Laboratório 2 - Zeros de Função

Instituto Tecnológico de Aeronáutica - ITA
Matemática Computacional CCI - 22
Nicholas Scharan Cysne
Turma 22.1

1. Introdução

Temos como objetivo deste laboratório a implementação de três dos cinco métodos aprendidos em aulas para determinação de raízes aproximadas de uma função em determinado intervalo menor que um erro determinado.

2. Resultados

A implementação em questão desejava encontrar uma raiz aproximada do polinômio $p(x) = x^3 - x - 1$ no intervalo $[1, 2]$, com erro ε menor que 10^{-3}. Para tal, foram utilizados os métodos Bissecção, Falsa Posição e Newton-Raphson. A Tabela 1 mostra os resultados obtidos por cada algoritmo e respectivas iterações necessárias.

MÉTODO UTILIZADO	RAIZ APROXIMADA	ITERAÇÕES NECESSÁRIAS
BISSECÇÃO	1.325061645407845	10
FALSA POSIÇÃO	1.3245319865800917	9
NEWTON-RAPHSON	1.324489506882484	6

Tabela 1. Resultados obtidos da implementação dos algoritmos.

3. Código

A implementação dos algoritmos foi feita utilizando Python3 e os códigos fonte estão em anexo no Apêndice A. O código foi feito utilizando uma única Main, importando outros três arquivos que possuem cada algoritmo implementado respectivamente.

4. Conclusão

Através do Software Wolfram Alpha, calculamos a raiz real do polinômio $p(x)$ com precisão 10^{-4}, obtendo $x_o = 1,3247$. Comparando em termos de precisão, o método da Falsa Posição encontrou o valor mais próximo do real com 9 iterações ocorridas, sendo $x_{FP} = 1.3245$, seguido pelo método de Newton-Raphson com $x_{NR} = 1.3244$, em que analisando número de iterações, concluído em 6 passos, e precisão mostrou maior eficiência em relação aos outros. O método da Bissecção resultou em menor precisão, com $x_B = 1.3250$ e maior número de iterações, concluindo com 10 passos, sendo o menos eficiente.

APÊNDICE A

Arquivo “Main.py”:

```
# Nicholas Scharan Cysne
# T22.1
# CCI-22 Professor: Johnny
# Zero de Funções

from bissector import *
from falsePosition import *
from fixedPoint import *

# Calculate function value via polinom
def calculate(coef, a):
	result = 0
	exp = len(coef) - 1

	for c in coef:
		result += c*math.pow(a, exp)
		exp -= 1
	return result

# Parameters definitions
interval = [1, 2]		# Interval of search
epsilon = 1e-3		# Acceptable error epsilon1 = epsilon2
polinom = [1, 0, -1, -1]	# x³ - x - 1
parameters = [[1, 2], epsilon, polinom]

print("Raiz utilizando o Método da Bissecção: {}".format(bissector(parameters, calculate)))
parameters = [[1, 2], epsilon, polinom]
print("Raiz utilizando o Método da Falsa Posição: {}".format(falsePosition(parameters,
calculate)))
parameters = [[1, 2], epsilon, polinom]
print("Raiz utilizando o Método do Newton-Raphson: {}".format(newtonRaphson(parameters,
calculate)))
```

Arquivo “bissector.py”:

```
# Nicholas Scharan Cysne
# T22.1
# CCI-22 Professor: Johnny
# Zero de Funções
# Método da Bissecção

import random

# Bissector Method Algorithm
def bissector(parameters, calculate):

    interval = parameters[0]
    epsilon = parameters[1]
    polinom = parameters[2]
    counter = 0

    if interval[1] - interval[0] < epsilon:
        return random.uniform(interval[0], interval[1]), counter

    k = 1
    while True:
        counter += 1
        M = calculate(polinom, interval[0])
        x = (interval[0] + interval[1])/2

        if M*calculate(polinom, x) > 0:
            interval[0] = x
        else:
            interval[1] = x

        if interval[1] - interval[0] < epsilon:
            return random.uniform(interval[0], interval[1]), counter
        k += 1
```

Arquivo: “falsePosition.py”

```
# Nicholas Scharan Cysne
# T22.1
# CCI-22 Professor: Johnny
# Zero de Funções
# Método da Falsa Posição

import random
import math

def falsePosition(parameters, calculate):
    interval = parameters[0]
    epsilon = parameters[1]
    polinom = parameters[2]
    counter = 0
    if interval[1] - interval[0] < epsilon:
        return random.uniform(interval[0], interval[1]), counter
    if math.fabs(calculate(polinom, interval[0])) < epsilon:
        return interval[0], counter
    if math.fabs(calculate(polinom, interval[1])) < epsilon:
        return interval[1], counter

    while True:
        counter += 1
        fa = calculate(polinom, interval[0])
        fb = calculate(polinom, interval[1])
        x = (interval[0]*fb - interval[1]*fa)/(fb - fa)
        M = fa

        if math.fabs(calculate(polinom, x)) < epsilon:
            return x, counter
        if M*calculate(polinom, x) > 0:
            interval[0] = x
        else:
            interval[1] = x
        if interval[1] - interval[0] < epsilon:
            return random.uniform(interval[0], interval[1]), counter
```

Arquivo: “fixedPoint.py”:

```
# Nicholas Scharan Cysne
# T22.1
# CCI-22 Professor: Johnny
# Zero de Funções
# Método de Newton-Raphson

import random
import math

def newtonRaphson(parameters, calculate):

    interval = parameters[0]
    epsilon = parameters[1]
    polinom = parameters[2]
    counter = 0

    x = interval[0]
    if math.fabs(calculate(polinom, x)) < epsilon:
        return interval[0], counter

    while True:
        counter += 1
        x_back = x
        x = x - calculate(polinom, x)/calculate([0, 3, 0, 1],x)
        if math.fabs(calculate(polinom, x)) < epsilon:
            return x, counter
        if math.fabs(x - x_back) < epsilon:
            return x, counter
```