Lab Task 1: Plot all the given functions to observe the roots by visualization, fill the table by your visual guess of root. We have plotted one function for you.

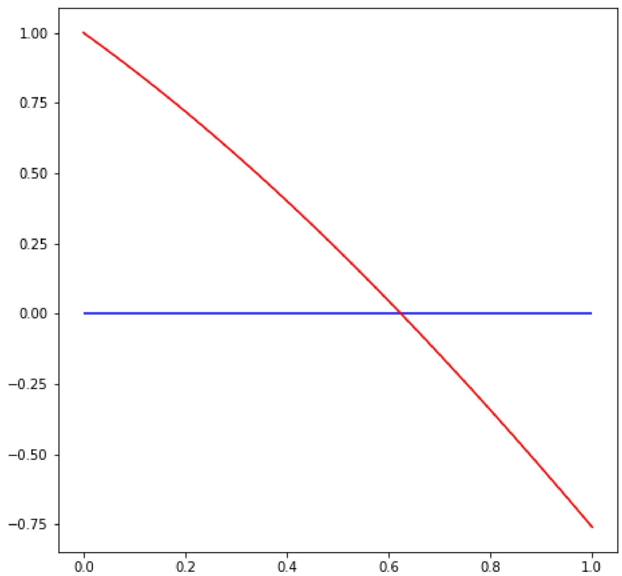
1) $f(x) = cos(x) - 1.3x$
2) $f(x) = xcos(x) - 2x^2 + 3x - 1$
3) $f(x) = 2xcos(2x) - (x+1)^2$

```
import numpy as np
from matplotlib import pyplot as plt

plt.rcParams["figure.figsize"] = [7.50, 7.50]

def f(x):
    return (np.cos(x)-1.3*x)

x = np.linspace(0,1 , 1000)
plt.plot(x,f(x), color='red')
plt.hlines(y=0,xmin=0,xmax=1,color='blue')
plt.show()
```



Lab Task 2: Complete the missing code of bisection method accordding to the explained algorithm and find root of given problems by bisection method according to the instructions given in table.

1) $f1(x) = cos(x) - 1.3x$
2) $f2(x) = xcos(x) - 2x^2 + 3x - 1$
3) $f3(x) = 2xcos(2x) - (x+1)^2$

```
import numpy as np
from tabulate import tabulate

## module Bisection
''' root = bisection(func, x1, x2, tol=0.0001, max_iter=100):.
    Finds a root of f(x) = 0 by bisection.
    The root must be bracketed in (x1,x2).

'''
def func(x):
    return (np.cos(x)-1.3*x)

def bisection(func, x1, x2, tol=0.0001, max_iter=100):
    if func(x1) * func(x2) >= 0:
        return  "Error: Choose different interval, function should have different signs at the interval endpoints."
    data=[]
    iter = 0
    xr = x2
    error = tol + 1
```

```
    while iter < max_iter and error > tol:
        xrold = xr
        xr = ((x1+x2)/2)
        iter += 1
        error = abs((xr - xrold) )

        test = func(x1) * func(xr)
        if  test < 0:
              x2 = xr
        elif test > 0:
              x1 = xr
        else:
              error = 0
        data.append([iter+1,x1,func(x1),x2,func(x2),xr,func(xr),error])
    print(tabulate(data,headers=['#','x1','f(x1)','x2','f(x2)','xr','f(xr)',"error"],tablefmt="github"))
    print('\nRoot of given function is x=%.9f in n=%d number of iterations with a tolerence=%.4f' %(xr,iter,tol))

    return

def f(x):
  return np.cos(x)-1.3*x


bisection(f,0,1)
```

#	x1	f(x1)	x2	f(x2)	xr	f(xr)	error
2	0.5	0.227583	1	-0.759698	0.5	0.227583	0.5
3	0.5	0.227583	0.75	-0.243311	0.75	-0.243311	0.25
4	0.5	0.227583	0.625	-0.00153688	0.625	-0.00153688	0.125
5	0.5625	0.114674	0.625	-0.00153688	0.5625	0.114674	0.0625
6	0.59375	0.0569735	0.625	-0.00153688	0.59375	0.0569735	0.03125
7	0.609375	0.0278184	0.625	-0.00153688	0.609375	0.0278184	0.015625
8	0.617188	0.0131656	0.625	-0.00153688	0.617188	0.0131656	0.0078125
9	0.621094	0.00582059	0.625	-0.00153688	0.621094	0.00582059	0.00390625
10	0.623047	0.0021434	0.625	-0.00153688	0.623047	0.0021434	0.00195312
11	0.624023	0.000303648	0.625	-0.00153688	0.624023	0.000303648	0.000976562
12	0.624023	0.000303648	0.624512	-0.00061652	0.624512	-0.00061652	0.000488281
13	0.624023	0.000303648	0.624268	-0.000156412	0.624268	-0.000156412	0.000244141
14	0.624146	7.36243e-05	0.624268	-0.000156412	0.624146	7.36243e-05	0.00012207
15	0.624146	7.36243e-05	0.624207	-4.13921e-05	0.624207	-4.13921e-05	6.10352e-05

```
Root of given function is x=0.624206543 in n=14 number of iterations with a tolerence=0.0001
```

Lab Task 3: Find root of given problems by Newton Raphson method according to the instructions given in table.

1) $f1(x) = cos(x) - 1.3x$
2) $f2(x) = xcos(x) - 2x^2 + 3x - 1$
3) $f3(x) = 2xcos(2x) - (x+1)^2$

```
import numpy as np
from tabulate import tabulate

## module Newton_Raphson
''' newton_raphson(func, dfunc, x0, tol=1e-4, max_iter=1000)
    Finds a root of f(x) = 0 by newton_raphson.
'''


def newton_raphson(func, dfunc, x0, tol=1e-4, max_iter=1000):
    xr = x0
    data=[]
    iter = 0
    error = tol + 1
    for i in range(max_iter):
        iter+=1
        fx = func(xr)
        dx = dfunc(xr)
        if abs(dx) < tol:
           raise Exception("Derivative is close to zero!")
        xrold=xr
        xr = xr - fx/dx
        error=abs(xr-xrold)
        data.append([iter,xr,func(xr),error])
        if error < tol:
           print(tabulate(data,headers=['Iteration','xr','f(xr)',"error"],tablefmt="github"))
```

```
            print('\nRoot of given function is x=%.9f in n=%d number of iterations with a tolerence=%.4f' %(xr,iter,tol))
            return

    raise Exception("Max iterations reached")


func = lambda x: np.cos(x) - 1.3*x
dfunc = lambda x: -np.sin(x) - 1.3
newton_raphson(func, dfunc, x0=1)
```

```
|   Iteration |       xr |         f(xr) |       error |
|-------------|----------|---------------|-------------|
|           1 | 0.645245 | -0.0398659    | 0.354755    |
|           2 | 0.624278 | -0.000176528  | 0.0209667   |
|           3 | 0.624185 | -3.55988e-09  | 9.36729e-05 |

Root of given function is x=0.624184580 in n=3 number of iterations with a tolerence=0.0001
```

```
func = lambda x: x*np.cos(x) - 2*x**2 + 3*x - 1
dfunc = lambda x: np.cos(x) - x*np.sin(x) - 4*x + 3
newton_raphson(func, dfunc, x0=1)
```

```
|   Iteration |      xr |         f(xr) |       error |
|-------------|---------|---------------|-------------|
|           1 | 1.41524 | -0.540841     | 0.415244    |
|           2 | 1.27672 | -0.0597912    | 0.138528    |
|           3 | 1.25704 | -0.00121677   | 0.0196752   |
|           4 | 1.25662 | -5.47933e-07  | 0.000417389 |
|           5 | 1.25662 | -1.11466e-13  | 1.88126e-07 |

Root of given function is x=1.256623323 in n=5 number of iterations with a tolerence=0.0001
```

```
func = lambda x: 2*x*np.cos(2*x) - (x+1)**2
dfunc = lambda x: 2*np.cos(2*x) - 4*x*np.sin(2*x) - 2*x - 2
newton_raphson(func, dfunc, x0=1)
```

```
|   Iteration |          xr |            f(xr) |         error |
|-------------|-------------|------------------|---------------|
|           1 |  0.429446   |     -1.48222     |  0.570554     |
|           2 | -0.0901012  |     -1.0052      |  0.519548     |
|           3 | 11.9884     |   -159.035       | 12.0785       |
|           4 | 20.4857     |   -502.258       |  8.49735      |
|           5 |  5.8299     |    -39.4599      | 14.6558       |
|           6 | 12.4787     |   -157.1         |  6.64877      |
|           7 |  2.82734    |    -10.0744      |  9.65133      |
|           8 | 19.2814     |   -386.279       | 16.454        |
|           9 | 15.3354     |   -244.302       |  3.94594      |
|          10 | 38.8461     |  -1639.13        | 23.5107       |
|          11 | 30.547      |  -1005.37        |  8.29905      |
|          12 | 48.1657     |  -2464.54        | 17.6187       |
|          13 | 38.9416     |  -1657.03        |  9.22409      |
|          14 | 29.5556     |   -983.119       |  9.38605      |
|          15 | 21.8437     |   -480.033       |  7.71188      |
|          16 | -4.27045    |     -5.27959     | 26.1142       |
|          17 | -4.93579    |     -6.58786     |  0.665339     |
|          18 | -4.48451    |     -4.08827     |  0.451271     |
|          19 | -5.98645    |    -34.7903      |  1.50194      |
|          20 | -4.59608    |     -3.98721     |  1.39037      |
|          21 | -0.640649   |     -0.494906    |  3.95543      |
|          22 | -0.830726   |      0.121759    |  0.190076     |
|          23 | -0.798925   |      0.00279293  |  0.0318004    |
|          24 | -0.79816    |      1.70532e-06 |  0.000765022  |
|          25 | -0.79816    |      6.38052e-13 |  4.67683e-07  |

Root of given function is x=-0.798159961 in n=25 number of iterations with a tolerence=0.0001
```

Lab Task 4: Find root of given problems by using fsolve command of sympy.optimize

1) $f1(x) = cos(x) - 1.3x$
2) $f2(x) = xcos(x) - 2x^2 + 3x - 1$
3) $f3(x) = 2xcos(2x) - (x+1)^2$

```
import sympy as sym

x = sym.symbols('x')
f1 = sym.cos(x) - 1.3*x
root = sym.nsolve(f1, x, 1)
print('Root of given function is x =', root)
```

```
f2 = x*sym.cos(x) - 2*x**2 + 3*x - 1
root = sym.nsolve(f2, x, 1)
print('Root of given function is x =', root)

f3 = 2*x*sym.cos(2*x) - (x+1)**2
root = sym.nsolve(f3, x, 1)
print('Root of given function is x =', root)
```

```
Root of given function is x = 0.624184577804122
Root of given function is x = 1.25662332250557
Root of given function is x = -0.798159961405796
```

Lab Task 5: Write program of Secant and False Position method by altering above codes.

```
import numpy as np
from tabulate import tabulate
## module Secant_method
''' secant_method(func, x0, x1, tol=1e-4, max_iter=1000)
 Finds a root of f(x) = 0 by secant_method.
 '''
def secant_method(func, x0, x1, tol=1e-4, max_iter=1000):
 data=[]
 iter = 0
 error = tol + 1
 while error > tol and iter < max_iter:
   iter += 1
   fx0 = func(x0)
   fx1 = func(x1)
   if abs(fx1 - fx0) < tol:
     raise Exception("Division by zero!")
   x2 = x1 - (fx1*(x1-x0))/(fx1-fx0)
   error = abs(x2 - x1)
   data.append([iter,x0,func(x0),x1,func(x1),x2,func(x2),error])
   x0, x1 = x1, x2

 if iter == max_iter:
   raise Exception("Max iterations reached")

 print(tabulate(data,headers=['Iteration','x0','f(x0)','x1','f(x1)','x2','f(x2)',"error"],tablefmt="github"))
 print('\nRoot of given function is x=%.9f in n=%d number of iterations with a tolerence=%.5f' %(x2,iter,tol))

 def f(x):
   return (np.cos(x)-1.3*x)

secant_method(f, -5, 5, tol=0.00001, max_iter=1000)
```

Iteration	x0	f(x0)	x1	f(x1)	x2	f(x2)	error
1	-5	6.78366	5	-6.21634	0.218202	0.692626	4.7818
2	5	-6.21634	0.218202	0.692626	0.697579	-0.140453	0.479377
3	0.218202	0.692626	0.697579	-0.140453	0.616758	0.0139718	0.0808203
4	0.697579	-0.140453	0.616758	0.0139718	0.624071	0.000214546	0.00731238
5	0.616758	0.0139718	0.624071	0.000214546	0.624185	-3.44219e-07	0.000114037
6	0.624071	0.000214546	0.624185	-3.44219e-07	0.624185	8.43836e-12	1.82669e-07

```
Root of given function is x=0.624184578 in n=6 number of iterations with a tolerence=0.00001
```

```
def false_position_method(func, a, b, tol=1e-4, max_iter=1000):
    data=[]
    iter = 0
    error = tol + 1
    while error > tol and iter < max_iter:
        iter+=1
        fa = func(a)
        fb = func(b)
        x = a - (fa*(b-a))/(fb-fa)
        fx = func(x)
        if fa * fx > 0:
            a = x
        else:
            b = x
        error = abs(fx)
        data.append([iter,x,func(x),error])
    print(tabulate(data,headers=['Iteration','x','f(x)',"error"],tablefmt="github"))
```

```
    print(tabulate(data,headers=['iteration','x','f(x)','error'],tablefmt='github'))
    print('\nRoot of given function is x=%.9f in n=%d number of iterations with a tolerence=%.4f' %(x,iter,tol))

a = 1
b = 2
false_position_method(f1, a, b)

a = 1
b = 2
false_position_method(f2, a, b)

a = 1
b = 2
false_position_method(f3, a, b)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-720286584f38> in <module>
     20 a = 1
     21 b = 2
---> 22 false_position_method(f1, a, b)
     23
     24 a = 1

NameError: name 'f1' is not defined
```

SEARCH STACK OVERFLOW