doubleInChar[i] != '\0')
243     {
244         decimal += (doubleInChar[i] - 48) / pow(10, j);
245         i++;
246         j++;
247     }
248
249     return signal*(integer + decimal);
250 }

```

Implementação 3.6: WithDefines/falsaPosicao.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "gnuplot.i.h"
6
7  #include "variables.h"
8
9  #define MAX_STRING 80
10 #define SLEEP_LGTH 2
11
12 double falsaPosicao(double p0, double p1, double TOL, int N, int plot);
13 double doubleAbs(double a);
14 double charToDouble(char* doubleInChar);
15 char* linePlotStringPoly(double x0, double y0, double x1, double y1);
16 char* doubleToStringPoly(double* poly, int order);
17
18 int main(int argc, char *argv[])
19 {
20
21     double zero;
22     double tol;
23     double p0;
24     double p1;
25     int n;
26     int plot;
27
28     p0 = charToDouble(argv[1]);
29     p1 = charToDouble(argv[2]);

```

```

30     tol = charToDouble(argv[3]);
31     n = (int) strtoul(argv[4], 0, 10);
32     plot = (int) strtoul(argv[5], 0, 10);
33
34     zero = falsaPosicao(p0, p1, tol, n, plot);
35     printf("\n\nf(x) = 0; x = %f\n", zero);
36
37
38     return 0;
39 }
40
41 double doubleAbs(double a)
42 {
43     return (a < 0) ? (-1*a) : a;
44 }
45
46
47 double falsaPosicao(double p0, double p1, double TOL, int N, int plot)
48 {
49     int i;
50     double q0;
51     double q1;
52     double p;
53     double q;
54
55     char* functLine;
56     gnuplot_ctrl *h1;
57
58     i = 1;
59     q0 = FUNCTION(p0);
60     q1 = FUNCTION(p1);
61
62     if (plot)
63     {
64         h1 = gnuplot_init();
65         gnuplot_plot_slope(h1, 1.0, 0.0, "f(x)" );
66         gnuplot_plot_slope(h1, 2.0, 0.0, "f'(x)" );
67
68         gnuplot_setstyle(h1, "lines");
69         gnuplot_cmd(h1, "set style line 1 lt 2 lw 15");
70
71         gnuplot_resetplot(h1);
72         gnuplot_plot_equation(h1, SFUNCTION(x), "f(x)");
73     }
74
75     while (i < N)
76     {
77
78         p = p1 - (q1*(p1 - p0)/(q1 - q0));
79
80         if (plot)
81         {
82             functLine = linePlotStringPoly(p0, q0, p1, q1);
83             gnuplot_setstyle(h1, "lines");
84             gnuplot_plot_equation(h1, functLine, "");
85             sleep(SLEEP_LGTH);
86         }
87
88         if (doubleAbs(p - p1) < TOL)
89         {
90             if (plot)
91             {

```

```

92         gnuplot_resetplot(h1);
93         gnuplot_cmd(h1, "set label '(%f,0)' at %f,%f point
           pointtype 2", p, p, 0.00);
94         gnuplot_plot_equation(h1, SFUNCTION(x), "f(x)");
95         getchar();
96         gnuplot_close(h1);
97     }
98
99     printf("\nSolucao encontrada em %d iteracoes", i+1);
100
101     return p;
102 }
103 printf("\n%f", p);
104
105 q = FUNCTION(p);
106
107 if ((q * q1) < 0)
108 {
109     p0 = p1;
110     q0 = q1;
111 }
112
113 p1 = p;
114 q1 = q; //q1 = q
115
116 i++;
117 }
118 printf("\nLimite de %d iterecoes atingido. Nao foi possivel determinar o zero da
           funcao!\n", N);
119
120 getchar();
121 if (plot)
122     gnuplot_close(h1);
123
124 return p;
125 }
126
127 //return f(x) = ax + b
128 char* linePlotStringPoly(double x0, double y0, double x1, double y1)
129 {
130     char* line;
131     double* functDouble = (double*) malloc (sizeof(double) * 2);
132
133     functDouble[1] = (y1 - y0) / (x1 - x0); //a
134     functDouble[0] = y1 - (functDouble[1]*x1); //b
135
136     line = doubleToStringPoly(functDouble, 2);
137
138     return line;
139 }
140
141 char* doubleToStringPoly(double* poly, int order)
142 {
143     int i;
144     char aux[MAX_STRING];
145     char* result;
146
147     result = (char*) malloc (sizeof(char) * MAX_STRING * (order + 1));
148
149     for(i = 0; i < order; i++)
150     {
151         sprintf(aux, "%f *(x**%d) + ", poly[i], i);

```

```

152         strcat(result, aux);
153     }
154
155     sprintf(aux, "%f *(x**%d)", poly[i], i);
156     strcat(result, aux);
157
158     return result;
159 }
160
161 double charToDouble(char* doubleInChar)
162 {
163     int i = 0;
164     int j = 1;
165     int signal = 1;
166     double integer = 0;
167     double decimal = 0;
168
169     if (doubleInChar == NULL)
170         return 0;
171
172     if (doubleInChar[0] == '-') //Se o numero for negativo
173     {
174         signal = -1;
175         i = 1;
176     }
177
178     //Define a parte inteira
179     while ((doubleInChar[i] != '\0') && (doubleInChar[i] != ',') && (doubleInChar[i] !=
180         '.'))
181     {
182         integer = (integer * 10) + (doubleInChar[i] - 48);
183         i++;
184     }
185
186     if (doubleInChar[i] != '\0')
187         i++; //pula a virgula
188
189     //Define a parte decimal
190     while (doubleInChar[i] != '\0')
191     {
192         decimal += (doubleInChar[i] - 48) / pow(10, j);
193         i++;
194         j++;
195     }
196
197     return signal*(integer + decimal);
198 }

```

Implementação 3.7: secante.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "gnuplot.i.h"
6
7  #define MAX_STRING 80
8  #define SLEEP_LGTH 2
9
10
11 double polyValue(double* poly, int order, double value);

```

```

12 double secante(double* poly, int polyOrder, double p0, double p1, double TOL, int N,
    int plot);
13 double doubleAbs(double a);
14 char* linePlotStringPoly(double x0, double y0, double x1, double y1);
15 double charToDouble(char* doubleInChar);
16 double* stringToDoublePoly(char* poly, int order);
17 char* doubleToStringPoly(double* poly, int order);
18
19 int main(int argc, char *argv[])
20 {
21
22     double zero;
23     double *poly;
24     double tol;
25     double p0;
26     double p1;
27     int order;
28     int n;
29     int plot;
30
31     order = (int) strtoul(argv[2], 0, 10);
32     p0 = charToDouble(argv[3]);
33     p1 = charToDouble(argv[4]);
34     tol = charToDouble(argv[5]);
35     n = (int) strtoul(argv[6], 0, 10);
36     plot = (int) strtoul(argv[7], 0, 10);
37
38     poly = stringToDoublePoly(argv[1], order);
39
40     zero = secante(poly, order, p0, p1, tol, n, plot);
41     printf("\n\nf(x) = 0; x = %f\n", zero);
42
43
44     return 0;
45 }
46
47 double doubleAbs(double a)
48 {
49     return (a < 0) ? (-1*a) : a;
50 }
51
52 double polyValue(double* poly, int order, double value)
53 {
54     int i;
55     double sum = 0;
56     for (i = 0; i < order+1; i++)
57         sum += poly[i]*pow(value,(double) i);
58     return sum;
59 }
60
61
62 double secante(double* poly, int polyOrder, double p0, double p1, double TOL, int N,
    int plot)
63 {
64     int i;
65     double q0;
66     double q1;
67     double p;
68
69     char* functLine;
70     gnuplot_ctrl *h1;
71

```

```

72     i = 1;
73     q0 = polyValue(poly, polyOrder, p0);
74     q1 = polyValue(poly, polyOrder, p1);
75
76     if (plot)
77     {
78         h1 = gnuplot_init();
79         gnuplot_plot_slope(h1, 1.0, 0.0, "f(x)" );
80         gnuplot_plot_slope(h1, 2.0, 0.0, "f'(x)" );
81
82         gnuplot_setstyle(h1, "lines");
83         gnuplot_cmd(h1, "set style line 1 lt 2 lw 15");
84
85         gnuplot_resetplot(h1);
86         gnuplot_plot_equation(h1, doubleToStringPoly(poly, polyOrder), "f(x)");
87     }
88
89     while (i < N)
90     {
91
92         p = p1 - (q1*(p1 - p0)/(q1 - q0));
93
94         if (plot)
95         {
96             functLine = linePlotStringPoly(p0, q0, p1, q1);
97             gnuplot_setstyle(h1, "lines");
98             gnuplot_plot_equation(h1, functLine, "");
99             sleep(SLEEP_LGTH);
100         }
101
102         if (doubleAbs(p - p1) < TOL)
103         {
104             if (plot)
105             {
106                 gnuplot_resetplot(h1);
107                 gnuplot_cmd(h1, "set label '(%f,0)' at %f,%f point
108                             pointtype 2", p, p, 0.00);
109                 gnuplot_plot_equation(h1, doubleToStringPoly(poly, polyOrder), "f(
110                                     x)");
111                 getchar();
112                 gnuplot_close(h1);
113             }
114
115             printf("\nSolucao encontrada em %d iteracoes", i+1);
116
117             return p;
118         }
119
120         if (plot)
121             printf("\n%f", p);
122
123         p0 = p1;
124         q0 = q1;
125         p1 = p;
126         q1 = polyValue(poly, polyOrder, p); //q1 = f(p)
127
128         i++;
129     }
130     printf("\nLimite de %d iterecoes atingido. Nao foi possivel determinar o zero da
131         funcao!\n", N);
132
133     getchar();

```

```

131         if (plot)
132             gnuplot_close(h1);
133
134         return p;
135     }
136
137     //return f(x) = ax + b
138     char* linePlotStringPoly(double x0, double y0, double x1, double y1)
139     {
140         char* line;
141         double* functDouble = (double*) malloc (sizeof(double) * 2);
142
143         functDouble[1] = (y1 - y0) / (x1 - x0); //a
144         functDouble[0] = y1 - (functDouble[1]*x1); //b
145
146         line = doubleToStringPoly(functDouble, 2);
147
148         return line;
149     }
150
151     double* stringToDoublePoly(char* poly, int order)
152     {
153         double* polyInDouble;
154
155         int i;
156         int j;
157         int beginVector;
158
159         polyInDouble = (double*) malloc (sizeof(double) * (order+1));
160
161         i = beginVector = j = 0;
162
163         while (poly[i] != '\0')
164         {
165             if (poly[i] == ',')
166             {
167                 poly[i] = '\0';
168
169                 polyInDouble[j] = charToDouble(poly + beginVector);
170                 i++; //ignora espaco vazio
171
172                 beginVector = i + 1;
173                 j++; //quantidade de virgulas
174             }
175
176             i++;
177         }
178
179         polyInDouble[j] = charToDouble(poly + beginVector);
180
181         return polyInDouble;
182     }
183
184     char* doubleToStringPoly(double* poly, int order)
185     {
186         int i;
187         char aux[MAX_STRING];
188         char* result;
189
190         result = (char*) malloc (sizeof(char) * MAX_STRING * (order + 1));
191
192

```



```

193     for(i = 0; i < order; i++)
194     {
195         sprintf(aux, "%f*(x**%d) + ", poly[i], i);
196         strcat(result, aux);
197     }
198
199     sprintf(aux, "%f*(x**%d)", poly[i], i);
200     strcat(result, aux);
201
202     return result;
203 }
204
205 double charToDouble(char* doubleInChar)
206 {
207     int i = 0;
208     int j = 1;
209     int signal = 1;
210     double integer = 0;
211     double decimal = 0;
212
213     if (doubleInChar == NULL)
214         return 0;
215
216
217     if (doubleInChar[0] == '-') //Se o numero for negativo
218     {
219         signal = -1;
220         i = 1;
221     }
222
223     //Define a parte inteira
224     while ((doubleInChar[i] != '\0') && (doubleInChar[i] != ',') && (doubleInChar[i] !=
225         ','))
226     {
227         integer = (integer * 10) + (doubleInChar[i] - 48);
228         i++;
229     }
230
231     if (doubleInChar[i] != '\0')
232         i++; //pula a virgula
233
234     //Define a parte decimal
235     while (doubleInChar[i] != '\0')
236     {
237         decimal += (doubleInChar[i] - 48) / pow(10, j);
238         i++;
239         j++;
240     }
241
242     return signal*(integer + decimal);
243 }

```

Implementação 3.8: WithDefines/secante.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "gnuplot.i.h"
6  #include "variables.h"
7
8  #define MAX_STRING 80
9  #define SLEEP_LENGTH 2

```

```

10
11
12 double secante(double p0, double p1, double TOL, int N, int plot);
13 double doubleAbs(double a);
14 char* linePlotStringPoly(double x0, double y0, double x1, double y1);
15 double charToDouble(char* doubleInChar);
16 char* doubleToStringPoly(double* poly, int order);
17
18 int main(int argc, char *argv[])
19 {
20
21     double zero;
22     double tol;
23     double p0;
24     double p1;
25     int n;
26     int plot;
27
28     p0 = charToDouble(argv[1]);
29     p1 = charToDouble(argv[2]);
30     tol = charToDouble(argv[3]);
31     n = (int) strtoul(argv[4], 0, 10);
32     plot = (int) strtoul(argv[5], 0, 10);
33
34     zero = secante(p0, p1, tol, n, plot);
35     printf("\n\nf(x) = 0; x = %f\n", zero);
36
37
38     return 0;
39 }
40
41 double doubleAbs(double a)
42 {
43     return (a < 0) ? (-1*a) : a;
44 }
45
46
47 double secante(double p0, double p1, double TOL, int N, int plot)
48 {
49     int i;
50     double q0;
51     double q1;
52     double p;
53
54     char* functLine;
55     gnuplot_ctrl *h1;
56
57     i = 1;
58     q0 = FUNCTION(p0);
59     q1 = FUNCTION(p1);
60
61     if (plot)
62     {
63         h1 = gnuplot_init();
64         gnuplot_plot_slope(h1, 1.0, 0.0, "f(x)" );
65         gnuplot_plot_slope(h1, 2.0, 0.0, "f'(x)" );
66
67         gnuplot_setstyle(h1, "lines");
68         gnuplot_cmd(h1, "set style line 1 lt 2 lw 15");
69
70         gnuplot_resetplot(h1);
71         gnuplot_plot_equation(h1, SFUNCTION(x), "f(x)");

```

```

72     }
73
74     while (i < N)
75     {
76
77         p = p1 - (q1*(p1 - p0)/(q1 - q0));
78
79         if (plot)
80         {
81             functLine = linePlotStringPoly(p0, q0, p1, q1);
82             gnuplot_setstyle(h1, "lines");
83             gnuplot_plot_equation(h1, functLine, "");
84             sleep(SLEEP_LGTH);
85         }
86
87         if (doubleAbs(p - p1) < TOL)
88         {
89             if (plot)
90             {
91                 gnuplot_resetplot(h1);
92                 gnuplot_cmd(h1, "set label '(%f,0)' at %f,%f point
93                                     pointtype 2", p, p, 0.00);
94                 gnuplot_plot_equation(h1, SFUNCTION(x), "f(x)");
95                 getchar();
96                 gnuplot_close(h1);
97             }
98             printf("\nSolucao encontrada em %d iteracoes", i+1);
99
100            return p;
101        }
102
103        p0 = p1;
104        q0 = q1;
105        p1 = p;
106        q1 = FUNCTION(p); //q1 = f(p)
107
108        i++;
109    }
110    printf("\nLimite de %d iterecoes atingido. Nao foi possivel determinar o zero da
111           funcao!\n", N);
112
113    getchar();
114    if (plot)
115        gnuplot_close(h1);
116
117    return p;
118 }
119 //return f(x) = ax + b
120 char* linePlotStringPoly(double x0, double y0, double x1, double y1)
121 {
122     char* line;
123     double* functDouble = (double*) malloc (sizeof(double) * 2);
124
125     functDouble[1] = (y1 - y0)/ (x1 - x0); //a
126     functDouble[0] = y1 - (functDouble[1]*x1); //b
127
128     line = doubleToStringPoly(functDouble, 2);
129
130     return line;
131 }

```

```

132
133
134 double* stringToDoublePoly(char* poly, int order)
135 {
136     double* polyInDouble;
137
138     int i;
139     int j;
140     int beginVector;
141
142     polyInDouble = (double*) malloc (sizeof(double) * (order+1));
143
144     i = beginVector = j = 0;
145
146     while (poly[i] != '\0')
147     {
148         if (poly[i] == ',')
149         {
150             poly[i] = '\0';
151
152             polyInDouble[j] = charToDouble(poly + beginVector);
153             i++; //ignora espaco vazio
154
155             beginVector = i + 1;
156             j++; //quantidade de virgulas
157         }
158         i++;
159     }
160
161     polyInDouble[j] = charToDouble(poly + beginVector);
162
163     return polyInDouble;
164 }
165
166 char* doubleToStringPoly(double* poly, int order)
167 {
168     int i;
169     char aux[MAX_STRING];
170     char* result;
171
172     result = (char*) malloc (sizeof(char) * MAX_STRING * (order + 1));
173
174     for(i = 0; i < order; i++)
175     {
176         sprintf(aux, "%f*(x**%d) + ", poly[i], i);
177         strcat(result, aux);
178     }
179
180     sprintf(aux, "%f*(x**%d)", poly[i], i);
181     strcat(result, aux);
182
183     return result;
184 }
185
186 double charToDouble(char* doubleInChar)
187 {
188     int i = 0;
189     int j = 1;
190     int signal = 1;
191     double integer = 0;
192     double decimal = 0;
193

```

```

194
195     if (doubleInChar == NULL)
196         return 0;
197
198
199     if (doubleInChar[0] == '-') //Se o numero for negativo
200     {
201         signal = -1;
202         i = 1;
203     }
204
205     //Define a parte inteira
206     while ((doubleInChar[i] != '\0') && (doubleInChar[i] != ',') && (doubleInChar[i] !=
207         ','))
208     {
209         integer = (integer * 10) + (doubleInChar[i] - 48);
210         i++;
211     }
212
213     if (doubleInChar[i] != '\0')
214         i++; //pula a virgula
215
216     //Define a parte decimal
217     while (doubleInChar[i] != '\0')
218     {
219         decimal += (doubleInChar[i] - 48) / pow(10, j);
220         i++;
221         j++;
222     }
223
224     return signal*(integer + decimal);
225 }

```

Referências Bibliográficas

- [1] Richard L. Burden and J. Douglas Faires. Análise Numérica. Cengage, tradução da 8ª edição norte-americana edition, 2008.
- [2] GNU Plot. Documentation of gnuplot 4.2, An Interactive Plotting Program, Setembro 2009. Disponível em <http://www.gnuplot.info/docs_4.2/gnuplot.html>.