*#import all libraries*

**import** numpy **as** np

**import** matplotlib.pyplot **as** ppt

# Question 1a

The use of Newton Raphson method to find the root of a function. Step to solve

1. Let x be a guess of the root of f(x) = 0
2. Define all user functions
3. Compute a variable u = f(x)/df(x)
4. Let x = x - u and repeat step using a loop until u> tol

f(x) = x^3-10x^2 + 5 = 0, x = 0.7, tol = 10^-9

x = 0.7 *# A variable x is defined as an assumed initial value*

​

*#All user functions are defined*

**def** f(x):

**return** (x**\*\***3) **-** (10 **\*** x**\*\***2) **+** 5

**def** df(x):

**return** (3 **\*** x**\*\***2) **-** (20 **\*** x) *#Derivative of the function is defined*

​

mylist=[] *#A list is defined to store all intermidiate values of x*

**def** newtonraphson(f, df, x, tol= 10**\*\*-**9):

​

u = (f(x)**/**df(x))

**while** abs(u) **>**=tol:

u = (f(x)**/**df(x))

x = x **-** u

mylist.append(x)

**return** x

root = newtonraphson(f, df, 0.7, tol= 10**\*\*-**9)

print("The root of the equation is {}" .format(root))

The root of the equation is 0.7346035077893033

*#Test for convergence by plotting graph*

c =np.linspace(0,1,len(mylist))

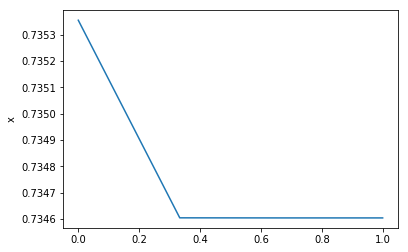
xvals= mylist

ppt.ylabel("x")

ppt.plot(c, xvals)

Out[3]:

[<matplotlib.lines.Line2D at 0x2aa3e0f44e0>]



# Question 1b

The use of Newton Raphson method to find the root of a function. Step to solve

1. Let x be a guess of the root of f(x) = 0
2. Define all user functions
3. Compute a variable u = f(x)/df(x)
4. Let x = x - u and repeat step using a loop until u> tol

f(x) = x^3-2x-5 = 0, x = 2.0, tol = 10^-9

​

x = 2.0 *# A variable x is defined as an assumed initial value*

​

*#All user functions are defined*

**def** f(x):

**return** (x**\*\***3) **-** (2 **\*** x) **-** 5

**def** df(x):

**return** (3 **\*** x**\*\***2) **-** (2) *#Derivative of the function is defined*

​

mylist=[] *#A list is defined to store all intermidiate values of x*

​

**def** newtonraphson(f, df, x, tol= 10**\*\*-**9):

u = (f(x)**/**df(x))

**while** abs(u) **>**=tol:

u = (f(x)**/**df(x))

x = x **-** u

mylist.append(x)

**return** x

root = newtonraphson(f, df, 2.0, tol= 10**\*\*-**9)

print("The root of the equation is {}" .format(root))

The root of the equation is 2.0945514815423265

*#Test for convergence by plotting graph*

c =np.linspace(0,1,len(mylist))

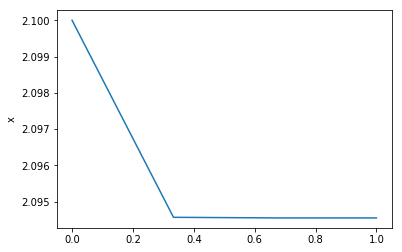
xvals= mylist

ppt.ylabel("x")

ppt.plot(c, xvals)

Out[5]:

[<matplotlib.lines.Line2D at 0x2aa3e148dd8>]



# Question 1c

The use of Newton Raphson method to find the root of a function. Step to solve

1. Let x be a guess of the root of f(x) = 0
2. Define all user functions
3. Compute a variable u = f(x)/df(x)
4. Let x = x - u and repeat step using a loop until u> tol

f(x) = x-2sin(x) = 0, x = 1.5, tol = 10^-9

*#math library is imported for trignometric functions*

**import** math

x = 1.5 *# A variable x is defined as an assumed initial value*

​

*#All user functions are defined*

**def** f(x):

**return** (x) **-** (2 **\*** math.sin(x))

**def** df(x):

**return** 1 **-** (2 **\*** math.cos(x)) *#Derivative of the function is defined*

​

mylist=[] *# A list is defined to store all intermidiate values of x*

​

​

**def** newtonraphson(f, df, x, tol= 10**\*\*-**9):

u = (f(x)**/**df(x))

**while** abs(u) **>**=tol:

u = (f(x)**/**df(x))

x = x **-** u

mylist.append(x)

**return** x

root = newtonraphson(f, df, 1.5, tol= 10**\*\*-**9)

print("The root of the equation is {}" .format(root))

The root of the equation is 1.895494267033981

*#Test for convergence by plotting graph*

c =np.linspace(0,1,len(mylist))

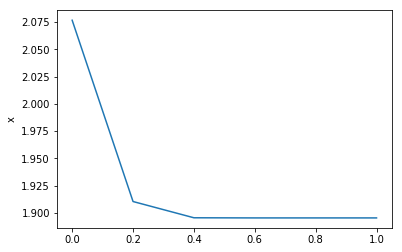
xvals= mylist

ppt.ylabel("x")

ppt.plot(c, xvals)

Out[7]:

[<matplotlib.lines.Line2D at 0x2aa3e1950b8>]



# Question 1d

The use of Newton Raphson method to find the root of a function. Step to solve

1. Let x be a guess of the root of f(x) = 0
2. Define all user functions
3. Compute a variable u = f(x)/df(x)
4. Let x = x - u and repeat step using a loop until u> tol

f(x) = x-2sin(x) = 0 We try to see a scenario where the newton raphson method wont work by choosing a different starting point. x = 13 tol = 10^-9

x = 13 *# A new variable x is defined as an assumed initial value*

​

*#All user functions are defined*

**def** f(x):

**return** (x) **-** (2 **\*** math.sin(x))

**def** df(x):

**return** 1 **-** (2 **\*** math.cos(x)) *#Derivative of the function is defined*

​

mylist=[] *# A list is defined to store all intermidiate values of x*

​

​

**def** newtonraphson(f, df, x, tol= 10**\*\*-**9):

u = (f(x)**/**df(x))

**while** abs(u) **>**=tol:

u = (f(x)**/**df(x))

x = x **-** u

mylist.append(x)

**return** x

root = newtonraphson(f, df, 13, tol= 10**\*\*-**9)

print("The root of the equation is {}" .format(root))

The root of the equation is 0.0

*#Test for convergence by plotting graph*

c =np.linspace(0,1,len(mylist))

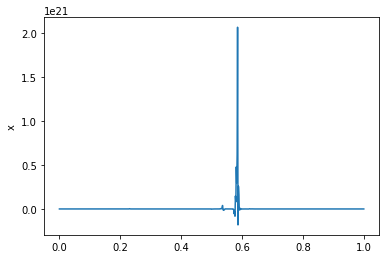
xvals= mylist

ppt.ylabel("x")

ppt.plot(c, xvals)

Out[7]:

[<matplotlib.lines.Line2D at 0x2e881320be0>]



# Bonus question.

A loan of A pound is repaid by making n equal monthly payments of M pounds, starting a month after the loan is made. It can be shown that if the monthly interest rate is r, then

Ar = M(1 - 1/(1+r)^n)

A car loan of £10,000 was repaid in 60 monthly payments of £250. Use the Newton Method to find the monthly interest rate correct to 4 signicant figures

The Newton raphson method is used to solve this problem.

​

x = 0.015 *# A variable x is defined as an assumed initial value*

​

*#All user functions are defined*

**def** f(x):

**return** 10000 **\***x **-** 250 **\*** (1**-** 1**/**(1**+**x)**\*\***60)

**def** df(x):

**return** 10000 **-** (15000**/**(1**+**x)**\*\***61)

​

mylist=[] *#A list is defined to store all intermidiate values of x*

**def** newtonraphson(f, df, x, tol= 10**\*\*-**9):

u = (f(x)**/**df(x))

**while** abs(u) **>**=tol:

u = (f(x)**/**df(x))

x = x **-** u

mylist.append(x)

**return** x

root = newtonraphson(f, df, 0.015, tol= 10**\*\*-**9)

print("The root of the equation is {:.4f}" .format(root))

The root of the equation is 0.0144

*#Test for convergence by plotting graph*

c =np.linspace(0,1,len(mylist))

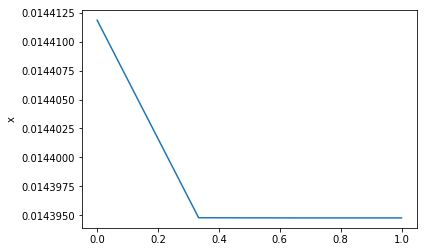
xvals= mylist

ppt.ylabel("x")

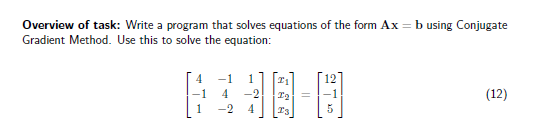
ppt.plot(c, xvals)

Out[9]:

[<matplotlib.lines.Line2D at 0x2aa3e4780b8>]



# Question 2

Write a program that solves equations of the form Ax = b using Conjugate Gradient Method. Use this to solve the eqn.

Step to solve:

1. All user variables are defined and variables A and b are defined using the numpy arrays.
2. Choose initial guess x
3. Define a conjugate gradient variable.
4. A for loop is called

**import** numpy **as** np

A = np.array([[4, **-**1, 1], [**-**1, 4, **-**2], [1, **-**2, 4]]) *#A variable that calls square matrix A is defined.*

b = np.array([12, **-**1, 5]) *#A vector variable b is defined*

x = np.ones(3) *#A matrix with same shape as b that serves as the initial guess with all ones.*

**def** As(s):

**return** np.dot(A,s)

**def** conjGrad(As, x, b, tol = 10**\*\*-**5):

n= len(b)

r = b **-** np.dot(A, x)

s = r.copy()

​

*#conjugate gradient for loop*

**for** k **in** range (0, 3):

numerator = np.dot(s,r)

As = np.dot(A,s)

denominator = np.dot(s, As)

alpha = numerator**/** denominator

x = x **+** alpha **\*** s

r = np.dot(A, x) **-** b

rnorm = np.dot(r, r)

**if** rnorm **<** tol:

**break**

**else**:

numerator = np.dot(r, As)

denominator = np.dot(s, As)

beta = numerator**/** denominator

s = r **+** beta **\*** s

*#Return the requested information*

**return** x, k

solution = conjGrad(As, x, b, tol = 10**\*\*-**5)

print(solution)

(array([ 2.52388666, -0.36654029, 2.36654029]), 2)