Universidade Federal do Amazonas

Instituto de Ciências Exatas

Departamento de Física

Algoritmos de implementação aos Métodos da Bissecção e Falsa Posição na linguagem Python 3.7

Relatório para a disciplina de Cálculo Numérico

por

Aluno: Micael Davi Lima de Oliveira

Orientador: Prof. Dr. José Francisco de Magalhães Neto

Manaus, Setembro de 2019

Micael Davi Lima de Oliveira

Algoritmos de implementação aos Métodos da Bissecção e Falsa Posição na linguagem Python 3.7

Relatório como requisito de nota parcial para a disciplina de Cálculo Numérico apresentado ao Curso de Bacharelado em Física na Universidade Federal do Amazonas.

Orientador: Prof. Dr. José Francisco de Magalhães Neto

Manaus, AM

2019

Agradecimentos

Agradeço muito a Deus, por me dar forças e esperança em viver. Ainda que eu não mereça tamanho amor, me fortalece nos dias de tristeza.

Aos meus pais, pelo carinho e apoio.

Ao professor José Reginaldo, cujo conhecimento transmitido foi imprescindível para a realização deste trabalho.

E por fim, aos autores das bibliotecas "Matplotlib"e "Qt5 for Python", as quais foram ferramentas indispensáveis para a construção deste software.

Resumo

Neste trabalho será feito a implementação de 2 métodos de Análise Numérica: O Método da Bissecção e o Método da Falsa Posição. A linguagem de programação adotada foi o Python na versão 3.7.3 de arquitetura 64 bits rodando numa distribuição Linux Ubuntu 19.04. A literatura já existente será crucial para a implementação. O Teorema de Bolzano é algo fundamental que rege o correto funcionamento de ambos os métodos, tal como, a ausência de descontinuidades na função a ser analisada. O software a ser construído irá calcular o valor de uma única raiz real de funções polinomiais. Por meio da biblioteca "*Matplotlib*" também haverá a possibilidade de o usuário plotar o gráfico e ter uma visualização geométrica das raízes. O usuário poderá escolher o número de iterações máximas, e após o cálculo computacional, o software irá gerar automaticamente uma tabela contendo todos os valores assumidos pelas seguintes variáveis: a_i, b_i, m_i, $f(a_i)$, $f(b_i)$, $f(m_i)$. O usuário também irá escolher o intervalo de busca das raízes a ser adotado pelo algoritmo. Destaca-se que não foi aplicado o Teorema de Lagrange para uma descoberta automática do intervalo onde encontram-se potenciais raízes reais. O usuário também escolherá a precisão do resultado calculado, e além disso, o software também irá mostrar a condição de parada: (1) Não existem raízes reais neste intervalo; (2) Encontrou-se uma solução real aproximada; (3) Encontrou-se uma solução real exata; (4) Atingiu-se o máximo de iterações. E por fim, o algoritmo também possibilita que o usuário veja uma comparação entre os métodos numéricos, permitindo visualizar qual método convergiu mais rapidamente.

Palavras-chave: Bissecção, Falsa Posição, Teorema de Bolzano, Python.

Lista de Figuras

Lista de Tabelas

Sumário

Introdução

Seja em ciência ou tecnologia, é recorrente a necessidade de se encontrar um número ϵ de tal maneira que uma função $f(x)$ seja zero, e que portanto, $f(\epsilon) = 0$. Embora cálculos computacionais dificilmente são capazes de atingir precisamente a raiz exata, mas apenas um valor aproximado. Mesmo com todo o progresso feito na matemática, sabe-se apenas que equações algébricas de 1º e 2º grau, e certas classes de 3º e 4º grau e algumas equações transcendentes podem ter suas raízes calculadas via métodos analíticos. E desta forma, para polinômios com grau acima de 4, apenas existem métodos que encontram de forma aproximada a raiz. [2]

Ainda que não seja possível calcular o valor exato da raiz, pode-se descobrir em conformidade com a exatidão desejada. Embora seja necessário que sejam seguidas algumas etapas: (a) Isola-se a raiz, de tal maneira a encontrar um intervalo $[a, b]$, o menor possível, que contenha apenas uma única raiz da equação $f(x) = 0$. (b) Diminuir o erro da raiz aproximada, e portanto, calcular uma raiz onde $x_0 \leq \epsilon$. [2]

1.1 Teorema de Bolzano

Bernard Bolzano em 1817 demonstrou este teorema usando técnicas consideradas rigorosas para a época, mas que no mundo moderno, são técnicas com um rigor não tão perfeito. Na verdade, este teorema é um corolário do Teorema do Valor Intermediário, pertencente ao ramo do Cálculo Diferencial na matemática. Se uma função contínua $f(x)$ assume sinais opostos nos pontos extremos do intervalo $[a, b]$, logo, $f(a) \cdot f(b) \leq 0$, e portanto, o intervalo conterá, no mínimo, uma raiz real da equação $f(x) = 0$.

$$f(a) \cdot f(b) < 0 \rightarrow \overline{x} \in [a, b] \rightarrow f(\overline{x}) = 0 \tag{1.1}$$

Desta forma, o Teorema de Bolzano é a forma pela qual sabe-se a existência ou não de uma raiz real num determinado intervalo numérico. Nos algoritmos a serem cons-

truídos, esta é uma condição essencial para que o computador busque a raiz via métodos recursivos, pois caso o teorema não seja satisfeito, o algoritmo entraria num looping, pois busca num intervalo onde inexiste qualquer raiz real para a função polinomial em análise. [2]

1.2 Teorema Fundamental da Álgebra

O problema de perceber o tipo de solução para uma equação polinomial, surgiu deste o início da História da Matemática. Séculos antes da descoberta dos números complexos, François Vieta(1540-1603), mostrou que que equação polinomiais de grau n possui n raízes. O Teorema Fundamental da Álgebra afirma que qualquer polinômio $p(z)$ de grau n possui n soluções, mas não necessariamente distintas. Isto ocorre porque o corpo dos números complexos é algebricamente fechado. Uma equação algébrica de grau n tem exatamente n raízes reais ou complexas. Embora, cada raiz é contada de acordo com sua multiplicidade. [5]

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_0 = 0 \tag{1.2}$$

$$p(z) \text{ com grau } n, \text{ tem exatamente } n \text{ raízes reais ou complexas.} \tag{1.3}$$

1.2.1 Multiplicidade das raízes

Num polinômio com raízes múltiplas, ao fazer uma observação superficial da imagem, somos levados a concluir erroneamente que o gráfico está incorreto, ou mesmo, que o Teorema Fundamental da Álgebra é falso, visto que foi encontrado apenas 6 pontos que interceptam o eixo das abscissas, sendo que é um polinômio de grau 8. Contudo, ao ser feita uma observação mais cuidadosa, ocorre a repetição de raízes, ou seja, esses pontos possuem multiplicidade igual a dois. E desta forma, encontram-se finalmente as oito raízes do polinômio. Um ponto $\overline{x} \in [a, b]$ é uma raiz de multiplicidade m da equação $f(x) = 0$ se $f(x) = (x - \overline{x})^m \cdot g(x)$, com $g(x) \neq 0 \in [a, b]$. [2]

$$p(r) = 0 \longleftrightarrow p(x) = (x - r)^k \cdot g(x) \tag{1.4}$$

1.2.2 Teorema dos conjugados em raízes complexas

Se os coeficientes de uma equação algébrica polinomial são todos reais, então as raízes complexas desta equação são complexos conjugados em pares. Desta forma, se $\epsilon_1 = \alpha + \beta i$ é uma raiz de multiplicidade m, então um número $\epsilon_2 = \alpha - \beta i$ também será raiz da equação com a mesma multiplicidade m. [2]

Quando $p(x)$ é de grau ímpar, haverá no mínimo 1 raiz real (1.5)

1.3 Método da Bissecção

O método da Bissecção é uma técnica numérica para encontrar a solução de equações polinomiais que sejam contínuas. A parte central deste método é a validade do Teorema de Bolzano. É preciso que se tenha uma função $f(x)$ contínua num dado intervalo $[a, b]$ e que $f(a) \cdot f(b) < 0$. Ao dividir o intervalo $[a, b]$ pela metade, obtém um x_0, surgindo então, dois subintervalos, $[a, x_0]$ e $[x_0, b]$. Se por acaso, $f(x_0) = 0$, então, $\xi_0 = x_0$. Caso contrário, a raiz estará no subintervalo onde a função adquire sinais opostos nos extremos, isto é, se $f(a) \cdot f(x_0) < 0$, logo, $\xi_0 \in (a, x_0)$. Por outro lado, se $f(a) \cdot f(x_0) > 0$, então, $\xi_0 \in (x_0, b)$. Este novo intervalo $[a_1, b_1]$ que contém x_0 é dividido novamente por 2, obtendo-se um x_1. As iterações se repetem até que finalmente se obtenha uma aproximação para a raiz, conforme a precisão ϵ adotada. [1,6]

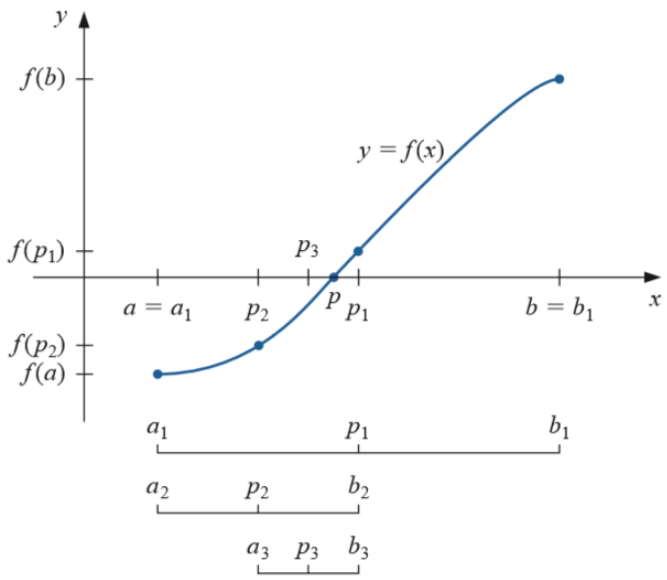

Figura 1: *Interpretação geométrica do Método da Bissecção.* [3]

1.3.1 Convergência do Método da Bissecção

Em algum momento entre as iterações, será encontrado uma raiz aproximada ou exata x_1, tal que:

$$f(a_n) \cdot f(b_n) < 0 \text{ e } n = 1, 2, 3, \ldots \tag{1.6}$$

A cada iteração o intervalo $[a, b]$ é dividido por 2, na n-ésima iteração o comprimento do intervalo poderá ser expresso por:

$$b_n - a_n = \frac{b-a}{2^n} \tag{1.7}$$

$$|x_n - x_{n-1}| = \frac{b-a}{2^{n+1}} \tag{1.8}$$

Desde que,

$$|x_n - x_{n-1}| \leq \epsilon \tag{1.9}$$

E portanto,

$$\left| \frac{b-a}{2^{n+1}} \leq \epsilon \right| \tag{1.10}$$

$$n \geq \frac{\ln[(b-a)/\epsilon]}{\ln 2} - 1 \tag{1.11}$$

Desta forma, num dado intervalo $[a, b]$ são necessárias, no mínimo, n interações para calcular a raiz x_1 com uma precisão ϵ. Quando $n \to \infty$, e considerando a inexistência de qualquer descontinuidade, haverá uma raiz x_1 de tal maneira que $f(x_1) = 0$. [1]

1.4 Método da Falsa Posição ou Regula Falsi

Uma função $f(x)$ contínua num intervalo $[x_0, x_1]$, onde $f(x_0) \cdot f(x_1) < 0$. Pelo Teorema de Bolzano, sabe-se que existe uma raiz nesse intervalo, e após sucessivas interações,

Considerando-se 2 triângulos semelhantes, tem-se,

$$\frac{0 - f(x_L)}{x_r - x_L} = \frac{0 - f(x_U)}{x_r - x_U} \tag{1.12}$$

A partir da conclusão, pode-se obter,

$$(x_r - x_L) \cdot f(x_U) = (x_r = x_U) \cdot f(x_L) \tag{1.13}$$

$$x_U \cdot f(x_L) - x_L \cdot f(x_U) = x_r \cdot [f(x_L) - f(x_U)] \tag{1.14}$$

$$x_r = x_L - \frac{f(x_L)}{\dfrac{f(x_U) - f(x_L)}{x_U = x_L}} \tag{1.15}$$

E desta forma, pode-se demonstrar que cada x_{n+1} pode ser expresso por,

$$x_{n+1} = x_n - \frac{f(x_n) \cdot (x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \tag{1.16}$$

Sendo assim, o Método da Falsa Posição busca reduzir o valor $f(x_{n-1})$ a um fator de $f(x_n)/[f(x_n) + f(x_{n+1})]$, que é um modo de evitar a retenção da raiz a ser encontrada. [6]

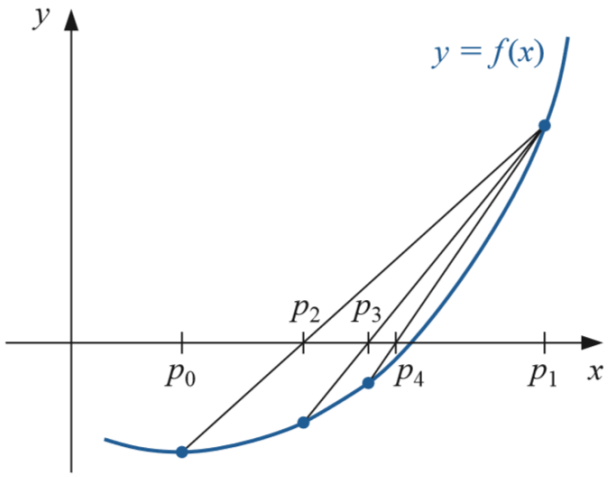

Figura 2: *Interpretação geométrica do Método da Falsa Posição.* [3]

Estrutura computacional

Utilizou-se uma linguagem de alto nível, Python na versão 3.7.3 numa arquitetura computacional de x64 bits rodando em Linux Ubuntu 19.04 x64. Os resultados aparentemente não diferiram quando o software foi testado no sistema operacional Windows 10 x64 bits. O ambiente de desenvolvimento(IDE) adotado foi o Visual Studio Code na versão 1.38.1. A interface gráfica foi construída com o auxílio da biblioteca Qt5 for Python. A plotagem das funções polinomiais foi apenas possível, devido ao uso da biblioteca Matplotlib na versão 2.0.2. A biblioteca Numpy foi importante para a construção de arrays que armazenassem os coeficientes do polinômio. Embora, muitas desses parâmetros que constituem a estrutura computacional, não mostraram ser significativos nos resultados obtidos no projeto. Percebeu-se que houve apenas uma distorção nas fontes tipográficas na transição do Ubuntu 19.04 para o Windows 10.

Figura 3: *Logo do sistema operacional Ubuntu 19.04.* [12]

Figura 4: *Logo do ambiente de desenvolvimento Visual Studio Code.* [10]

Figura 5: *Logo da linguagem de programação Python.* [8]

Figura 6: *Logo da biblioteca de interface gráfica Qt5.* [11]

Figura 7: *Logo da biblioteca Matplotlib.* [9]

Para facilitar o entendimento da estrutura computacional adotada, será apresentada uma tabela. Será possível visualizar a relação entre um certo parâmetro e seu respectivo impacto no software já compilado. É importante destacar que após a compilação não foi possível construir um arquivo binário, seja no formato *.exe* ou algum formato para que houvesse um programa totalmente responsivo, que funcionasse da mesma forma, independente do sistema operacional. Outro ponto importante, é que os algoritmos da Bissecção e Falsa Posição foram construídos sob um único programa *.py*. E por isso, um único software é capaz de lidar com ambos os métodos numéricos, cabendo ao usuário a escolha a ser adotada pelo algoritmo.

Tabela 1: *Comparativo da influência dos parâmetros computacionais no software.*

Parâmetro	**Característica**	**Influência computacional**
Sistema Operacional	Ubuntu 19.04	Baixa
Arquitetura	64 bits	Mediana
Linguagem de Programação	Python 3.7.3	Alta
Interface Gráfica	Qt5	Baixa
Biblioteca para Arrays	Numpy	Desconhecido
Biblioteca para Plotagem	Matplotlib 2.0.2	Desconhecido

Algoritmo construído

Listing 3.1: *Código-fonte que compõe todo o software programado.*

```
from PyQt5 import QtWidgets, uic, QtGui, QtCore
from PyQt5.QtWidgets import QTableWidget, QTableWidgetItem
from numpy import*

from PyQt5.QtWidgets import QMessageBox

import numpy as np
from math import*
import matplotlib.pyplot as plt

import sys
import os

app = QtWidgets.QApplication([])
dlg = uic.loadUi("raizes.ui")

dlg.bt_conf1.setIcon(QtGui.QIcon('Images/ok.png'))
dlg.bt_conf2.setIcon(QtGui.QIcon('Images/ok.png'))
dlg.bt_conf3.setIcon(QtGui.QIcon('Images/ok.png'))
dlg.bt_conf4.setIcon(QtGui.QIcon('Images/ok.png'))

coef = 0
aux = 0
cont_term = 0
max_it = 0
epsilon = 0
aprox = 0

metodo = 0    # 0 para bissec   o; 1 para falsa posi   o
```

```
m0 = 0 # valor inicial para a m dia aritm tica ou ponderada dos extremos do interva


# Constru  o da tabela contendo os resultados #

table    = QTableWidget()
tableItem         = QTableWidgetItem()
table.setWindowTitle("Resultados obtidos")
table.resize(650, 450)
table.setRowCount(1)
table.setColumnCount(6)

table.setHorizontalHeaderItem(0, QTableWidgetItem("a_n"))
table.setHorizontalHeaderItem(1, QTableWidgetItem("b_n"))
table.setHorizontalHeaderItem(2, QTableWidgetItem("m_n"))
table.setHorizontalHeaderItem(3, QTableWidgetItem("f(a_n)"))
table.setHorizontalHeaderItem(4, QTableWidgetItem("f(b_n)"))
table.setHorizontalHeaderItem(5, QTableWidgetItem("f(m_n)"))

def f(x):
    funcao = 0
    for i in range(size(coef)):
        funcao = funcao + float(coef[i]) * x ** int(size(coef) - i - 1)
    return funcao

def conf1():
   dlg.bt_conf1.setEnabled(False)
   dlg.txt_gr.setEnabled(False)
   dlg.txt_coef.setEnabled(True)
   dlg.bt_conf2.setEnabled(True)
   dlg.cb_coef.setEnabled(True)

   global coef
   coef = zeros(int(dlg.txt_gr.text()) + 1)

   sup = str.maketrans("0123456789", "                                    ")
   for i in range(int(dlg.txt_gr.text()) + 1):
    dlg.cb_coef.addItem("x" + str(int(dlg.txt_gr.text())-i).translate(sup))
```

```
def conf2():
    global cont_term

    if (dlg.txt_coef.text() != ""):
        coef[dlg.cb_coef.currentIndex()] = dlg.txt_coef.text()
        dlg.cb_coef.setCurrentIndex(dlg.cb_coef.currentIndex()+1)

        if (cont_term == size(coef)-1):
            dlg.cb_coef.setEnabled(False)
            dlg.bt_conf2.setEnabled(False)
            dlg.txt_coef.setEnabled(False)
            dlg.txt_it.setEnabled(True)
            dlg.bt_conf3.setEnabled(True)
        else:
            dlg.txt_coef.setText("")

        cont_term += 1

    print(coef)

def conf3():
    global max_it
    max_it = int(dlg.txt_it.text())

    dlg.bt_conf3.setEnabled(False)
    dlg.txt_it.setEnabled(False)
    dlg.bt_conf4.setEnabled(True)
    dlg.txt_pr.setEnabled(True)

def conf4():
    global epsilon
    epsilon = float(dlg.txt_pr.text())

    global aprox
    aprox = len(dlg.txt_pr.text()) - 1

    dlg.bt_conf4.setEnabled(False)
    dlg.txt_pr.setEnabled(False)
    dlg.bt_mfp.setEnabled(True)
```

```
    dlg.bt_mb.setEnabled(True)

    dlg.lb_eq2.setEnabled(True)
    dlg.lb_n2.setEnabled(True)
    dlg.lb_pr2.setEnabled(True)

    dlg.plot_f.setEnabled(True)
    dlg.actionComparar.setEnabled(True)
    dlg.lb_a.setEnabled(True)
    dlg.lb_b.setEnabled(True)

    aux = "f(x) = "
    sup = str.maketrans("0123456789", "                    ")

    for i in range(size(coef)):
        aux = aux + str(abs(coef[i])) + "x" + str(size(coef) - i - 1).translate(sup)
        if ((i) <= (size(coef) - 2)):
            if (coef[i+1] >= 0):
                aux = aux + " + "
            else:
                aux = aux + " - "

    dlg.lb_eq2.setText(aux)
    dlg.lb_n2.setText("i = " + str(max_it))
    dlg.lb_pr2.setText("  = " + str(epsilon))

dlg.bt_conf1.clicked.connect(conf1)
dlg.bt_conf2.clicked.connect(conf2)
dlg.bt_conf3.clicked.connect(conf3)
dlg.bt_conf4.clicked.connect(conf4)

def graf():
    X = np.linspace(float(dlg.lb_a.text()), float(dlg.lb_b.text()), 200, endpoint=True
    F = f(X)

    plt.grid(True)
    plt.grid(b=True, which='major', color='#666666', linestyle='-', alpha=0.4)
    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999', linestyle='--', alpha=0.20)
```

```
    plt.plot(X,F)
    plt.show()

def raizes():
    dlg.lb_r2.setEnabled(True)
    dlg.lb_r3.setEnabled(True)
    dlg.lb_tot_it2.setEnabled(True)
    dlg.c_parada.setEnabled(True)
    dlg.lb_obs2.setEnabled(True)

    aut = 0

    if (dlg.lb_a.text() == "") or (dlg.lb_b.text() == ""):
        choice = QMessageBox.information(dlg, 'Intervalo de busca', "Por favor, insira

    if ((dlg.lb_a.text() != "") and (dlg.lb_b.text() != "")) or (aut == 1):

        global a_n
        global b_n
        global m_n

        global f_m_n
        global f_a_n
        global f_b_n

        global n

        a_n = float(dlg.lb_a.text())
        b_n = float(dlg.lb_b.text())
        m_n= 0

        f_m_n = 0
        f_a_n = 0
        f_b_n = 0

        global a
        global b
        global m
```

```
global f_a
global f_b
global f_m

a = zeros(1000)
b = zeros(1000)
m = zeros(1000)
f_a = zeros(1000)
f_b = zeros(1000)
f_m = zeros(1000)

n = 1

for n in range(1, max_it+1):

    f_m_n = 0
    f_a_n = 0
    f_b_n = 0

    if (metodo == 0): # Metodo da Bissec  o
        m_n = (a_n + b_n)/2

    if (metodo == 1): # Metodo da Falsa Posi  o

        for i in range(size(coef)):
            f_a_n = f_a_n + float(coef[i]) * (a_n) ** int(size(coef) - i - 1)
            f_b_n = f_b_n + float(coef[i]) * (b_n) ** int(size(coef) - i - 1)

        m_n = (b_n) - ((f_b_n*(b_n - a_n)) / (f_b_n - f_a_n))

    for i in range(size(coef)):
        f_a_n = f_a_n + float(coef[i]) * (a_n) ** int(size(coef) - i - 1)
        f_b_n = f_b_n + float(coef[i]) * (b_n) ** int(size(coef) - i - 1)
        f_m_n = f_m_n + float(coef[i]) * (m_n) ** int(size(coef) - i - 1)

    if (f_a_n)*(f_m_n) < 0:
        b_n = m_n

    if (f_b_n)*(f_m_n) < 0:
```

```
        a_n = m_n

    a[n-1] = a_n
    b[n-1] = b_n
    m[n-1] = m_n
    f_a[n-1] = f_a_n
    f_b[n-1] = f_b_n
    f_m[n-1] = f_m_n

    if (f_m[n-1]) == 0:
        dlg.lb_obs2.setText("Encontrou-se 1 solu   o real exata.")
        dlg.lb_obs2.setStyleSheet("color: green")
        dlg.lb_r2.setText(" x   = " + str(round(m[n-1], aprox)))
        dlg.lb_r3.setText("f( x   ) = " + str(round(f_m[n-1], aprox)))
        dlg.lb_tot_it2.setText("i = " + str(n))
        break

    if (f_a_n)*(f_b_n) > 0:
        dlg.lb_obs2.setText("N o existem existem ra zes reais neste interval
        dlg.lb_obs2.setStyleSheet("color: red")
        dlg.lb_r2.setText(" x   =   " + str(round(m[n-1], aprox)))
        dlg.lb_r3.setText("f( x   )   =   " + str(round(f_m[n-1], aprox)))
        dlg.lb_tot_it2.setText("i = " + str(n))

        dlg.c_parada.setText("Raiz real n o encontrada")

        break

    if abs(m_n) < sqrt(10): # Condi   o de parada por Teste Absoluto de Erro
        if (abs(b_n - a_n) <= epsilon) or (abs(f_m_n) <= epsilon): # Condi
            dlg.lb_obs2.setText("Encontrou-se uma solu   o real aproximada."
            dlg.lb_obs2.setStyleSheet("color: green")
            dlg.lb_r2.setText(" x   =   " + str(round(m[n-1], aprox)))
            dlg.lb_r3.setText("f( x   )   =   " + str(round(f_m[n-1], aprox)))
            dlg.lb_tot_it2.setText("i = " + str(n))

            if abs(b_n - a_n) <= epsilon:
                dlg.c_parada.setText("b_n - a_n <= epsilon")
            else:
```

```
                dlg.c_parada.setText("f_m_n <= epsilon")

                break

        if (abs(m_n) >= sqrt(10)): # Condi  o de parada por Teste Relativo de E
            if (abs(b_n - a_n) / abs(b_n)) <= epsilon : # Condi  o de parada su
                dlg.lb_obs2.setText("Encontrou-se uma solu  o real aproximada.")
                dlg.lb_obs2.setStyleSheet("color: green")
                dlg.lb_r2.setText(" x     " + str(round(m[n-1], aprox)))
                dlg.lb_r3.setText("f( x  )     " + str(round(f_m[n-1], aprox)))
                dlg.lb_tot_it2.setText("i = " + str(n))

                dlg.c_parada.setText("[(b_n - a_n) / (b_n)] <= epsilon")

                break

table.show()
table.setRowCount(n)
k = 0
for k in range(n):
    table.setItem(k,0, QTableWidgetItem(str(round(a[k], aprox))))
    table.setItem(k,1, QTableWidgetItem(str(round(b[k], aprox))))
    table.setItem(k,2, QTableWidgetItem(str(round(m[k], aprox))))
    table.setItem(k,3, QTableWidgetItem(str(round(f_a[k], aprox))))
    table.setItem(k,4, QTableWidgetItem(str(round(f_b[k], aprox))))
    table.setItem(k,5, QTableWidgetItem(str(round(f_m[k], aprox))))

global n0
global m0
global f0

global n1
global m1
global f1

if (metodo == 0):
    n0 = str(n)
    m0 = str(round(m[k], aprox))
    f0 = str(round(f_m[k], aprox))
```

```
        if (metodo == 1):
            n1 = str(n)
            m1 = str(round(m[k], aprox))
            f1 = str(round(f_m[k], aprox))

        if (abs(f_m_n) > epsilon) and ((f_a_n)*(f_b_n)) < 0:
            dlg.lb_obs2.setText("N o foi poss vel atingir a precis o.")
            dlg.lb_obs2.setStyleSheet("color: red")
            dlg.lb_r2.setText(" x      " + str(round(m[n-1], aprox)))
            dlg.lb_r3.setText("f( x  )      " + str(round(f_m[n-1], aprox)))
            dlg.lb_tot_it2.setText("i =" + str(n)) # n o alterar para n-1

            if (n >= max_it):
                dlg.c_parada.setText("Itera   es excedidas")
            else:
                dlg.c_parada.setText("Desconhecido")

def bisseccao():
    global metodo
    metodo = 0

    dlg.groupBox_3.setTitle("M todo da Bissec   o")
    raizes()

def falsa_posicao():
    global metodo
    metodo = 1

    dlg.groupBox_3.setTitle("M todo da Falsa Posi   o")
    raizes()

def novo():
    python = sys.executable
    os.execl(python, python, * sys.argv)

def sair():
    app.exit()
```

```
def comp():
    bisseccao()
    falsa_posicao()

    choice = QMessageBox.information(dlg, 'Comparativo entre m todos', "----------

def sobre():
    choice = QMessageBox.information(dlg, 'Agradecimentos', "Agrade o muito a Deus, p

dlg.bt_mfp.triggered.connect(falsa_posicao)
dlg.bt_mb.triggered.connect(bisseccao)
dlg.plot_f.triggered.connect(graf)
dlg.actionNovo.triggered.connect(novo)
dlg.actionSair.triggered.connect(sair)
dlg.actionComparar.triggered.connect(comp)
dlg.actionSobre.triggered.connect(sobre)

dlg.show()
app.exec()
```

Resultados e Discussões

O algoritmo construído mostrou-se relativamente eficaz em ambos os métodos de análise numérica. Todavia, não apresentou resultados totalmente precisos comparados à literatura. A principal fonte de erro deu-se principalmente pela condição de parada do laço de repetição, isto porque, por algum motivo desconhecido o Método da Bissecção e Falsa Posição comportam-se de forma sutilmente diferentes para que haja a parada do laço. Percebeu-se que determinados exemplos excediam o número de iterações necessárias, ainda que tenha apresentado um resultado coerente, isto é, cuja imagem da função de fato aproximou-se de zero. De uma forma geral, o algoritmo ainda que tenha suas falhas internas, e mesmo tendo sido levado em consideração 3 condições: (a) Parada por erro absoluto; (b) Parada por erro relativo; (c) Parada por $f(m_n) \approx 0$ conforme a precisão. Ainda que tenha sido feito um grande esforço para corrigir tais problemas de saída de dados, até o momento, não descobriu-se sua real causa.

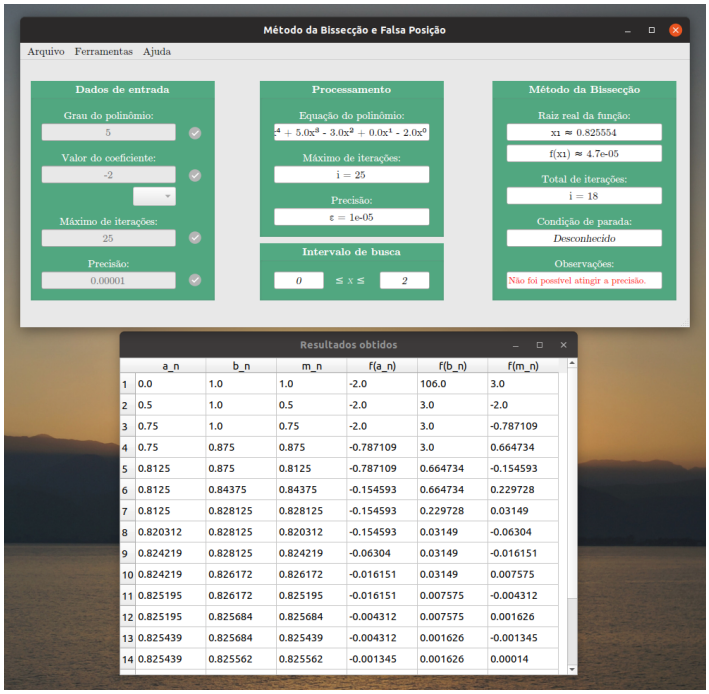

Figura 8: *Interface gráfica do software construído.*

Percebeu-se que grande parte dos casos de teses apresentaram resultados bem próximos do esperado, ainda que tenha havido constantemente um erro interno no algoritmo. Outro ponto importante, é que em boa parte dos exemplos testados, o método da Falsa Posição mostrou-se possuir um grau de convergência maior que o Método da Bissecção. Todavia, isto não foi uma regra, já que houve certos exemplos onde a Bissecção mostrou-se mais eficiente.

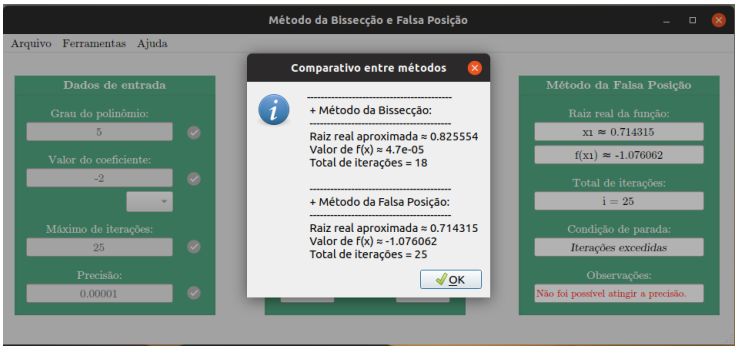

Figura 9: *Janela comparativa de convergência entre os métodos numéricos.*

E desta forma, foi construído uma tabela que pudesse exemplificar a observação de que nem sempre a Falsa Posição convergiu mais rapidamente. Não se sabe a explicação disto, talvez sejam falhas no próprio algoritmo construído. Já que as condições de parada estiveram muito suscetíveis a falhas. Todavia, como recurso adicional, foi utilizado uma calculadora online de Análise Numérica.[13] Percebeu-se então, que a eficiência dos métodos não mostrou uma regularidade visível, constatando-se que mesmo softwares profissionais sofrem com esta mesma oscilação de eficiência entre os métodos.

Tabela 2: *Comparativo da eficiência dos métodos numéricos.*

Polinômio	**Método numérico**	**Iterações**	m_n	$f(x_n)$
$x^2 - 3$	Bissecção	15	$1,73209$	12×10^{-5}
$x^2 - 3$	Falsa Posição	9	$1,73203$	$-9,0 \times 10^{-5}$
$x^3 - 9x + 3$	Bissecção	15	$0,33759$	$2,0 \times 10^{-4}$
$x^3 - 9x + 3$	Falsa Posição	4	$0,33762$	$-6,0 \times 10^{-5}$
$x^4 - 16$	Bissecção	15	$1,99997$	-98×10^{-5}
$x^4 - 16$	Falsa Posição	22	$2,00000$	$-6,0 \times 10^{-5}$

Conclusões

Em geral, percebeu-se que o algoritmo construído mostrou-se eficaz em grande parte dos casos de teste. Embora, possua falhas internas, foi capaz de calcular raízes coerentes com o polinômio inserido. Todavia, as condições de parada apresentam certas inconsistências, acarretando num maior ou menor número de iterações comparado a softwares profissionais, como por exemplo, Casio Numerical Analysis. Adotou-se diversas condições de parada no algoritmo afim de que apresentasse menos falhas, mas ainda sim, os erros foram inevitáveis. De qualquer forma, mesmo com os erros de saída, os resultados apresentam uma grande coerência entre si. Embora, em certos casos, o número de iterações ou a raiz encontrada não tenha coincidido precisamente com o suposto valor real de um software profissional, o algoritmo mostra-se de uma forma geral, eficiente para o entendimento de análise numérica. Desta forma, devido às inconsistências internas, o software não poderia ser aplicado em simulações ou em sistemas de controle no mundo real. Outro ponto importante, é que o algoritmo mesmo já tendo alcançado a raiz aproximada conforme a precisão escolhida, o software continuava procurando a raiz, de tal maneira a se afastar cada vez mais da convergência. Desta forma, percebe-se que após o encontro com a raiz, o algoritmo deixa de apresentar um comportamento monotônico, e oscila, no sentido em que se afasta da raiz, ao invés de aproximar-se cada vez mais.

Bibliografia

[1] Ruggiero, Márcia A. Cálculo Numérico: Aspectos Teóricos e Computacionais. São Paulo: Pearson Education, 2ª edição, 2000.

[2] Barroso, Leônicas Conceição. Cálculo Numérico - Com Aplicações. Sâo Paulo: Editora Harbra, 2ª edição, 1987.

[3] Burden R. L. et al. Numerical Analysis. 9 ed. Brooks/Cole Cengage Learning, 2010.

[4] The Bissection Method. Acesso em: 21 de setembro de 2019.
http://pages.cs.wisc.edu/~sifakis/courses/cs412-s13/lecture_notes/CS412_5_Feb_2013.pdf

[5] Teorema Fundamental da Álgebra. Acesso em: 21 de setembro de 2019.
https://www.ime.usp.br/~oliveira/TFACOLEGIAL5.pdf

[6] Métodos de Cálculo Numérico. Acesso em: 22 de setembro de 2019.
https://sites.google.com/site/calcnum10/home/lista-2/metodos/metodo-de-falsa-posicao

[7] False-position method. Acesso em: 22 de setembro de 2019.
http://nm.mathforcollege.com/mws/gen/03nle/mws_gen_nle_txt_falseposition.pdf

[8] Programming Languague of High Level Python 3.7.3 x64.
https://www.python.org/downloads/

[9] Library Matplotlib 3.1.1
https://matplotlib.org/index.html

[10] Integrated Development Environment(IDE) Visual Studio Code 1.38
https://code.visualstudio.com/download

[11] GUI Library Qt5 for Python.
https://www.qt.io/download

[12] Operational System Linux Ubuntu 19.04.

https://ubuntu.com/#download

[13] Casio Numerical Analysis Calculator.

https://keisan.casio.com/exec/system/1222999061