A picture containing shape

Description automatically generated

**Kaunas university of technology**

**Faculty of Informatics**

P170B115 Numerical methods and algorithms

Lab work 1 report

**Lecturer:** Assoc. prof. Andrius KRIŠČIŪNAS

**Student:** Mehmet Şahin

**Group:** IFU-9

**Kaunas 2021**

**Contents**

1-Tasks………………………………………………………..….….2

2-Implementations…………………………………………………..3-10

3-Results……………………………………………………..……..11-12

4-Third equation implementation……………………………………...13

5 Results ………………………………………………………………14

6 Theory……………………………………………………………….15

**Equations:**



Table

Description automatically generated

**Task:**

* Finding Interval of F(x)

0,85x^4 – 9,92x^3 +40,02x^2 – 64,68x+34,25

Text

Description automatically generatedA1 = 0,85

A2 = – 9,92

A3 = 40,02

A4 = – 64,68

A5 = 34,25

1+40,02**/**0,85**=** 48,25 **->** (**48,25, -48,25**)

Graphical user interface, application, Teams

Description automatically generated

*Figure 1.0 results from wolframalpha*

F(X)

CHORD

import math

import numpy as np;

import matplotlib.pyplot as plt

def func(x):

    return 0.85 \* x \*\* 4 - 9.92 \* x \*\* 3 + 40.02 \* x \*\* 2 - 64.68 \* x + 34.25

def SearchforRoots(func,a1,a2):

    step=0.01

    x1 = a1; f1 = func(a1)

    x2 = a1 + step; f2 = func(x2)

    while f1\*f2 > 0.0:

        if x1 >= a2:

            return None,None

        x1 = x2; f1 = f2

        x2 = x1 + step; f2 = func(x2)

    return x1,x2

def Chord(func,x0,x1):

    e=1e-10

    Counter = 1

    print('\n\nITERATIONSOFCHORD')

    condition = True

    while condition:

        x2 = (x1\*func(x0)- x0\*func(x1))/(func(x0)-func(x1))

        print('Iteration-%d, x2 = %0.6f and f(x2) = %0.6f' % (Counter, x2, func(x2)))

        if func(x0) \* func(x2) < 0:

           x1 = x2

        else:

            x0 = x2

        Counter = Counter + 1

        condition = abs(func(x2)) > e

    print('\nRequired root is: %0.8f' % x2)

    return x2

def TotalRoots(func, a1, a2, eps=1e-6):

    print ('The roots on the interval [%f, %f] are:' % (a1,a2))

    while 1:

        x1,x2 = SearchforRoots(func,a1,a2)

        if x1 != None:

            a1 = x2

            root = Chord(func,x1,x2)

            if root != None:

                pass

                print (round(root,-int(math.log(eps, 10))))

        else:

            print ('\nDone')

            break

TotalRoots(func, 0, 5)

x = np.linspace(0, 5, 100)

y = func(x)

#plt.plot(x, y)

plt.grid()

#plt.plot(x1, func(x1), markersize=7, marker='o')

SECANT

import math

import numpy as np;

import matplotlib.pyplot as plt

def func(x):

    return 0.85 \* x \*\* 4 - 9.92 \* x \*\* 3 + 40.02 \* x \*\* 2 - 64.68 \* x + 34.25

def SearchforRoots(func,a1,a2):

    step=0.01

    x1 = a1; f1 = func(a1)

    x2 = a1 + step; f2 = func(x2)

    while f1\*f2 > 0.0:

        if x1 >= a2:

            return None,None

        x1 = x2; f1 = f2

        x2 = x1 + step; f2 = func(x2)

    return x1,x2

def Secant(func, x0, x1):

    e = 1e-10

    N = 100

    print('\n\nIterations:')

    counter = 1

    condition = True

    while condition:

        if func(x0) == func(x1):

            print('ERROR!')

            break

        x2 = x0 - (x1 - x0) \* func(x0) / (func(x1) - func(x0))

        print('Iteration-%d, x2 = %0.4f and f(x2) = %0.6f' % (counter, x2, func(x2)))

        x0 = x1

        x1 = x2

        counter = counter + 1

        if counter > N:

            print('No Root Found!')

            break

        condition = abs(func(x2)) > e

    print('\n Exact root is: %0.8f' % x2)

    return x2

def TotalRoots(func, a1, a2, eps=1e-6):

    print ('The roots on the interval [%f, %f] are:' % (a1,a2))

    while 1:

        x1,x2 = SearchforRoots(func,a1,a2)

        if x1 != None:

            a1 = x2

            root = Secant(func,x1,x2)

            if root != None:

                pass

                print (round(root,-int(math.log(eps, 10))))

        else:

            print ('\nDone')

            break

TotalRoots(func, 0, 5)

x = np.linspace(0, 5, 100)

y = func(x)

plt.plot(x, y)

plt.grid()

#plt.plot(x1, func(x1), markersize=7, marker='o')

G(X)

SECANT

import math

import numpy as np;

import matplotlib.pyplot as plt

e=2.220446049250313e-16

def func(x):

    return math.cos(2\*x)\*e\*\*(-x/2)\*\*2  #0.85 \* x \*\* 4 - 9.92 \* x \*\* 3 + 40.02 \* x \*\* 2 - 64.68 \* x + 34.25

def SearchforRoots(func,a1,a2):

    step=0.01

    x1 = a1; f1 = func(a1)

    x2 = a1 + step; f2 = func(x2)

    while f1\*f2 > 0.0:

        if x1 >= a2:

            return None,None

        x1 = x2; f1 = f2

        x2 = x1 + step; f2 = func(x2)

    return x1,x2

def Secant(func, x0, x1):

    e = 1e-10

    N = 100

    print('\n\nIterations:')

    counter = 1

    condition = True

    while condition:

        if func(x0) == func(x1):

            print('ERROR!')

            break

        x2 = x0 - (x1 - x0) \* func(x0) / (func(x1) - func(x0))

        print('Iteration-%d, x2 = %0.4f and f(x2) = %0.6f' % (counter, x2, func(x2)))

        x0 = x1

        x1 = x2

        counter = counter + 1

        if counter > N:

            print('No Root Found!')

            break

        condition = abs(func(x2)) > e

    print('\n Exact root is: %0.8f' % x2)

    return x2

def TotalRoots(func, a1, a2, eps=1e-6):

    print ('The roots on the interval [%f, %f] are:' % (a1,a2))

    while 1:

        x1,x2 = SearchforRoots(func,a1,a2)

        if x1 != None:

            a1 = x2

            root = Secant(func,x1,x2)

            if root != None:

                pass

                print (round(root,-int(math.log(eps, 10))))

        else:

            print ('\nDone')

            break

TotalRoots(func, -6, 6)

CHORD

import math

import numpy as np;

import matplotlib.pyplot as plt

e=2.220446049250313e-16

def func(x):

    return math.cos(2\*x)\*e\*\*(-x/2)\*\*2  #0.85 \* x \*\* 4 - 9.92 \* x \*\* 3 + 40.02 \* x \*\* 2 - 64.68 \* x + 34.25

def SearchforRoots(func,a1,a2):

    step=0.01

    x1 = a1; f1 = func(a1)

    x2 = a1 + step; f2 = func(x2)

    while f1\*f2 > 0.0:

        if x1 >= a2:

            return None,None

        x1 = x2; f1 = f2

        x2 = x1 + step; f2 = func(x2)

    return x1,x2

def Chord(func,x0,x1):

    e=1e-10

    Counter = 1

    print('\n\nITERATIONSOFCHORD')

    condition = True

    while condition:

        x2 = (x1\*func(x0)- x0\*func(x1))/(func(x0)-func(x1))

        print('Iteration-%d, x2 = %0.6f and f(x2) = %0.6f' % (Counter, x2, func(x2)))

        if func(x0) \* func(x2) < 0:

           x1 = x2

        else:

            x0 = x2

        Counter = Counter + 1

        condition = abs(func(x2)) > e

    print('\nRequired root is: %0.8f' % x2)

    return x2

def TotalRoots(func, a1, a2, eps=1e-6):

    print ('The roots on the interval [%f, %f] are:' % (a1,a2))

    while 1:

        x1,x2 = SearchforRoots(func,a1,a2)

        if x1 != None:

            a1 = x2

            root = Secant(func,x1,x2)

            if root != None:

                pass

                print (round(root,-int(math.log(eps, 10))))

        else:

            print ('\nDone')

            break

TotalRoots(func, -6, 6)

**Results**

Chart, line chart

Description automatically generated

Table

Description automatically generated

*Figure 1.1 result of f(x) with secant method Figure 1.2 result of f(x) with chord method*

Table

Description automatically generated with low confidence

Table

Description automatically generated

*Figure 1.3 result of g(x) with chord method Figure 1.4 result of g(x) with secant method*

Text

Description automatically generated with low confidenceTable

Description automatically generated

*Figure 1.5 result of f(x) with chord method ,all roots Figure 1.6 result of f(x) with secant method, all roots*

Chart, line chart

Description automatically generated

*Figure 1.7 f(x) all roots,represented*

Chart, line chart

Description automatically generated

*Figure 1.8 f(x) all roots,represented*

**Third equation:**

**-110x\*e^15k+1**

import matplotlib.pyplot as plt

import numpy as np

import  time  as tm

def func(x):

  return -110\*np.exp(15\*x)+1;

def Secant(func, x0, x1):

    e = 1e-10

    N = 100

    print('\n\nIterations:')

    counter = 1

    condition = True

    while condition:

        if func(x0) == func(x1):

            print('ERROR!')

            break

        x2 = x0 - (x1 - x0) \* func(x0) / (func(x1) - func(x0))

        print('Iteration-%d, x2 = %0.4f and f(x2) = %0.6f' % (counter, x2, func(x2)))

        x0 = x1

        x1 = x2

        counter = counter + 1

        if counter > N:

            print('No Root Found!')

            break

        condition = abs(func(x2)) > e

    print('\n Exact root is: %0.8f' % x2)

    return x2

Secant(func,0,110)

x = np.linspace(-10, 10, 100)

y = func(x)

plt.plot(x, y)

plt.grid()

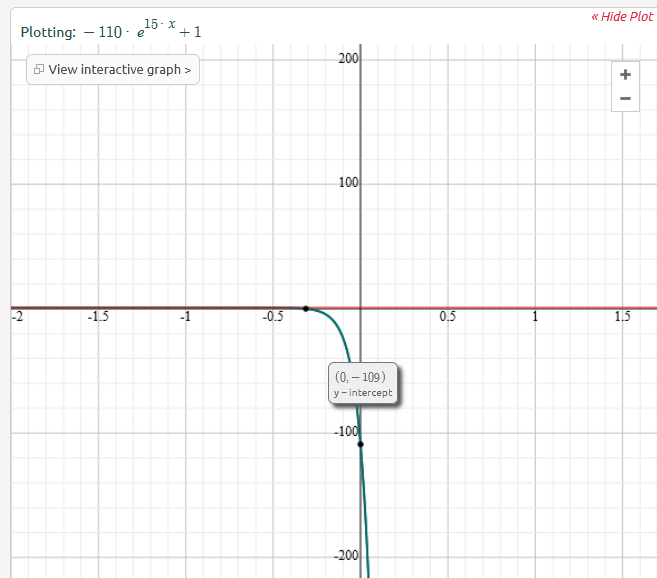
plt.show()

Graphical user interface

Description automatically generated with low confidence



*Figure 1.9 one root of third equations,represented*





*Figure 1.10 third equation’s all roots,represented*

**Secant method:**

Initialization: Two initial interval must be picked xn, xn+1

Xn+1 = xn – func(xn) \* (xn – xn-1) / func(xn) – func(xn-1)

It is clear from the numerical results that the secant method  
requires more iterates than the Newton method (e.g., with  
Newton’s method, the iterate is accurate). But note that the secant  
method does not require a knowledge of f ′(x ), whereas Newton’s  
method requires both f (x ) and f ′(x ). Note also that the secant method can be considered anapproximation of the Newton method.

**Features of secant method :**  
 - It converges at faster than a linear rate, so that it is more  
rapidly convergent than the bisection method.  
- It does not require use of the derivative of the function,  
something that is not available in a number of applications.  
 - It requires only one function evaluation per iteration, as  
compared with Newton’s method which requires two.  
 - It may not converge.  
 There is no guaranteed error bound for the computed iterates.  
 - It is likely to have difficulty if f ′(α) = 0. This means the x -axis  
is tangent to the graph of y = f (x ) at x = α.  
 -Newton’s method generalizes more easily to new methods for  
solving simultaneous systems of nonlinear equations.

**The Method of False Position**

The poor convergence of the bisection method as well as its poor adaptability to higher dimensions (i.e., systems of two or more non-linear equations) motivate the use of better techniques. One such method is the *Method of False Position*. Here, we start with an initial interval [*x*1,*x*2], and we assume that the function changes sign only *once* in this interval. Now we find an *x*3 in this interval, which is given by the intersection of the *x* axis and the straight line passing through (*x*1,*f*(*x*1)) and (*x*2,*f*(*x*2)). It is easy to verify that *x*3 is given by

|  |  |
| --- | --- |
| Diagram  Description automatically generated with medium confidence |  |

*Figure 1.11 fasle position method*

Now, we choose the new interval from the two choices [*x*1,*x*3] or [*x*3,*x*2] depending on in which interval the function changes sign.

The false position method differs from the bisection method only in the choice it makes for subdividing the interval at each iteration. It converges faster to the root because it is an algorithm which uses appropriate weighting of the intial end points *x*1 and *x*2 using the information about the function, or the data of the problem. In other words, finding *x*3 is a *static* procedure in the case of the bisection method since for a given *x*1 and *x*2, it gives *identical* *x*3, no matter what the function we wish to solve. On the other hand, the false position method uses the information about the function to arrive at *x*3.

**References:**

<https://www.math.usm.edu/lambers/mat772/fall10/lecture4.pdf>

<https://mathworld.wolfram.com/MethodofFalsePosition.html>

<https://www.youtube.com/watch?v=jy5WiicoEDY>

https://www.youtube.com/watch?v=qFJGMBDfFMY&list=PLkZjai-2Jcxn35XnijUtqqEg0Wi5Sn8ab

https://www.youtube.com/watch?v=pStQf71WwB4&list=PLgH3sgdvgO4QXFTC27OOog4RL2NCUkC0u

<https://www.wolframalpha.com/input/?i=using+secant+method+solve+x%5E3-2+at+x1%3D-3+and+x2%3D3>

<https://mathworld.wolfram.com/SecantMethod.html>

http://web.mit.edu/10.001/Web/Course\_Notes/NLAE/node5.html