Department Of Aerospace Engineering,
Indian Institute Of Technology Madras

AS2101 : Introduction to Aerospace Engineering

Report 4 : Finding the Roots of a Funtion using Bisection, Newton and Secant Method

Pranit Zope
AE20B046

September 28, 2021

Contents

List of Figures

A Python code for computing and Plotting the roots of a mathematical function

```
#!/usr/bin/python3.9
# Pranit Zope
# AE20B046
# AS2101 : Assignment 04

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

def f1(x):
    return (x**3-3*x**2+x+9)
def f2(x):
    return math.exp(x)*(x**3-3*x**2+x+9)

def f1dash(x):
    return (3*x**2-6*x+1)
def f2dash(x):
    return math.exp(x)*(x**3-5*x+10)
#defined all the necessary mathematical functions which we will be
                                    needing

def bisection(f,a,b,spec=10**-12,counter=50):
    """A function that computes the root of a function in x by
                                        Bisection method.

    Args:
        f (Function): A mathematical Function in variable x
        a (float): Lower limit of calling the bisection function
        b (float): Upper limit of calling the bisection function
        spec (float, optional): The specification value upto which you
                                            want the accuracy to be
                                            maintained. Defaults to 10
                                            **-12.
        counter (int, optional): Maximum number if iterations. Defaults
                                             to 50.

    Returns:
        float: The value of the root of the function f(x).
    """

    if f(a)*f(b)>=0:
        return "Error"
    else:
        counter_index=0
        c=1
        while(abs(f(c))>spec and counter_index<counter):
            c=(a+b)/2
            if f(c)*f(b)>0:
                b=c
            if f(c)*f(a)>0:
                a=c
            counter_index+=1
        return c
```

```
def newton(f,fdash,x,spec=10**-12,counter=50):
    """A function that computes the root of a function in x by Newton
                                        method.

    Args:
        f (function):  A mathematical Function in variable x
        fdash (function):  A mathematical Function in variable x which
                                            is the derivative of f(x)
        x (float): The value at which the Newton function should be
                                            initialised.
        spec (float, optional): The specification value upto which you
                                            want the accuracy to be
                                            maintained. Defaults to 10
                                            **-12.
        counter (int, optional): Maximum number if iterations. Defaults
                                             to 50.

    Returns:
        float: The value of the root of the function f(x).
    """

    counter_index=0
    while(abs(f(x))>spec and counter_index<counter):
        counter_index+=1
        x=x-f(x)/fdash(x)
    return x



def secant(f,x0,x1,spec=10**-12,counter=50):
    """A function that computes the root of a function in x by Secant
                                        method.

    Args:
        f (function):  A mathematical Function in variable x
        x0 (float): Lower limit of calling the Secant function
        x1 (float): Upper limit of calling the Secant function
        spec (float, optional): The specification value upto which you
                                            want the accuracy to be
                                            maintained. Defaults to 10
                                            **-12.
        counter (int, optional): Maximum number if iterations. Defaults
                                             to 50.

    Returns:
        x1: The value of the root of the function f(x).
    """

    counter_index=0
    x1=x1-((x0-x1)/(f(x0)-f(x1))*f(x1))
    while(abs(f(x1))>spec and counter_index<counter):
        counter_index+=1
        x1=x1-((x0-x1)/(f(x0)-f(x1))*f(x1))
    return x1
```

```
print("First function with Bisection Method")
print(bisection(f1,-2,-1,0.00001,100))
print("Second function with Bisection Method")
print(bisection(f2,-2,-1,0.00001,100))

print("First function with Newton's method")
print(newton(f1,f1dash,-1,0.00001,100))
print("Second function with Newton's method")
print(newton(f2,f2dash,-1,0.00001,100))

print("First function with Secant Method")
print(secant(f1,-2,-1,0.00001,100))
print("Second function with Secant Method")
print(secant(f2,-2,-1,0.00001,100))
#Printing the outcomes of executing the Bisection, Newton and Secant
                                    Functions


#### SECTION TO PLOT THE FUNCTION VS ITERATION GRAPH OF f1(x) WITH ALL
                                    THE THREE METHODS ####
x=[]
y=[]
for i in range(1,50):
    x.append(i)
    y.append(f1(bisection(f1,-2,-1,counter=i)))
plt.xlabel("Number of Iterations")
plt.ylabel("Value of Function")
plt.plot(x,y,label="f1(x) with Bisection Method",marker=".",markersize=
                                    "5")


x=[]
y=[]
for i in range(1,50):
    x.append(i)
    y.append(f1(newton(f1,f1dash,-1,counter=i)))
plt.xlabel("Number of iterations")
plt.ylabel("Value of function")
plt.plot(x,y,label="f1(x) with Newton's method",marker=".",markersize="
                                    5")


x=[]
y=[]
for i in range(1,50):
    x.append(i)
    y.append(f1((secant(f1,-2,-1,counter=i))))
plt.plot(x,y,label="f1(x) with Secant method",marker=".",markersize="5"
                                    )
plt.grid(linestyle='--')
plt.legend()
plt.savefig("f1_all")
plt.clf()


#### SECTION TO PLOT THE FUNCTION VS ITERATION GRAPH OF f1(x) WITH ALL
                                    THE THREE METHODS ####
```

```
x=[]
y=[]
for i in range(1,50):
    x.append(i)
    y.append(f2(bisection(f2,-2,-1,counter=i)))
plt.xlabel("Number of iterations")
plt.ylabel("Value of function")
plt.plot(x,y,label="f2(x) with Bisection Method",marker=".",markersize=
                                    "5")

x=[]
y=[]
for i in range(1,50):
    x.append(i)
    y.append(f2(newton(f2,f2dash,-1,counter=i)))
plt.plot(x,y,label="f2(x) with Newton's method",marker=".",markersize="
                                    5")

x=[]
y=[]
for i in range(1,50):
    x.append(i)
    y.append(f2(secant(f2,-2,-1,counter=i)))
plt.plot(x,y,label="f2(x) with Secant method",marker=".",markersize="5"
                                    )
plt.grid()
plt.legend()
plt.savefig("f2_all")
plt.clf()

#### SECTION TO PLOT THE FUNCTION VS ITERATION GRAPH OF BOTH FUNCTIONS
                                    WITH BISECTION METHOD ####
x=[]
y=[]
z=[]
for i in range(1,50):
    x.append(i)
    y.append(f1(bisection(f1,-2,-1,counter=i)))
    z.append(f1(bisection(f1,-3,-1,counter=i)))
plt.plot(x,y,label="f1(x) with Bisection Method, Point -2",marker=".",
                                    markersize="5")
plt.plot(x,z,label="f1(x) with Bisection Method, Point -3",marker=".",
                                    markersize="5")
plt.grid()
plt.legend()
plt.savefig("f1b")
plt.clf()

x=[]
y=[]
z=[]
for i in range(1,50):
    x.append(i)
    y.append(f2(bisection(f2,-2,-1,counter=i)))
    z.append(f2(bisection(f2,-1.5,-1,counter=i)))
plt.plot(x,y,label="f2(x) with Bisection Method, Point -2",marker=".",
                                    markersize="5")
```

```
plt.plot(x,z,label="f2(x) with Bisection Method, Point -1.5",marker="."
                                        ,markersize="5")
plt.grid()
plt.legend()
plt.savefig("f2b")
plt.clf()

#### SECTION TO PLOT THE FUNCTION VS ITERATION GRAPH OF BOTH FUNCTIONS
                                        WITH NEWTON METHOD ####
x=[]
y=[]
z=[]
for i in range(1,50):
    x.append(i)
    y.append(f1(newton(f1,f1dash,-1,counter=i)))
    z.append(f1(newton(f1,f1dash,-2,counter=i)))
plt.plot(x,y,label="f1(x) with Newton Method, Point -1",marker=".",
                                        markersize="5")
plt.plot(x,z,label="f1(x) with Newton Method, Point -2",marker=".",
                                        markersize="5")
plt.xlabel("Number of iterations")
plt.ylabel("Value of function")
plt.grid()
plt.legend()
plt.savefig("f1n")
plt.clf()

x=[]
y=[]
z=[]
for i in range(1,50):
    x.append(i)
    y.append(f2(newton(f2,f2dash,-1,counter=i)))
    z.append(f2(newton(f2,f2dash,-2,counter=i)))
plt.plot(x,y,label="f2(x) with Newton Method, Point -1",marker=".",
                                        markersize="5")
plt.plot(x,z,label="f2(x) with Newton Method, Point -2",marker=".",
                                        markersize="5")
plt.xlabel("Number of iterations")
plt.ylabel("Value of function")
plt.grid()
plt.legend()
plt.savefig("f2n")
plt.clf()


#### SECTION TO PLOT THE FUNCTION VS ITERATION GRAPH OF BOTH FUNCTIONS
                                        WITH SECANT METHOD ####
x=[]
y=[]
z=[]
for i in range(1,50):
    x.append(i)
    y.append(f1(secant(f1,-2,-1,counter=i)))
    z.append(f1(secant(f1,-2,-1.5,counter=i)))
plt.plot(x,y,label="f1(x) with Secant Method, Point -1",marker=".",
                                        markersize="5")
```

```
plt.plot(x,z,label="f1(x) with Secant Method, Point -1.5",marker=".",
                                        markersize="5")
plt.xlabel("Number of iterations")
plt.ylabel("Value of function")
plt.grid()
plt.legend()
plt.savefig("f1s")
plt.clf()

x=[]
y=[]
z=[]
for i in range(1,50):
    x.append(i)
    y.append(f2(secant(f2,-2,-1,counter=i)))
    z.append(f2(secant(f2,-2,-1.5,counter=i)))
plt.plot(x,y,label="f2(x) with Secant Method, Point -1",marker=".",
                                        markersize="5")
plt.plot(x,z,label="f2(x) with Secant Method, Point -1.5",marker=".",
                                        markersize="5")
plt.xlabel("Number of iterations")
plt.ylabel("Value of function")
plt.grid()
plt.legend()
plt.savefig("f2s")
plt.clf()
```