

Utilização de Métodos Numéricos para Encontrar Raízes

Jhonattan C. B. Cabral¹, Daniel M. P. Carvalho¹

¹Instituto de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte (UFRN)

jhonattan.yoru@gmail.com, danielmarx08@gmail.com

Abstract. Task applied in the first unit of the Numerical Calculus discipline (DIM0404), aims to use some numerical methods to find approximate roots for some specific problems.

Resumo. Tarefa aplicada na primeira unidade da disciplina de Cálculo Numérico (DIM0404), tem como objetivo utilizar de alguns métodos numéricos para encontrar raízes aproximadas para alguns determinados problemas.

1. Descreva o método da Bissecção e utilize-o para encontrar as raízes de:

$$f(x) = x^3 + 4.6x^2 + 1.6x - 7.2.$$

O método da Bissecção trata-se da primeira técnica baseada no Teorema do Valor Intermediário para encontrar raízes aproximadas. Suponha que f seja uma função contínua definida no intervalo $[a, b]$, com $f(a)$ e $f(b)$ de sinais opostos. De acordo com o Teorema do Valor Intermediário, existe um número p em (a, b) com $f(p) = 0$. [Burden and Faires 2008]

1.1. Gráfico

Visando a facilitação da aplicação do método numérico, optamos em estudar o gráfico, possibilitando assim a aplicação de intervalos tendenciosos no algoritmo.

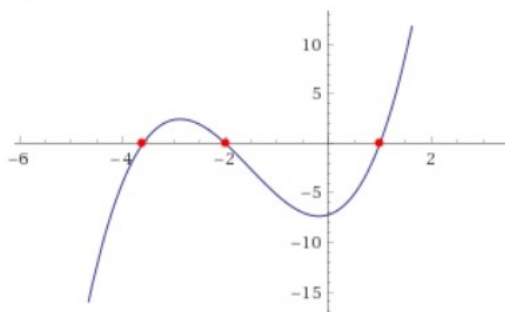


Figura 1. Gráfico da função $f(x) = x^3 + 4.6x^2 + 1.6x - 7.2$.

Observando o gráfico, percebemos que a função possui 3 raízes (-3.6 , -2 e 1). Para facilitar o encontro de cada uma delas, deixamos fixos alguns intervalos no algoritmo, são eles: $[-5, -3]$, $[-3, -1]$ e $[-1, 2]$.

1.2. Algoritmo

```
1  /*
2  * @file questao01.cpp
3  * @brief Universidade Federal do Rio Grande Do Norte (UFRN)
4  * @brief DIM0404 - Calculo Numerico
5  * @brief Tarefa: Encontrar raizes de equacoes
6  * @date 27/02/2018
7  * @author Jhonattan Cabral e Daniel Marx.
8  */
9
10 #include <iostream>
11 #include <math.h>
12 #include <stdlib.h>
13
14 using namespace std;
15
16 double funcao01(double x);
17 double (*func01) (double) = funcao01; //Ponteiro para a funcao01
18
19
20 bool funcaoBissecao(double a, double b, double (*func)(double),
21                     double *p);
22
23 int main()
24 {
25     double p = 0;
26     std::cout << ">>Consultando o grafico percebemos que a
27     funcao possui 3 raizes" << std::endl;
28     //>>Primeira raiz
29     bool r1 = funcaoBissecao(-1,2, func01, &p);
30     if(r1)
31         std::cout << "Para o intervalo de -1 ate 2, temos a raiz
32         (aproximada): " <<
33         p << std::endl;
34     else
35         std::cout << "O metodo falhou" << std::endl;
36
37     p = 0;
38     //>>Segunda raiz
39     bool r2 = funcaoBissecao(-3, -1, func01, &p);
40     if(r2)
41         std::cout << "Para o intervalo de -3 ate -1, temos a
42         raiz(aproximada): " <<
43         p << std::endl;
44     else
45         std::cout << "O metodo falhou" << std::endl;
46
47     p = 0;
48     //>>Terceira raiz
49     bool r3 = funcaoBissecao(-5, -3, func01, &p);
```

```

44     if(r3)
45         std::cout << "Para o intervalo de -5 ate -3, temos a
           raiz(aproximada): " <<
46 p << std::endl;
47     else
48         std::cout << "O metodo falhou" << std::endl;
49
50     return 0;
51 }
52
53 /*
54  *@brief Funcao da questao 01
55  *@param x - Valor de entrada para a funcao
56  *@return Resultado da funcao
57  */
58 double funcao01(double x)
59 {
60     return pow(x,3) + 4.6*pow(x,2) + 1.6*x - 7.2;
61 }
62
63 /*
64  * @brief Metodo iterativo da bissecao
65  * @param a - Inicio do intervalo
66  * @param b - Fim do intervalo
67  * @param *p - ponteiro para a variavel que ira receber o valor
           da raiz
68  * @param *func - Pontoeiro pra funcao na qual se deseja
69  * encontrar as raizes
70  * @return true - Se o metodo conseguir encontrar a raiz(
           aproximada)
71  * @return false - Caso o metodo falhe e nao consiga encontrar a
           raiz
72  */
73 bool funcaoBissecao(double a, double b, double (*func)(double),
           double *p)
74 {
75     double fa, fp;
76
77     for(int i = 0; i < 1000; i++)
78     {
79         fa = func(a);
80         *p = (a + b)/2;
81         fp = func(*p);
82
83         //Tolerancia de 0.000001
84         if(fp == 0 or ((b - a)/2) < 0.000001)
85         {
86             return true;
87         }

```

```

88     else
89     {
90         //Verifica se possui o mesmo sinal
91         if(fa*fp > 0)
92             a = *p;
93         else
94             b = *p;
95     }
96 }
97 return true;
98 }

```

Com o algoritmo, podemos perceber que a função que contém o método da Bisseção possui uma clara semelhança com o método de busca binária.

1.3. Resultados

Para a obtenção dos resultados, utilizamos uma tolerância fixa de 0.000001(mostrado no algoritmo). E como falado antes, observando o gráfico da função, inserimos no algoritmo intervalos tendenciosos, assim facilitando o encontro das raízes.

```

jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos
Arquivo Editar Ver Pesquisar Terminal Ajuda
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos $ ./questao01
>>Consultando o grafico percebemos que a funcao possui 3 raizes
Para o intervalo de -1 ate 2, temos a raiz(aproximada): 1
Para o intervalo de -3 até -1, temos a raiz(aproximada): -2
Para o intervalo de -5 ate -3, temos a raiz(aproximada): -3.6
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos $

```

Figura 2. Resultados obtidos.

2. Descreva o método de Newton-Raphson e utilize-o para encontrar as raízes de $x^3 - 1.7x^2 - 12.78x - 10.08$.

O método de Newton é um dos métodos numéricos mais eficientes e conhecidos para a solução de um problema de determinação de raiz e pode ser introduzido de várias formas. [Burden and Faires 2008]. Uma das formas é escolhendo uma aproximação inicial para a função. Depois, calculasse a Derivada da função nesse ponto e a interseção dela com o eixo das abcissas, a fim de encontrar uma melhor aproximação para a raiz. Repetindo-se o processo, cria-se um método iterativo para encontrarmos a raiz da função.

2.1. Gráfico

Ainda usando a estratégia que foi aplicada para resolver a questão 1, decidimos estudar o gráfico a fim de estabelecer pontos que facilitem o encontro das raízes.

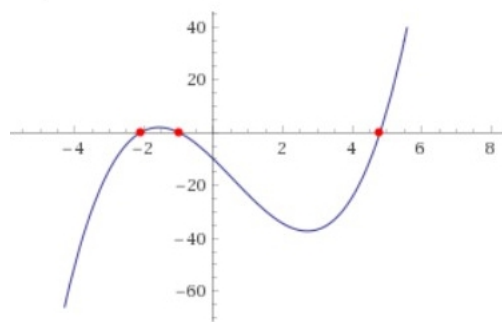


Figura 3. Gráfico da função $f(x) = x^3 - 1.7x^2 - 12.78x - 10.08$.

Na análise do gráfico concluímos que a função contém 3 raízes (-2.1 , -1 e 4.8). Analogamente a questão 1, deixamos fixos 3 pontos, são eles: -4 , 2 e 9 .

2.2. Derivada da função

Como já falado antes, o Método de Newton-Raphson exige o cálculo da derivada da função a qual se deseja encontrar as raízes. Calculando a derivada simples de $f(x) = x^3 - 1.7x^2 - 12.78x - 10.08$, podemos obter $f'(x) = 3x^2 - 3.4x - 12.78$.

2.3. Algoritmo

```

1  /*
2   * @file questao02.cpp
3   * @brief Universidade Federal do Rio Grande Do Norte (UFRN)
4   * @brief DIM0404 - Calculo Numerico
5   * @brief Tarefa: Encontrar raizes de equacoes
6   * @date 27/02/2018
7   * @author Jhonattan Cabral e Daniel Marx.
8   */
9
10 #include <iostream>
11 #include <math.h>
12 #include <cmath>
13 #include <stdlib.h>
14
15 double funcao02(double x);
16 double func02Derivada(double x);
17 double (*func02) (double) = funcao02; //Ponteiro para a funcao02
18
19 double (*funcDerivate) (double) = func02Derivada; //Ponteiro para
    a derivada.
20
21 bool funcaoNewton(double p0, double (*func) (double), double (*
    funcDerivate) (double), double *p);
22
23 int main()
24 {
25     double p = 0;
26     std::cout << ">>Consultando o grafico percebemos que a
    funcao possui 3 raizes" << std::endl;

```

```

25
26 //>>Primeira raiz
27 bool r1 = funcaoNewton(-4, func02, funcDerivate, &p);
28 if(r1)
29     std::cout << "Para o ponto inicial -4, temos a raiz(
        aproximada): " << p << std::endl;
30 else
31     std::cout << "O metodo falhou" << std::endl;
32
33 p = 0;
34 //>>Segunda raiz
35 bool r2 = funcaoNewton(2, func02, funcDerivate, &p);
36 if(r2)
37     std::cout << "Para o ponto inicial 2, temos a raiz(
        aproximada): " << p << std::endl;
38 else
39     std::cout << "O metodo falhou" << std::endl;
40
41 p = 0;
42 //>>Terceira raiz
43 bool r3 = funcaoNewton(9, func02, funcDerivate, &p);
44 if(r3)
45     std::cout << "Para o ponto inicial 9, temos a raiz(
        aproximada): " << p << std::endl;
46 else
47     std::cout << "O metodo falhou" << std::endl;
48
49 return 0;
50 }
51
52 /*
53  *@brief Funcao da questao 02
54  *@param x - Valor de entrada para a funcao
55  *@return Resultado da funcao
56  */
57 double funcao02(double x)
58 {
59     return pow(x,3) - 1.7 * pow(x,2) - 12.78 * x - 10.08;
60 }
61
62 /*
63  *@brief Derivada da funcao da questao 02
64  *@param x - Valor de entrada da funcao
65  *@return Resultado da funcao
66  */
67 double func02Derivada(double x)
68 {
69     return 3*pow(x,2) - 3.4*x - 12.78;
70 }

```

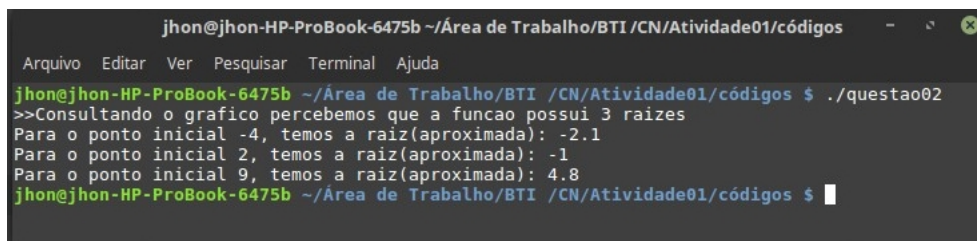
```

71
72 /*
73  * @brief Metodo iterativo de Newton
74  * @param p0 - Ponto de inicio para a estimativa
75  * @param *p - Ponteiro para uma variavel que ira receber a raiz
76   (se houver)
77  * @param *func - Ponteiro pra funcao na qual se deseja
78  * encontrar as raizes.
79  * @param *funcDerivate - Ponteiro pra derivada da funcao da
80   questao 02
81  * @return true - Caso o metodo seja eficiente e retorne a raiz(
82   aproximada)
83  * @return false - Caso o metodo falhe e nao consiga encontrar a
84   raiz.
85  */
86 bool funcaoNewton(double p0, double (*func)(double), double (*
87   funcDerivate)(double), double *p)
88 {
89     for(int i = 0; i < 1000; i++)
90     {
91         *p = p0 - func(*p)/funcDerivate(*p);
92         //Tolerancia fixa de 0.000001
93         if(std::abs(*p - p0) < 0.000001)
94         {
95             return true;
96         }
97         p0 = *p;
98     }
99     return false;
100 }

```

2.4. Resultados

Utilizamos uma tolerância fixa igual a questão anterior, e também como já mencionado antes, fizemos o uso de pontos tendenciosos a fim de achar todas as raízes.



```

jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos
Arquivo Editar Ver Pesquisar Terminal Ajuda
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos $ ./questao02
>>Consultando o grafico percebemos que a funcao possui 3 raizes
Para o ponto inicial -4, temos a raiz(aproximada): -2.1
Para o ponto inicial 2, temos a raiz(aproximada): -1
Para o ponto inicial 9, temos a raiz(aproximada): 4.8
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos $

```

Figura 4. Resultados obtidos.

3. (Burden, pag.62) Um objeto em queda vertical no ar está sujeito á resistência, bem como à força da gravidade. Suponha que um objeto com massa m seja solto a uma altura s_0 e que a altura do objeto após t segundos seja

$$s(t) = s_0 - \frac{mg}{k}t + \frac{m^2g}{k^2}(1 - e^{-kt/m})$$

Onde $g = 32.17\text{pes}/s^2$ e k representa o coeficiente de resistência do ar em lbs/pe . Suponha que $s_0 = 300\text{pes}$, $m = 0.25\text{lb}$ e $k = 0.1\text{lbs}/pe$. Determine, com precisão de $0.00001s$, o tempo decorrido até que o objeto alcance o solo.

Para solucionar este problema decidimos utilizar o método da Bissecção. Dessa vez optamos por uma situação mais real e decidimos não estudar o gráfico da função. Após substituir os valores dados pelo questão, teremos a seguinte função:

$$s(t) = 300 - \frac{0.25 * 32.17}{0.1}t + \frac{0.25^2 * 32.17}{0.1^2}(1 - e^{-0.1t/0.25})$$

3.1. Algoritmo

```
1  /*
2  * @file questao03.cpp
3  * @brief Universidade Federal do Rio Grande Do Norte (UFRN)
4  * @brief DIM0404 - Calculo Numerico
5  * @brief Tarefa: Encontrar raizes de equacoes
6  * @date 27/02/2018
7  * @author Jhonattan Cabral e Daniel Marx.
8  */
9
10 #include <iostream>
11 #include <math.h>
12 #include <cmath>
13 #include <stdlib.h>
14
15 double funcao03(double t);
16 double (*func03) (double) = funcao03; //Ponteiro para a funcao03
17
18 double funcaoBissecao(double t1, double t2, double (*func) (
19     double));
20
21 int main(void)
22 {
23     std::cout << "Tempo decorrido ate certo objeto alcancar o
24         solo, seguindo a formula da questao" << std::endl;
25
26     //>>Primeira raiz
27     double r1 = funcaoBissecao(0, 1000, func03);
```



```

25     std::cout << "Tempo aproximado para chegada ao solo, foi de:
        " << r1 << std::endl;
26     return 0;
27 }
28
29 /*
30  * @brief Funcao da questao 03
31  * @param x - Valor de entrada para o tempo de queda
32  * @return Resultado da funcao
33  */
34 double funcao03(double t)
35 {
36     double g = 32.17;
37     double s0 = 300;
38     double m = 0.25;
39     double k = 0.1;
40
41     return s0 - ((m*g)/k)*t + ((pow(m,2)*g)/pow(k,2))*(1 -
        pow(M_E, ((-1*k)*t)/m)); //M_E - constante do C++ para
        numero de Euler
42 }
43
44 /*
45  * @brief Metodo iterativo da bissecao
46  * @param t1 - Inicio do intervalo
47  * @param t2 - Fim do intervalo
48  * @param *func - Ponteiro pra funcao na qual se deseja
49  * encontrar as raizes
50  * @return p - Valor aproximado do tempo decorrido ate o objeto
        chegar ao solo
51  */
52 double funcaoBissecao(double t1, double t2, double (*func)(
        double))
53 {
54     double fa, fp, p;
55
56     for(int i = 0; i < 10000; i++)
57     {
58         fa = func(t1);
59         p = (t1 + t2)/2;
60         fp = func(p);
61
62         if(fp == 0 or ((t2 - t1)/2) < 0.00001)
63         {
64             return p;
65         }
66         else
67         {
68             //Verifica se possui o mesmo sinal

```

```

69         if(fa*fp > 0)
70             t1 = p;
71         else
72             t2 = p;
73     }
74 }
75 return p;
76 }

```

No código fizemos algumas poucas alterações se compararmos com o código da questão 1. Primeiro atualizamos a função, alteramos a tolerância fixa para 0.000001, que é a tolerância exigida no problema, aumentamos a quantidade de iterações para 10000.

3.2. Resultados

Com um cenário mais realista, colocamos um intervalo maior de $[1, 1000]$ para garantir o encontro da raiz, assim conseguimos obter 6.00373 como resposta.

```

jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI/CN/Atividade01/códigos
Arquivo Editar Ver Pesquisar Terminal Ajuda
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI/CN/Atividade01/códigos $ ./qu
estao03
Tempo decorrido até certo objeto alcançar o solo, seguindo a fórmula da questão
Tempo aproximado para chegada ao solo, foi de: 6.00373
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI/CN/Atividade01/códigos $

```

Figura 5. Resultados obtidos.

4. (Burden, pag. 95) Duas escadas se cruzam em um beco de largura L . Cada escada tem uma extremidade apoiada na base de uma parede e a outra extremidade apoia em algum ponto na parede oposta. As escadas se cruzam a uma altura A acima do pavimento. Calcule L , sendo $x_1 = 20m$ e $x_2 = 30m$ os respectivos comprimentos das escadas e $A = 8m$.

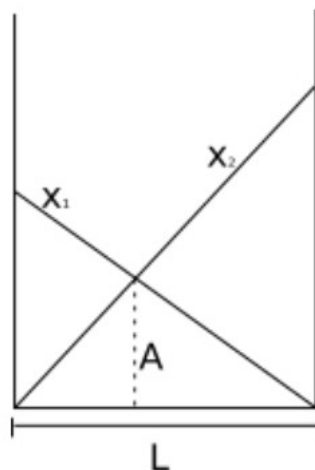


Figura 6. Relações entre as escadas.

Para a execução do cálculo e a obtenção da equação, usamos técnicas como semelhança de triângulos e o teorema de Tales.

4.1. Cálculo

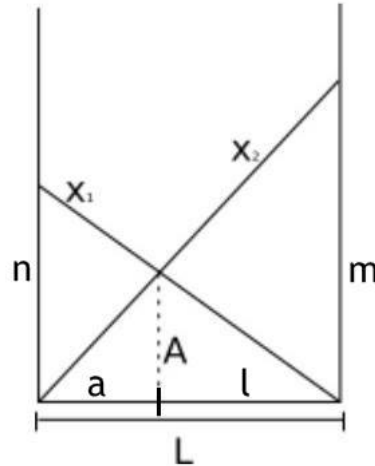


Figura 7. Relações entre as escadas - Análise.

Analisando detalhadamente a figura 7, podemos chegar as seguintes relações:

$$x_1^2 = L^2 + m^2 \Rightarrow m = \sqrt{x_1^2 - L^2}$$

$$x_2^2 = L^2 + n^2 \Rightarrow n = \sqrt{x_2^2 - L^2}$$

Por semelhança de triângulos temos:

$$\frac{m}{L} = \frac{A}{a}$$

$$\frac{n}{L} = \frac{A}{l}$$

Temos ainda que: $L = a + l$ (I).

Faremos então: $a = \frac{AL}{m}$ e $l = \frac{AL}{n}$. Substituindo em (I):

$$L = \frac{AL}{m} + \frac{AL}{n} \Rightarrow L * \frac{1}{L} = \frac{AL}{m} * \frac{1}{L} + \frac{AL}{n} * \frac{1}{L}$$

$$\Rightarrow 1 = \frac{A}{m} + \frac{A}{n} \Rightarrow 1 = \frac{Am + An}{m * n} \Rightarrow 1 = \frac{A(m + n)}{m * n}$$

$$\Rightarrow \frac{1}{A(m + n)} = \frac{1}{m * n} \Rightarrow \frac{1}{A} = \frac{m + n}{m * n}$$

$$\Rightarrow \frac{1}{A} = \frac{m}{m * n} + \frac{n}{m * n} \Rightarrow \frac{1}{A} = \frac{1}{n} + \frac{1}{m}$$

$$\frac{1}{A} = \frac{1}{\sqrt{x_2^2 - L^2}} + \frac{1}{\sqrt{x_1^2 - L^2}}$$

Dadas as duas primeiras relações encontradas, então: $x_1 = 20cm$, $x_2 = 30cm$, $A = 8cm$.

4.2. Algoritmo

```

1  /*
2  * @file questao04.cpp
3  * @brief Universidade Federal do Rio Grande Do Norte (UFRN)
4  * @brief DIM0404 - Calculo Numerico
5  * @brief Tarefa: Encontrar raizes de equacoes
6  * @date 27/02/2018
7  * @author Jhonattan Cabral e Daniel Marx.
8  */
9
10 #include <iostream>
11 #include <math.h>
12 #include <cmath>
13 #include <stdlib.h>
14
15 double funcao04(double L);
16 double(*func04) (double) = funcao04; //Ponteiro para a funcao03.
17 double funcaoBissecao(double t1, double t2, double(*func)(double
18     ));
19
20 int main(void)
21 {
22     std::cout << "Calculo da largura do beco, por meio do
23         metodo da bissecao" << std::endl;
24
25     double r1 = funcaoBissecao(0, 1000, func04);
26     std::cout << "Largura aproximada do beco e: " << r1 <<
27         std::endl;
28     return 0;
29 }
30
31 /*
32 * @brief Funcao da questao 04; Desrita por: 1/A = 1/(x1^2 - L^2)
33     + 1/(x2^2 - L^2)
34 * @param L -> Valor que se espera ser igual a largura do beco
35 * @return Resultado da funcao
36 */
37 double funcao04(double L)
38 {

```

```

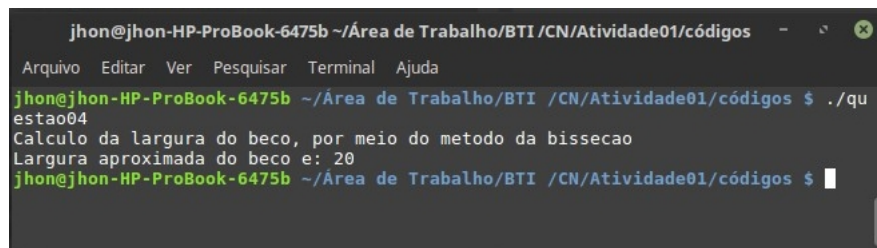
35     double x1 = 20;
36     double x2 = 30;
37     double A = 8;
38
39     return A/(sqrt(pow(x1, 2) - pow(L, 2))) + A/(sqrt(pow(x2
40         , 2) - pow(L, 2)));
41 }
42
43 /*
44  * @brief Metodo iterativo da bissecao; Retorna a largura de um
45     beco no problema da questao 04
46  * @param l1 - Inicio do intervalo
47  * @param l2 - Fim do intervalo
48  * @param *func - Ponteiro pra funcao na qual se deseja
49     encontrar as raizes
50  * @return p - Valor aproximado do tempo decorrido ate o objeto
51     chegar ao solo
52  */
53 double funcaoBissecao(double l1, double l2, double(*func)(double
54     ))
55 {
56     double fa, fp, L;
57
58     for (int i = 0; i < 10000; i++)
59     {
60         fa = func(l1);
61         L = (l1 + l2) / 2; //Valor da largura do beco
62         fp = func(L);
63
64         if (fp == 1 or ((l2 - l1) / 2) < 0.000001) //
65             Diferentemente das outras questoes, o valor
66             da funcao devera ser igual a 1 para encontrar
67             o tamanho do beco
68         {
69             return L;
70         }
71         else
72         {
73             //Verifica se possui o mesmo sinal
74             if (fa*fp > 0)
75                 l1 = L;
76             else
77                 l2 = L;
78         }
79     }
80     return -1; //Algum erro ocorreu
81 }

```

Novamente utilizamos o método da Bisseção para resolver o problema, uma vez encontrado a equação e os valores, moldamos os novos parâmetros no código. Diferentemente das questões anteriores o valor resultante deverá ser próximo de 1 para que assim possamos encontrar a largura aproximada do beco.

4.3. Resultados

Para a obtenção do resultado, novamente simulamos um ambiente mais real para o problema. Colocamos um intervalo maior de $[1, 1000]$ e uma tolerância de 0.000001 . Assim conseguimos obter como resultado uma largura aproximada de 20cm.



```
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos
Arquivo Editar Ver Pesquisar Terminal Ajuda
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos $ ./qu
estao04
Calculo da largura do beco, por meio do metodo da bissecao
Largura aproximada do beco e: 20
jhon@jhon-HP-ProBook-6475b ~/Área de Trabalho/BTI /CN/Atividade01/códigos $
```

Figura 8. Resultados obtidos.

Referências

- [Burden and Faires 2008] Burden, R. L. and Faires, J. D. (2008). *Análise Numérica*. Addison-Wesley, 8th edition.