DOCUMENTAÇÃO

Zero da função

22 DE JUNHO DE 2023

GABRIEL COELHO DOS SANTOS

RENIVALDO LOPES DA SILVA

VÍTOR FERNANDES MARINELLI

Resumo (português)

Esse programa em *python* foi feito para achar o zero de uma função usando o método da bissecção. Este documento visa explicar a lógica usada e a alguns pontos da tecnologia utilizada, junto com algumas ressalvas e os testes realizados.

Abstract (Inglês)

This python program was made to find the zero of a function using the bisection method. This document aims to explain the logic used and some points of the technology used, along with some caveats and the tests performed.

Sumário

Funcionamento do programa

Ao executar o programa abre uma janela onde o usuário pode digitar os valores do intervalo, a função e a taxa de erro. Caso os valores estejam errados, uma mensagem aparece avisando que houve os dados de entrada são incorretos, mas se os valores de entrada forem computáveis, então uma tela mostra os resultados e o x de cada interação, junto com a taxa de erro.

Atenção: Esse programa foi escrito na linguagem *python*, por conta disso números reais devem ser representados por um ponto e não uma virgula, exemplo “4.807213”.

Código

Bibliotecas:

Funções criada por outras pessoas, para facilitar algumas operações.

Nome da biblioteca	Motivo
tkinter	Biblioteca para criação e manipulação de janelas (interface de usuário [UI])
messagebox	Função do *tkinter*, para criar uma janela de *pou-pop*
re	Biblioteca para o uso de *regexs* (excreções regulares)
math	Biblioteca com a implementação de diversas funções da matemática

Classe MyWindow:

Uma classe criada para gerar e manipular uma janela gráfica.

Todas as funções dessa classe têm como parâmetros o self, ou seja, o próprio objeto, então cada função tem acesso a tudo dessa classe.

Função __init__:

Essa função cria uma janela gráfica com o título de “Calculadora zero da função”, e chama a função “criar_interface( )”.

Entradas: *self* (o próprio objeto, nesse caso a janela).

Saídas: não se aplica.

Variáveis locais: não se aplica.

Lógica: não se aplica.

Observações: não se aplica.

Função criar_interface:

Essa função serve para colocar e configurar os elementos na janela gráfica.

Entradas: *self* (o próprio objeto, nesse caso a janela).

Saídas: não se aplica.

Variáveis locais:

Nome	Tipo de elemento	motivo
Titulo	*Label* (texto)	Um texto usado para melhorar o visual do programa
texto_label1	*Label* (texto)	Um texto para indicar ao usuário o que deve ser colocado na caixa de texto
texto_label2	*Label* (texto)	Um texto para indicar ao usuário o que deve ser colocado na caixa de texto
texto_label3	*Label* (texto)	Um texto para indicar ao usuário o que deve ser colocado na caixa de texto
texto_label4	*Label* (texto)	Um texto para indicar ao usuário o que deve ser colocado na caixa de texto
campo1	*Entry* (Caixa de texto)	Para pegar a função fornecida pelo usuário
campo2	*Entry* (Caixa de texto)	Para pegar o início do intervalo
campo3	*Entry* (Caixa de texto)	Para pegar o final do intervalo
campo4	*Entry* (Caixa de texto)	Para pegar a taxa de erro máximo
botao	*Button* (botão)	Iniciar o cálculo

Lógica: O botão calcular está correlacionado com a função calcular, em outras palavras ao apertar o botão, essa função é chamada.

Observações: somente o título recebe uma fonte e um tamanho que é fora do padrão, isso é feito para gerar destaque.

Função Calcular:

Essa função serve para pegar os valores fornecidos pelo usuário, tratá-los, além de chamar a função que iniciara o cálculo e mostra o resultado ou uma mensagem de erro na tela.

Entradas: *self* (o próprio objeto, nesse caso a janela).

Saídas: não se aplica.

Variáveis locais:

Nome	Tipo	motivo
func	Função anônima	Armazena a função criada pelo usuário
a	Número Real	Armazena o início do intervalo indicado pelo usuário
b	Número Real	Armazena o final do intervalo indicado pelo usuário
erro	Número Real	Armazena o valor de erro máximo solicitado pelo usuário
mensagem	Cadeia de caracteres (texto)	Armazena a informação de uma falha de execução ou o resultado das interações

Logica:

Por meio da tecnologia *try e except* do *python*, o computador irar tentar realizar todas as operações que estão dentro do *try*, se algo falhar ele executa o *except*, isso garante que não entre uma letra no lugar de um número por exemplo.

Dentro do *try* primeiro ocorre o tratamento da função (por meio da função criarFunc), e a conversão do intervalo e do erro de um texto para um número Real.

No final do *try* é chamada a função IniciarCalculo, passando os dados já tratados e recebendo um texto, caso ocorra qualquer erro que faria a execução do programa parar, faz o código ir para execução do *except*, assim a execução não é interrompida.

No fim da função é criada uma tela do estilo *pou-pop* que vai mostrar um texto, sendo uma mensagem de erro ou o resultado das interações.

Observações: não se aplica.

Função Iniciar:

Essa função só inicia o *loop* principal que mantem a interface gráfica ativa, essa função vem da biblioteca tkinter.

Entradas: *self* (o próprio objeto, nesse caso a janela).

Saídas: não se aplica.

Variáveis locais: não se aplica.

Lógica: não se aplica.

Observações: não se aplica.

Funções para tratamento:

Aqui é descrito as funções que realizam o tratamento da função fornecida pelo usuário.

Função criarFunc:

Essa função serve para transformar o texto que o usuário digitou em uma função lambda anônima.

Entradas: tf (texto a ser convertido em função)

Saídas: uma função lambda anônima.

Variáveis locais:

Nome	Tipo	Motivo
f	texto	Armazena o texto da função já tratado

Lógica:

O texto que é recebido de entrada é enviado para a função arrumar_func, onde será tratado e devolvido. Esse texto é convertido em uma função lambda anônima, que é a forma do *python* tratar funções matemáticas.

Observações: não se aplica.

Função arrumar_func:

Essa função troca os textos comumente usados em programas de computadores com a finalidade de se aproximar da matemática humana, por textos de funções da biblioteca math do python para o funcionamento correto do programa.

Entradas: f (texto que possui operações matemáticas).

Saídas: um texto modificado para o padrão da biblioteca math.

Variáveis locais:

Nome	Tipo	Motivo
a	Cadeia de caracteres	Ela vai armazenar o texto para poder ser aplicado as modificações sem alterar o texto original

Lógica:

Usando a função *replace* da biblioteca *re* do *python*, é verificado se há uma cadeia de caracteres que corresponde ao texto desejado e substitui pela cadeia desejada. Cada linha vai trocar uma cadeia de caracteres diferentes, referente a cada tipo de função matemática que essa função trata. Após o tratamento de cada função matemática é chamada a função raízes () passando o texto depois das modificações e no final o devolve já preparado para ser transformando em uma função matemática.

Observações: Por limitações da biblioteca math, não há uma forma de fazer logaritmo natural. Isso poderá ser resolvido com o uso de outra biblioteca ou por alguma regra matemática.

As notações matemáticas que essa função trata são:

Notação reconhecida	Equivalente matemático	Notação do python
^	Expoente	**
sen	Seno	math.sin
cos	Cosseno	math.cos
tg	Tangente	math.tan
π	Pi	math.pi
e	Constante de Euler	math.e
!	Fatorial	math.factorial
log	Logaritmo de base 10	math.log10

Função raízes:

Uma função que transforma um texto de uma notação matemática para uma raiz n.

Entradas: f (texto que possui operações matemáticas).

Saídas: um texto modificado para o padrão da biblioteca math.

Variáveis locais:

Nome	Tipo	Motivo
regex	Cadeia de caracteres	Guarda o padrão que vai ser reconhecido
a	Função lambda anônima	Armazena a função usada para a raiz n
resultado	Função lambda anônima	Armazena a função de raiz n

Lógica:

Primeiro criamos o padrão de expressão regular usado para identificar cadeias de texto da seguinte forma “[n]√(x)”, depois criamos uma função lambda anônima $f(x) = x^{\frac{1}{n}}$, a ideia é usar a relação entre expoente de fração e raiz:

$$x^{\frac{a}{n}} \equiv \sqrt[n]{x^a}$$

No final trocamos o padrão pela função já modificada no texto original.

Observações: por padrão adotamos o “a” como 1, pois o importante é a raiz, e queríamos simplificar a sintaxe e a função, caso o usuário queira digitar um expoente é só usar o “^” em conjunto com a raiz.

Funções para o Cálculo:

Nessa etapa são explicadas as funções utilizadas para o cálculo do zero da função e do erro.

Função IniciarCalculo:

Essa função calcula a interação 0 e inicia os cálculos das demais interações.

Entradas: f (função lambda anônima, o equivalente a função matemática), a (número real que marca o início do intervalo), b (número real que marca o fim do intervalo) e erro (número real que marca o erro máximo pedido pelo usuário).

Saídas: Uma cadeia de caracteres com o resultado de cada interação, a taxa de erro e o resultado.

Variáveis locais:

Nome	Tipo	Motivo
x	Número real	Armazena o ponto médio do intervalo inicial
er	Número real	Armazena o erro aproximado

Lógica:

Primeiro calculamos o ponto médio do intervalo [a, b], por meio da média aritmética de “a” e “b”. Depois calculamos o erro atual por meio da fórmula $|a - b|$.

Por fim usamos a função Bolzano( ), para verificar qual subintervalo está o meu zero da função, se for [a, x] então a função calcular, passando esse subintervalo, se não a função calcular usar o subintervalo [x, b].

Observações: não se aplica.

Função Calcular:

Função responsável por fazer o cálculo de cada interação por meio de recursão (a função a chama a si própria a cada interação até o resultado desejado, devolvendo o valor de cada um em um texto único para exibir para o usuário).

Entradas: f (função lambda anônima, o equivalente a função matemática), a (número real que marca o início do intervalo), b (número real que marca o fim do intervalo), erroM (número real que marca o erro máximo pedido pelo usuário), i (número inteiro que indica marca qual é o índice da interação atual), anterior (número real que armazena o x da interação anterior), erro (número real que indica o erro da interação anterior), mensagem (uma cadeia de caracteres que vai armazenado o erro, o i, e o resultado de cada interação)

Saídas: Uma cadeia de caracteres com o resultado de cada interação, a taxa de erro e o resultado.

Variáveis locais:

Nome	Tipo	Motivo
x	Número real	Guarda o ponto médio do intervalo
intervaloRaiz	Número inteiro	Guarda a indicação de qual subintervalo se encontra o ponta médio

Lógica:

Primeiro calculamos o ponto médio do intervalo, depois verificamos pela função de Bolzano em qual subintervalo se encontra o zero da função e calculamos o erro anterior, tudo da mesma forma que na função IniciarCalculo.

Por conta do uso de recursão, não há a necessidade de usar um *loop*, mas temos que usar uma condicional, se a nossa condição de parada for atingida então é devolvido o texto, mas enquanto não for atingido o programa chama a função calcular novamente.

A condição de parada é se o erro atual for menor ou igual ao erro fornecido pelo usuário.

Se a condição para parada não for atingida ainda, a mensagem concatena um texto com os valores de cada interação, atualiza o valor de i, e de acordo com o valor do intervaloRaiz, é decidido se qual subintervalo será enviado para a chamada da função Calcular.

Observações: Para facilitar o entendimento há um exemplo a seguir:

Função = $x^3 - x + 5$

Intervalo = [-2, 2]

Erro = 0.04

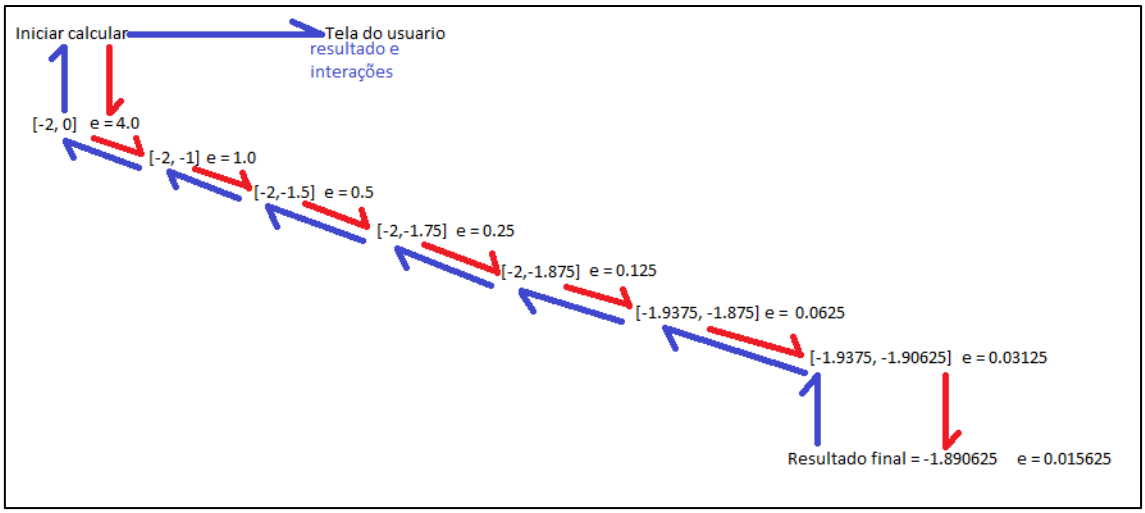

Figura 1: representação do exemplo, mostrando as chamadas e os retornos para exemplificar à recursão

A seta vermelha indica a chama de cada função calcular, e a seta azul indica o retorno de cada função.

Função Bolzano:

Essa função verifica qual dos subintervalos possui o zero da função.

Entradas: f (função lambda anônima, o equivalente a função matemática), a (número real que marca o início do intervalo), x (número real que marca o fim do primeiro subintervalo).

Saídas: um número inteiro, sendo 1 ou 2 para indicar se pertence ao primeiro ou segundo subintervalo.

Variáveis locais:

Nome	Tipo	Motivo
yA	Número Real	Guarda o resultado da aplicação de “a” na função
yX	Número Real	Guarda o resultado da aplicação de “x” na função
intervalo1	Número Real	Serve para verificar qual subintervalo será escolhido
intervalo	Número Inteiro	Indica qual o subintervalo será escolhido

Lógica:

Primeiro aplicamos a “a” e “x” na função, com o resultado desses subintervalos fazemos a multiplicação dos dois, isso é usado para verificar se o valor é menor que zero, ou seja, um teve uma troca de sinal, então a função devolve 1 para indicar que o primeiro subintervalo é onde possui o zero da função. Mas se for positivo (maior que zero), então devolve 2 para indicar que o zero da função fica no segundo subintervalo.

Observações: não se aplica.

Função principal(main):

Aqui temos as funções que iniciam as janelas, iniciando o *loop* que mantem a janela ativa e inicia a classe *MyWindow.*

Testes:

Os primeiros testes demonstraram que havia um erro na condição de parada, o correto era uso do erro atual sendo menor que o erro do usuário para parar os cálculos, mas havíamos usado 3 condições de parada.

Após arrumar isso verificamos que não aparentava ter erros, mas em um teste na hora de apresentar para a turma, foi identificado que o erro ainda estava maior, podendo assim ter mais interações, mas o resultado correspondia com o esperado.

Para demostrar o funcionamento do programa iremos colocar algumas prints, para o exemplo da função $x^3 - x + 5$, intervalo [-2.0, 2.0] e erro de 0.04 (4%):

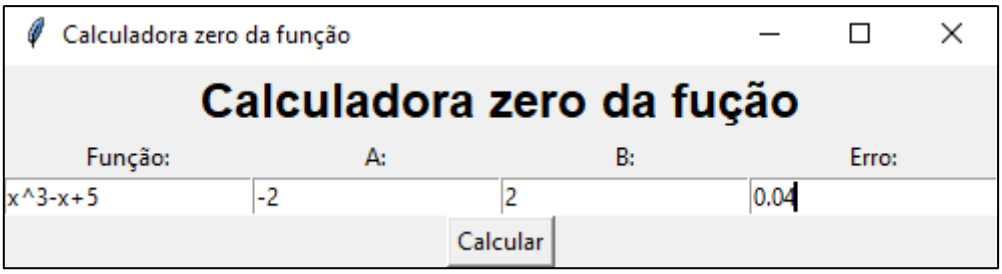

Figura 2: representação da janela para a entrada dos dados pelo usuário

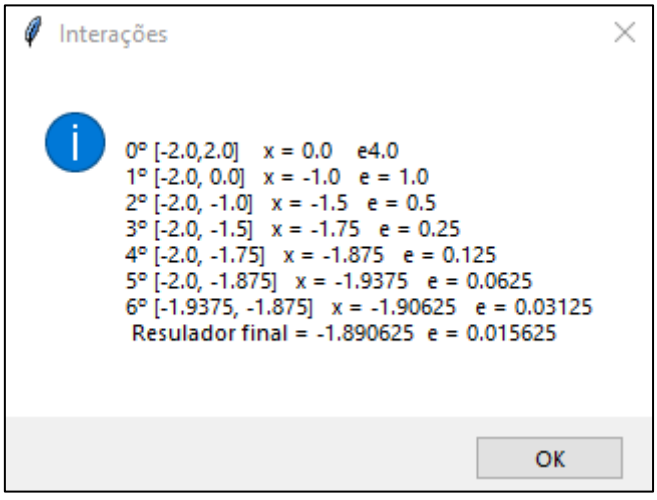

Figura 3: representação da janela pou-pop mostrando que os resultados

Caso algo seja digitado de maneira errada o programa exibe a seguinte mensagem:

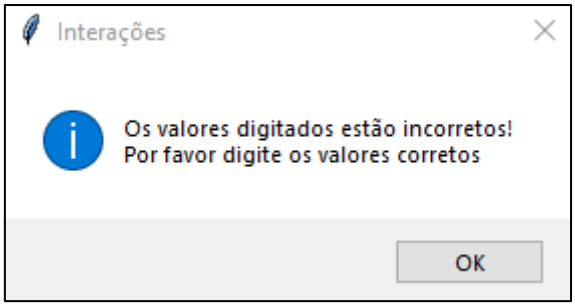

Figura 4: representação da janela pou-pop mostrando um erro

Conclusões:

O programa parece está funcionado para a maioria dos casos que foram testados. Não temos mais o exemplo que foi passado em aula para realizar uma análise mais detalhada, mas acreditamos que o problema foi o fato de o texto apresentado do erro ser da interação erro anterior, dando a impressão que o erro está incorreto ou maior que a taxa de erro que o usuário solicitou. Pedimos desculpas as pessoas que usarem esse software por conta desse detalhe e alertamos que mais testes devem ser feitos para um melhor veredito do funcionamento

Apêndice A – Código completo:

```
import tkinter as tk
from tkinter import messagebox
import re
import math



#region UI
class MyWindow:
    def __init__(self):
        self.janela = tk.Tk()
        self.janela.title("Calculadora zero da função")
        self.criar_interface()
    def criar_interface(self):
        # Configuração da fonte para o título
        fonte_titulo = ("Helvetica", 18, "bold")

        # Criação do título centralizado
        titulo = tk.Label(self.janela, text="Calculadora zero da fução",
font=fonte_titulo)
        titulo.grid(row=0, column=0, columnspan=4)

        # Criação dos textos acima de cada campo
        texto_label1 = tk.Label(self.janela, text="Função:")
        texto_label1.grid(row=1, column=0)
        texto_label2 = tk.Label(self.janela, text="A:")
        texto_label2.grid(row=1, column=1)
        texto_label3 = tk.Label(self.janela, text="B:")
        texto_label3.grid(row=1, column=2)
        texto_label4 = tk.Label(self.janela, text="Erro:")
        texto_label4.grid(row=1, column=3)

        # Criação dos campos de texto
        self.campo1 = tk.Entry(self.janela)
        self.campo1.grid(row=2, column=0)
        self.campo2 = tk.Entry(self.janela)
        self.campo2.grid(row=2, column=1)
        self.campo3 = tk.Entry(self.janela)
        self.campo3.grid(row=2, column=2)
        self.campo4 = tk.Entry(self.janela)
        self.campo4.grid(row=2, column=3)

        # Botão
        botao = tk.Button(self.janela, text="Calcular",
command=self.calcular)
        botao.grid(row=3, column=0, columnspan=4)
```

```
    #ação do botão
    def calcular(self):

        try:
            func = criarFunc(self.campo1.get())
            a = float(self.campo2.get())
            b = float(self.campo3.get())
            erro = float(self.campo4.get())
            mensagem = IniciarCalculo(a,b,func,erro)
        except Exception as e:
            mensagem = "Os valores digitados estão incorretos!\nPor favor
digite os valores corretos"


        messagebox.showinfo("Interações", mensagem)

    def iniciar(self):
        self.janela.mainloop()

#endregion

#region tratamento da função
def criarFunc(tf):
    f = arrumar_func(tf)
    return lambda x: eval(f)


def arrumar_func(f):
  a = f
  a = a.replace('^', '**')
  a = a.replace('sen','math.sin')
  a = a.replace('cos','math.cos')
  a = a.replace('tg','math.tan')
  a = a.replace('π', 'math.pi')
  a = a.replace('e', 'math.e')
  a = a.replace('!', 'math.factorial')
  a = a.replace('log', 'math.log10')
  a = raizes(a)
  return a

def raizes(f):
   regex = r'\[(\d+)\]√\((.+?)\)'
   a = lambda match: f'math.pow(({match.group(2)}), 1/{match.group(1)})'
   resultado = re.sub(regex,a,f)
   return resultado
#endregion

#region calculo do zero da função
def IniciarCalculo(a,b,f,erro):
```

```
    x = (a+b)/2
    er = abs(a-b)
    if(Bolzano(f,a,x) == 1):
        return calcular(a,x,f,erro,1,x,er,f"\n{0}º [{a},{b}]    x =
{x}    e{er}")
    else:
        return calcular(x,b,f,erro,1,x,er,f"\n{0}º [{a},{b}]    x =
{x}    e{er}")


def calcular(a,b,f,erroM,i,anterior,erro,mensagem):
    #divisão do intervalo
    x = (a+b)/2
    intervaloRaiz = Bolzano(f,a,x)
    erro = abs(x-anterior)
    if((b-a<=erroM)):
        mensagem = mensagem + f"\n Resulador final = {x}  e = {erro}\n\n"
        return mensagem
    else:

        mensagem = mensagem + f"\n{i}º [{a}, {b}]   x = {x}   e = {erro}"
        i = i+1

        if(intervaloRaiz == 1):
            return calcular(a,x,f,erroM,i,x,erro,mensagem)
        else:
            return calcular(x,b,f,erroM,i,x,erro,mensagem)



def Bolzano(f,a,x):
    #aplicação das funções
    yA = f(a)
    yX = f(x)
    intervalo1 = yA*yX
    #verificação do intervalo
    if(intervalo1 < 0):
        intervalo = 1
    else:
        intervalo = 2

    return intervalo

#endregion

# Iniciar o loop principal da janela
janela = MyWindow()
janela.iniciar()
```