Lab Assignment 1

Student Name: Ahmed Yoshay

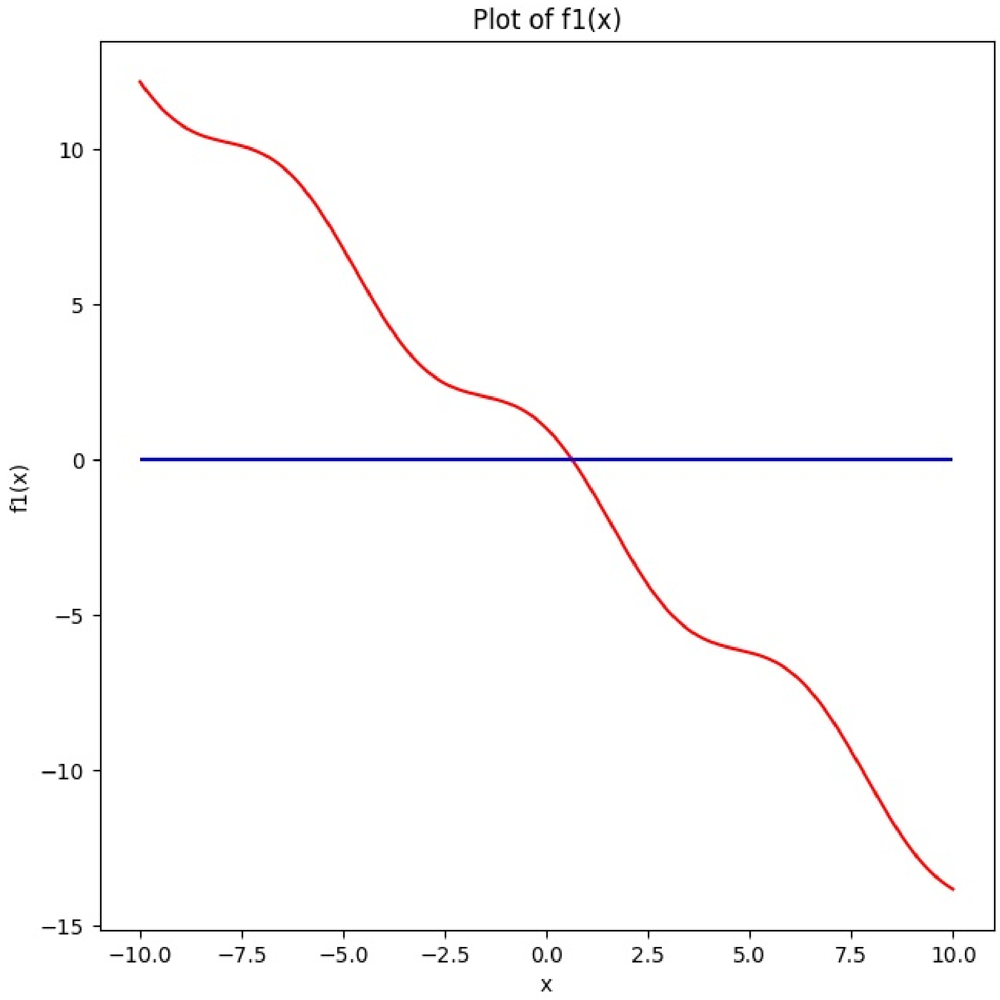
Section: BCS-4J

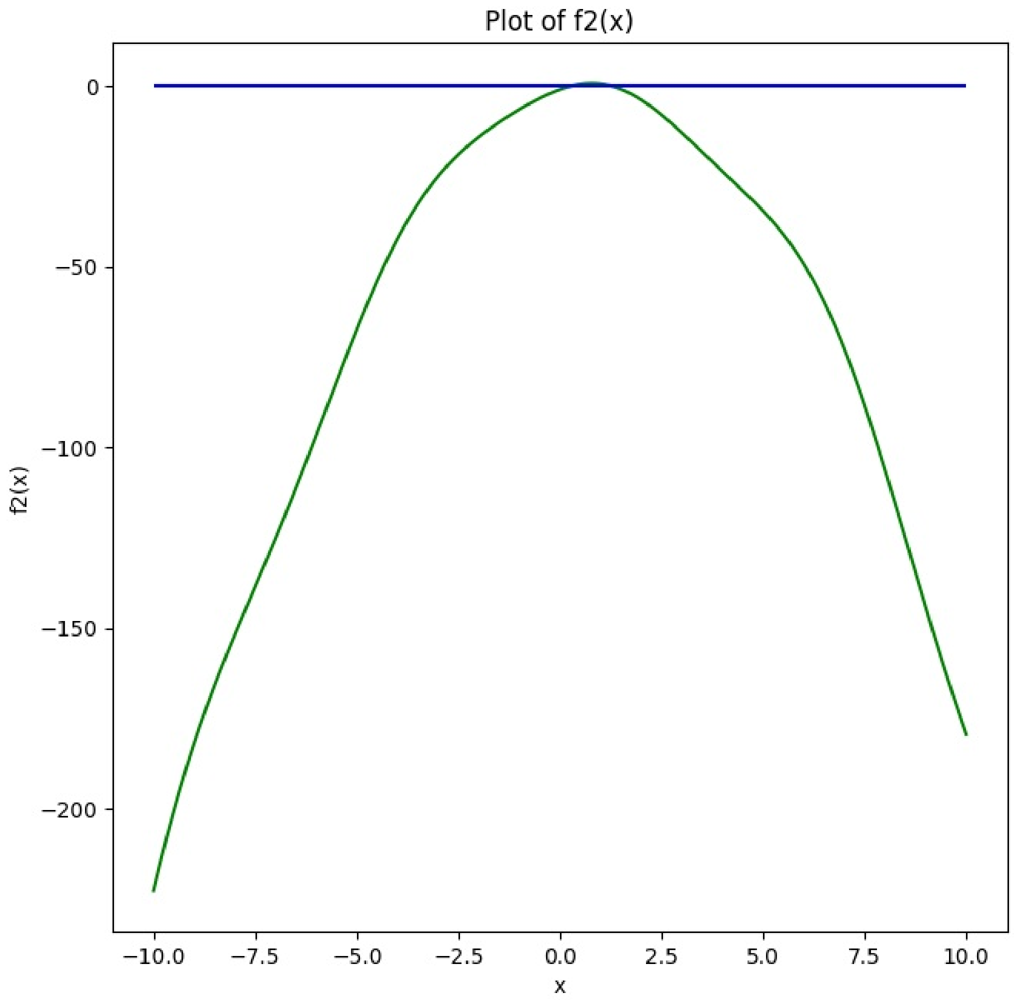
Lab Task 1: Plot all the given functions to observe the roots by visualization, fill the table by your visual guess of root. We have plotted one function for you.

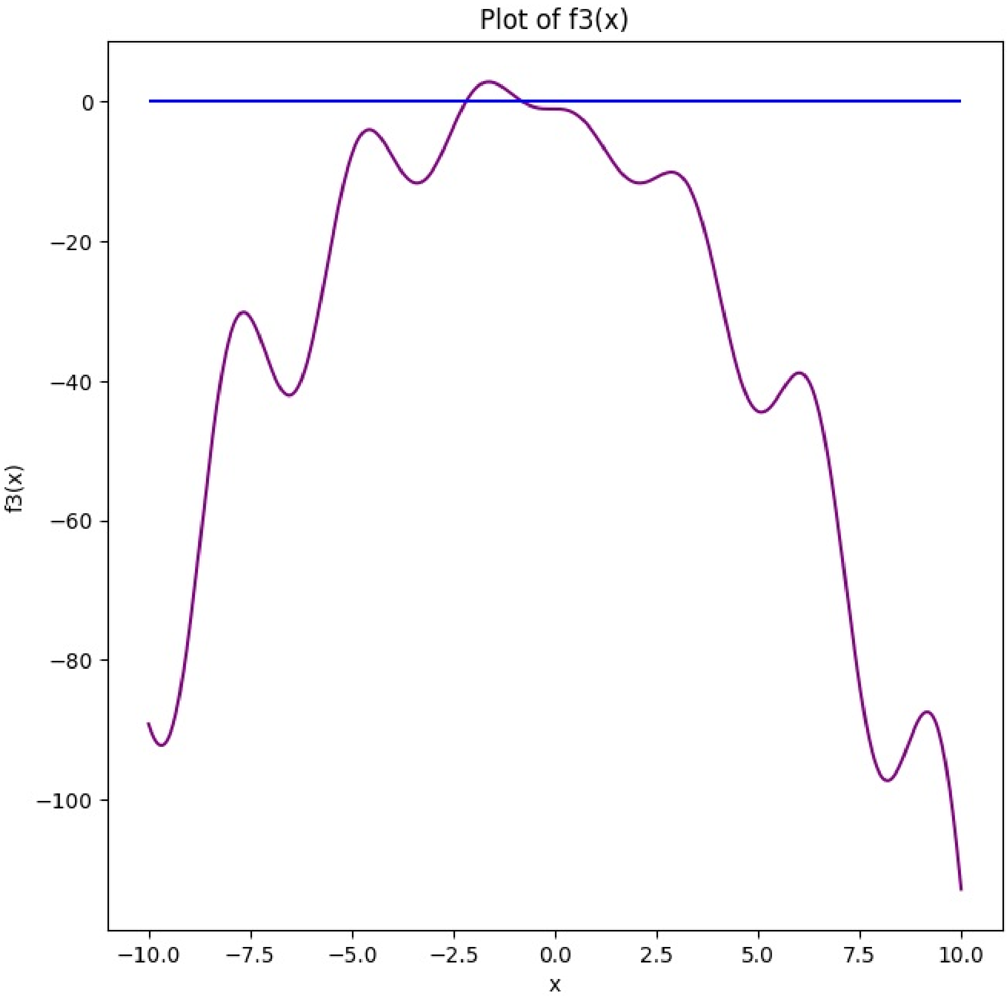
1. $f_1(x) = \cos(x) - 1.3x$
2. $f_2(x) = x\cos(x) - 2x^2 + 3x - 1$
3. $f_3(x) = 2x\cos(2x) - (x+1)^2$

In [2]:

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


def f1(x):
    return np.cos(x) - 1.3 * x

def f2(x):
    return x * np.cos(x) - 2 * x**2 + 3 * x - 1

def f3(x):
    return 2 * x * np.cos(2 * x) - (x + 1)**2


x = np.linspace(-10, 10, 1000)

# Plot f1(x) and the horizontal line
plt.plot(x, f1(x), color='red')
plt.hlines(y=0, xmin=-10, xmax=10, color='blue')
plt.title('Plot of f1(x)')
plt.xlabel('x')
plt.ylabel('f1(x)')
plt.show()

# Plot f2(x) and the horizontal line
plt.plot(x, f2(x), color='green')
plt.hlines(y=0, xmin=-10, xmax=10, color='blue')
plt.title('Plot of f2(x)')
plt.xlabel('x')
plt.ylabel('f2(x)')
plt.show()

# Plot f3(x) and the horizontal line
plt.plot(x, f3(x), color='purple')
plt.hlines(y=0, xmin=-10, xmax=10, color='blue')
plt.title('Plot of f3(x)')
plt.xlabel('x')
plt.ylabel('f3(x)')
plt.show()
```

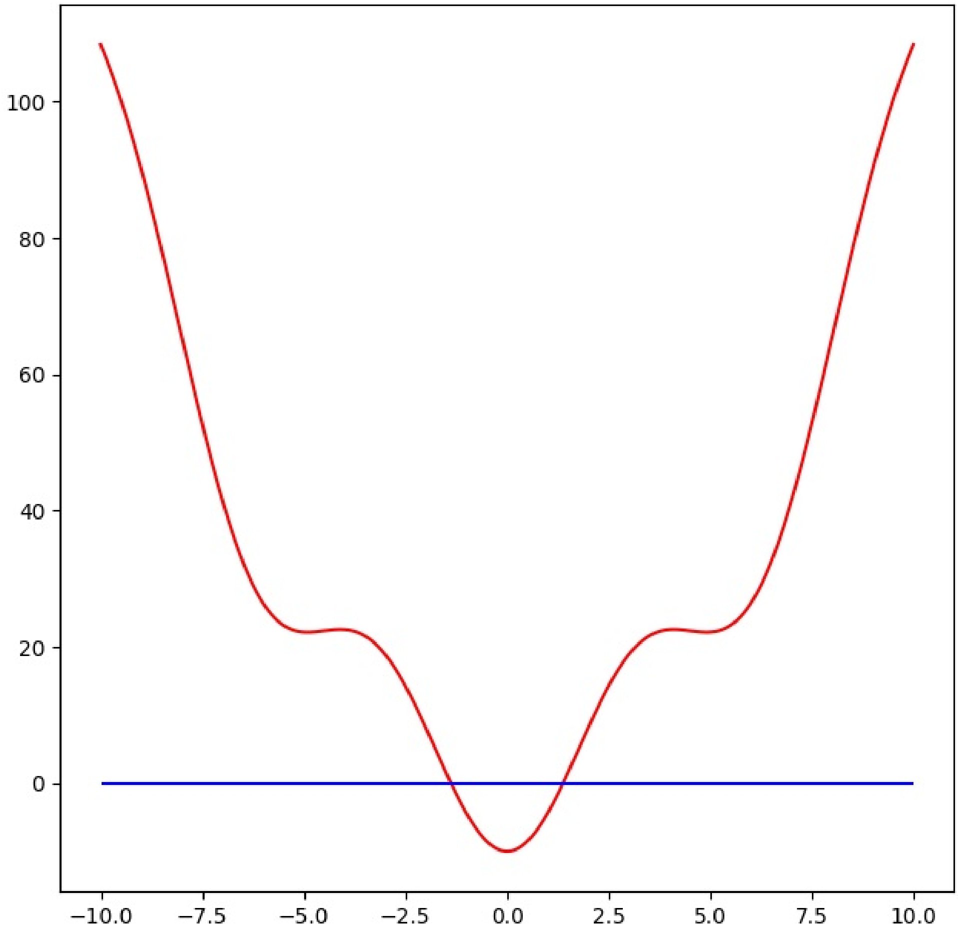

Plot of f1(x)
10
5
0
−5
−10
−15
f1(x)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x


Plot of f2(x)
0
−50
−100
−150
−200
f2(x)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x



In [1]:

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

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

def f(x):
    return (x**2-10*np.cos(x))

x = np.linspace(-10,10 , 1000)
plt.plot(x,f(x), color='red')
plt.hlines(y=0,xmin=-10,xmax=10,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. $f_1(x) = \cos(x) - 1.3x$
2. $f_2(x) = x\cos(x) - 2x^2 + 3x - 1$
3. $f_3(x) = 2x\cos(2x) - (x+1)^2$

In [3]:
```python
import numpy as np
from tabulate import tabulate


''' 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 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 endpoin
    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, 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, 
```

```python
    return

def f1(x):
    return np.cos(x) - 1.3 * x

def f2(x):
    return x * np.cos(x) - 2 * x**2 + 3 * x - 1

def f3(x):
    return 2 * x * np.cos(2 * x) - (x + 1)**2


print("Roots of f1(x):")
bisection(f1, 0, 1)
print("\nRoots of f2(x):")
bisection(f2, 0, 1)
print("\nRoots of f3(x):")
bisection(f3, -2, 1)
```

Roots of f1(x):

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

Roots of f2(x):

#	x1	f(x1)	x2	f(x2)	xr	f(xr)	error
1	0	-1	0.5	0.438791	0.5	0.438791	0.5
2	0.25	-0.132772	0.5	0.438791	0.25	-0.132772	0.25
3	0.25	-0.132772	0.375	0.19269	0.375	0.19269	0.125
4	0.25	-0.132772	0.3125	0.0395525	0.3125	0.0395525	0.0625
5	0.28125	-0.0442537	0.3125	0.0395525	0.28125	-0.0442537	0.03125
6	0.296875	-0.00175623	0.3125	0.0395525	0.296875	-0.00175623	0.015625
7	0.296875	-0.00175623	0.304688	0.0190474	0.304688	0.0190474	0.0078125
8	0.296875	-0.00175623	0.300781	0.0086828	0.300781	0.0086828	0.00390625
9	0.296875	-0.00175623	0.298828	0.00347258	0.298828	0.00347258	0.00195312
10	0.296875	-0.00175623	0.297852	0.000860498	0.297852	0.000860498	0.000976562
11	0.297363	-0.000447286	0.297852	0.000860498	0.297363	-0.000447286	0.000488281
12	0.297363	-0.000447286	0.297607	0.000206751	0.297607	0.000206751	0.000244141
13	0.297485	-0.000120231	0.297607	0.000206751	0.297485	-0.000120231	0.00012207
14	0.297485	-0.000120231	0.297546	4.32688e-05	0.297546	4.32688e-05	6.10352e-05

Root of given function is x=0.297546387 in n=14 number of iterations with a tolerence=0.0001

Roots of f3(x):

#	x1	f(x1)	x2	f(x2)	xr	f(xr)	error
1	-2	1.61457	-0.5	-0.790302	-0.5	-0.790302	1.5
2	-1.25	1.94036	-0.5	-0.790302	-1.25	1.94036	0.75
3	-0.875	0.296306	-0.5	-0.790302	-0.875	0.296306	0.375
4	-0.875	0.296306	-0.6875	-0.365159	-0.6875	-0.365159	0.1875
5	-0.875	0.296306	-0.78125	-0.0608144	-0.78125	-0.0608144	0.09375
6	-0.828125	0.111819	-0.78125	-0.0608144	-0.828125	0.111819	0.046875
7	-0.804688	0.0239252	-0.78125	-0.0608144	-0.804688	0.0239252	0.0234375
8	-0.804688	0.0239252	-0.792969	-0.0188499	-0.792969	-0.0188499	0.0117188
9	-0.798828	0.00243764	-0.792969	-0.0188499	-0.798828	0.00243764	0.00585938
10	-0.798828	0.00243764	-0.795898	-0.0082313	-0.795898	-0.0082313	0.00292969
11	-0.798828	0.00243764	-0.797363	-0.0029031	-0.797363	-0.0029031	0.00146484
12	-0.798828	0.00243764	-0.798096	-0.000234294	-0.798096	-0.000234294	0.000732422
13	-0.798462	0.00110128	-0.798096	-0.000234294	-0.798462	0.00110128	0.000366211
14	-0.798279	0.000433396	-0.798096	-0.000234294	-0.798279	0.000433396	0.000183105
15	-0.798187	9.95265e-05	-0.798096	-0.000234294	-0.798187	9.95265e-05	9.15527e-05

Root of given function is x=-0.798187256 in n=15 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. $f_1(x) = \cos(x) - 1.3x$

2. $f_2(x) = x\cos(x) - 2x^2 + 3x - 1$
3. $f_3(x) = 2x\cos(2x) - (x + 1)^2$

In [4]:
```
import numpy as np
from tabulate import tabulate

''' 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 tolerance=%.4f' %(xr,i
           return

    raise Exception("Max iterations reached")

# Define the functions and their derivatives
def f1(x):
    return np.cos(x) - 1.3 * x

def df1(x):
    return -np.sin(x) - 1.3

def f2(x):
    return x * np.cos(x) - 2 * x**2 + 3 * x - 1

def df2(x):
    return -x * np.sin(x) + np.cos(x) - 4 * x + 3

def f3(x):
    return 2 * x * np.cos(2 * x) - (x + 1)**2

def df3(x):
    return 2 * np.cos(2 * x) - 4 * x * np.sin(2 * x) - 2 * (x + 1)

print("\nRoots of f1(x):")
try:
    newton_raphson(f1, df1, 0.5)
except Exception as e:
    print(e)

print("\nRoots of f2(x):")
try:
    newton_raphson(f2, df2, 0.5)
except Exception as e:
    print(e)

print("\nRoots of f3(x):")
try:
    newton_raphson(f3, df3, -1)
except Exception as e:
    print(e)
```

```
Roots of f1(x):
|   Iteration |       xr |        f(xr) |       error |
|-------------|----------|--------------|-------------|
|           1 | 0.627897 | -0.00700074  | 0.127897    |
|           2 | 0.624188 | -5.57173e-06 | 0.00370911  |
|           3 | 0.624185 | -3.54672e-12 | 2.95671e-06 |

Root of given function is x=0.624184578 in n=3 number of iterations with a tolerance=0.0001

Roots of f2(x):
|   Iteration |       xr |        f(xr) |       error |
|-------------|----------|--------------|-------------|
|           1 | 0.232096 | -0.185575    | 0.267904    |
|           2 | 0.294132 | -0.00913008  | 0.062036    |
|           3 | 0.29752  | -2.79097e-05 | 0.00338747  |
|           4 | 0.29753  | -2.64366e-10 | 1.04188e-05 |

Root of given function is x=0.297530234 in n=4 number of iterations with a tolerance=0.0001

Roots of f3(x):
|   Iteration |        xr |        f(xr) |       error |
|-------------|-----------|--------------|-------------|
|           1 | -0.813783 | 0.0576701    | 0.186217    |
|           2 | -0.798346 | 0.000678024  | 0.0154371   |
|           3 | -0.79816  | 1.00778e-07  | 0.000185892 |
|           4 | -0.79816  | 2.07473e-15  | 2.76384e-08 |

Root of given function is x=-0.798159961 in n=4 number of iterations with a tolerance=0.0001
```

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

1. $f_1(x) = \cos(x) - 1.3x$
2. $f_2(x) = x\cos(x) - 2x^2 + 3x - 1$
3. $f_3(x) = 2x\cos(2x) - (x + 1)^2$

In [5]:
```python
import numpy as np
from scipy.optimize import fsolve
from sympy import symbols

x = symbols('x')

def f1(x):
    return np.cos(x) - 1.3*x

def f2(x):
    return x*np.cos(x) - 2*x**2 + 3*x - 1

def f3(x):
    return 2*x*np.cos(2*x) - (x + 1)**2

# Find roots using fsolve
root_f1 = fsolve(f1, 0)
root_f2 = fsolve(f2, 0)
root_f3 = fsolve(f3, -0.5)

# Print the roots
print("Roots using fsolve:")
print("Root of f1(x): ",root_f1[0])
print("Root of f2(x): ",root_f2[0])
print("Root of f3(x) :",root_f3[0])
```

```
Roots using fsolve:
Root of f1(x):  0.6241845778041218
Root of f2(x):  0.2975302336716408
Root of f3(x) : -0.7981599614057959
```

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

In [6]:
```python
import numpy as np
from sympy import symbols, cos
from scipy.optimize import fsolve
from sympy.utilities.lambdify import lambdify
from tabulate import tabulate

# Define the symbol
x = symbols('x')

# Define the function
f1 = x**2 - 4

def secant_method(func, x0, x1, tol=1e-6, max_iter=100):
    # Convert the symbolic expression to a numeric function
```

```
    func_numeric = lambdify(x, func, 'numpy')

    data = []
    iter = 0
    error = tol + 1

    while iter < max_iter and error > tol:
        iter += 1
        f_x0 = func_numeric(x0)
        f_x1 = func_numeric(x1)

        if f_x1 - f_x0 == 0:
            raise Exception("Secant method: Division by zero.")

        x2 = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)

        f_x2 = func_numeric(x2)

        x0, x1 = x1, x2

        error = abs(x2 - x1)

        data.append([iter, x0, f_x0, x1, f_x1, x2, f_x2, error])

    print(tabulate(data, headers=['Iteration', 'x0', 'f(x0)', 'x1', 'f(x1)', 'x2', 'f(x2)', 'Error'], tablefmt=
    print('\nRoot of given function is x=%.9f in n=%d number of iterations with a tolerance=%.6f' % (x2, iter, 
    return x2

print("\nSecant Method:")
secant_method(f1, 0, 2)


def false_position_method(func, x0, x1, tol=1e-6, max_iter=100):
    func_numeric = lambdify(x, func, 'numpy')

    data = []
    iter = 0
    error = tol + 1

    while iter < max_iter and error > tol:
        iter += 1
        f_x0 = func_numeric(x0)
        f_x1 = func_numeric(x1)

        if f_x1 - f_x0 == 0:
            raise Exception("False Position method: Division by zero.")

        x2 = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)

        f_x2 = func_numeric(x2)

        if f_x0 * f_x2 < 0:
            x1 = x2
        else:
            x0 = x2

        error = abs(x2 - x1)

        data.append([iter, x0, f_x0, x1, f_x1, x2, f_x2, error])

    print(tabulate(data, headers=['Iteration', 'x0', 'f(x0)', 'x1', 'f(x1)', 'x2', 'f(x2)', 'Error'], tablefmt=
    print('\nRoot of given function is x=%.9f in n=%d number of iterations with a tolerance=%.6f' % (x2, iter, 
    return x2

print("\nFalse Position Method:")
false_position_method(f1, 0, 2)
```

```
Secant Method:
|   Iteration |   x0 |   f(x0) |   x1 |   f(x1) |   x2 |   f(x2) |   Error |
|-------------|------|---------|------|---------|------|---------|---------|
|           1 |    2 |      -4 |    2 |       0 |    2 |       0 |       0 |

Root of given function is x=2.000000000 in n=1 number of iterations with a tolerance=0.000001

False Position Method:
|   Iteration |   x0 |   f(x0) |   x1 |   f(x1) |   x2 |   f(x2) |   Error |
|-------------|------|---------|------|---------|------|---------|---------|
|           1 |    2 |      -4 |    2 |       0 |    2 |       0 |       0 |

Root of given function is x=2.000000000 in n=1 number of iterations with a tolerance=0.000001
```

Out[6]: 2.0

Processing math: 100%