LAB 1: Revision

Date: 19-12-2023

In [3]:
```
import numpy as np
```

WAP to solve the following system of linear equations:

$$2x - y + 3z = 8$$
$$x + 2y + z = 7$$
$$3x + y - 2x = 9$$

In [7]:
```
A = np.matrix([[2,-1,8],
               [1,2,1],
               [3,1,-2]])
B = np.matrix([8,7,9]).T

print(A.I * B)
```

```
[[2.74545455]
 [1.85454545]
 [0.54545455]]
```

WAP to find the roots of the given quadratic equation:

$$x^2 + x + 1 = 0$$

In [25]:
```
P = np.array([1,1,1])
print(np.roots(P))
```

```
[-0.5+0.8660254j -0.5-0.8660254j]
```

WAP to display 10 values between 0 and 10 including 10.

In [18]:
```
np.linspace(0,10,10)
```

Out[18]:
```
array([ 0.        ,  1.11111111,  2.22222222,  3.33333333,  4.44444444,
        5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.        ])
```

WAP to display 10 values between 0 and 10 excluding 10.

In [19]:
```
np.arange(0,10,1)
```

Out[19]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

WAP to add and multiply two matrices.

In [21]:
```
A = np.matrix(eval(input("Enter First Matrix:")))
B = np.matrix(eval(input("Enter Second Matrix:")))

print("SUM = ")
print(A+B)
print("PRODUCT = ")
print(A*B)
```

```
Enter First Matrix:[[1,4],[5,9]]
Enter Second Matrix:[[-2,4],[3,6]]
SUM =
[[-1  8]
 [ 8 15]]
PRODUCT =
[[10 28]
 [17 74]]
```

WAP to find the invese of a matrix

In [22]:
```
A = np.matrix(eval(input("Enter First Matrix:")))
print("Invese = ")
print(A.I)
```

```
Enter First Matrix:[[1,4],[5,9]]
Invese =
[[-0.81818182  0.36363636]
 [ 0.45454545 -0.09090909]]
```

LAB 2: Algebraic and Transcedental Equations

Date: 02-01-2024

In [1]:
```
import numpy as np
from scipy.misc import derivative
import scipy.optimize as opt
```

BISECTION METHOD

In [2]:
```python
def bisect(f:'function',a = -1,b = 1):
    """
        Bisect the interval with respect to given function

        Parameters
        ----------
        a : int
            Lower Limit
        b : int
            Upper Limit
        f : function
            Function to be evaluated

        Returns
        -------
        New interval after single bisection.
        """
    if(f(a)>f(b)):
        a,b = b,a;
    if(f(a)*f(b)>0):
        raise Exception("INVALID INTERVAL",(a,b))
    mid = (a+b)/2
    a,b = [mid,b] if f(mid)<0 else [a,mid]
    return [a,b]

def bisectionMethod(a,b,f:'function' = lambda x:0,error = 1e-10):
    """
        Find roots using Bisection Method

        Parameters
        ----------
        a : int
            Lower Limit
        b : int
            Upper Limit
        error : float
            Approximation required
        f : function
            Function to be evaluated

        Returns
        -------
        Approximate root, number of iterations
        """
    n = 0;
    while((abs(a-b) > error) and (n<100)):
        a,b = bisect(f,a,b)
        n += 1;
    return [(a+b)/2,n]
```

In [3]:
```
f = lambda x: x**3 - 5*x - 9
bisectionMethod(2,3,f,1e-7)
```

Out[3]:
```
[2.855196565389633, 24]
```

In [4]:
```
poly = [1,0,-5,-9]
np.roots(poly)
```

Out[4]:
```
array([ 2.85519654+0.j        , -1.42759827+1.05551431j,
       -1.42759827-1.05551431j])
```

WAP to find the roots of the following equations

- $f(x) = x^3 - x - 1$
- $f(x) = x^2 - 3x - 1$
- $f(x) = x^2 - 3e^x$

In [5]:
```
f = lambda x: x**3 - x - 1
bisectionMethod(1,2,f)
```

Out[5]:
```
[1.324717957264511, 34]
```

In [6]:
```
f = lambda x: x**2 - 3*np.exp(x)
bisectionMethod(0,-2,f)
```

Out[6]:
```
[-1.0332309037039522, 35]
```

REGULA FALSI METHOD or METHOD OF FALSE POSITION

In [7]:
```
def regula(f:'function',a = -1,b = 1):
    if(a>b):
        a,b = b,a;
    if(f(a)*f(b) > 0):
        raise Exception("INVALID INTERVAL",(a,b))
    return [b,b - ((b-a)*f(b)/(f(b)-f(a)))]
def regulaFalsiMethod(a,b,f:'function' = lambda x:0,error = 1e-10):
    n = 0;
    while((abs(f(b)) > error) and (n<100)):
        a,b = regula(f,a,b)
        n += 1;
    return [b,n]
```

In [8]:
```
f = lambda x: x**3 - 2*x - 5
regulaFalsiMethod(-1,5,f)
```

Out[8]:
```
[2.094551481535422, 85]
```

In [9]:
```python
np.roots([1,0,-2,5])
```

Out[9]:
```
array([-2.09455148+0.j        ,  1.04727574+1.13593989j,
        1.04727574-1.13593989j])
```

WAP to find the roots of the following equations

- $f(x) = x^4 - 3$
- $f(x) = 2\cos x - x$
- $f(x) = xe^x - 1$

In [10]:
```python
f = lambda x: x**4 - 3
regulaFalsiMethod(-1,2,f)
```

Out[10]:
```
[1.3160740129466892, 44]
```

In [11]:
```python
f = lambda x: 2*np.cos(x) - x
regulaFalsiMethod(-2,2,f)
```

Out[11]:
```
[1.0298665293179292, 12]
```

In [12]:
```python
f = lambda x: x*np.exp(x) - 1
regulaFalsiMethod(-2,2,f)
```

Out[12]:
```
[0.5671432903759988, 78]
```

FIXED POINT METHOD

In [13]:
```python
def fixedPointMethod(a,f:'function' = lambda x:0, g:'function' = lambda x:
0, error = 1e-10,thresh = 500):
    n = 0;
    while((abs(f(a)) > error) and (n<thresh)):
        a = g(a)
        n += 1;
    return [a,False if n == thresh else True,n]
```

In [14]:
```python
f = lambda x: x**3 + x**2 - 2
g = lambda x: (2-x**2)**(1/3)
fixedPointMethod(2,f,g)
```

Out[14]:
```
[(1.000000000012568-1.254483651600653e-11j), True, 62]
```

In [15]:
```python
opt.root(f,2)
```

Out[15]:
```
    fjac: array([[-1.]])
     fun: array([0.])
 message: 'The solution converged.'
    nfev: 10
     qtf: array([-2.91384694e-10])
       r: array([-5.00000208])
  status: 1
 success: True
       x: array([1.])
```

WAP to find the roots of the following equations

- $f(x) = x^3 - 2x - 5$
- $f(x) = 2x - 3 - \cos x$
- $f(x) = \sin x - 10(x - 1)$

In [16]:
```python
f = lambda x: x**3 - 2*x - 5
g = lambda x: (5+2*x)**(1/3)
fixedPointMethod(2,f,g),opt.root(f,2).x
```

Out[16]:
```
([2.0945514815401305, True, 13], array([2.09455148]))
```

In [17]:
```python
f = lambda x: 2*x - 3 - np.cos(x)
g = lambda x: (np.cos(x) + 3)/2
fixedPointMethod(2,f,g),opt.root(f,2).x
```

Out[17]:
```
([1.5235929331230837, True, 34], array([1.52359293]))
```

In [18]:
```python
f = lambda x: np.sin(x) - 10*(x-1)
g = lambda x: np.sin(x)/10 + 1
fixedPointMethod(2,f,g),opt.root(f,2).x
```

Out[18]:
```
([1.0885977523989665, True, 8], array([1.08859775]))
```

NEWTON RAPHSON METHOD

In [19]:
```python
def getRaphson(a,f,error):
    if(derivative(f,a,dx=error) == 0):
        raise Exception("DERIVATIVE IS ZERO",(a))
    return a-(f(a)/derivative(f,a,error))

def newtonRaphsonMethod(a,f:'function' = lambda x:0, error = 1e-10,thresh = 
500):
    n = 0;
    while((abs(f(a)) > error) and (n<thresh)):
        a = getRaphson(a,f,error)
        n += 1
    return [a,False if n == thresh else True,n]
```

In [20]:
```
f = lambda x: x**3 + 2*x**2 + x - 1
newtonRaphsonMethod(0,f),opt.root(f,0)
```

Out[20]:
```
([0.4655712318767954, True, 6],
      fjac: array([[-1.]])
       fun: array([-9.88098492e-15])
   message: 'The solution converged.'
      nfev: 11
       qtf: array([-3.42745543e-09])
         r: array([-3.51254448])
    status: 1
   success: True
         x: array([0.46557123]))
```

WAP to find the roots of the following equations

- $f(x) = x^3 - 5x^2$
- $f(x) = x^3 - 3x - 1$
- $f(x) = x^3 + 2x^2 + x - 1$
- $f(x) = 3cos(x) + x$

In [29]:
```
f = lambda x: x**3 - 5*x**2
newtonRaphsonMethod(6,f),opt.root(f,6).x
```

Out[29]:
```
([5.000000000000115, True, 5], array([5.]))
```

In [30]:
```
f = lambda x: x**3 - 3*x - 1
newtonRaphsonMethod(2,f),opt.root(f,2).x
```

Out[30]:
```
([1.8793852415718182, True, 4], array([1.87938524]))
```

In [32]:
```
f = lambda x: x**3 + 2*x**2 + x - 1
newtonRaphsonMethod(0,f),opt.root(f,0).x
```

Out[32]:
```
([0.4655712318767954, True, 6], array([0.46557123]))
```

In [35]:
```
f = lambda x: 3*np.cos(x) + x
newtonRaphsonMethod(1,f),opt.root(f,1).x
```

Out[35]:
```
([2.663178883323163, True, 5], array([2.66317888]))
```

LAB 3: System of Equation

Date: 13-01-2024

In [1]:
```
import numpy as np
```

WAP to find the inverse of the following matrix and solve the system of equations.

In [2]:
```
A = np.matrix([[4,-2,1],[-2,4,-2],[1,-2,4]])
A
```

Out[2]:
```
matrix([[ 4, -2,  1],
        [-2,  4, -2],
        [ 1, -2,  4]])
```

In [3]:
```
B = np.matrix([11,-16,17]).T
B
```

Out[3]:
```
matrix([[ 11],
        [-16],
        [ 17]])
```

In [4]:
```
#Inverse
A.I
```

Out[4]:
```
matrix([[0.33333333, 0.16666667, 0.        ],
        [0.16666667, 0.41666667, 0.16666667],
        [0.        , 0.16666667, 0.33333333]])
```

In [5]:
```
#Solution
np.linalg.solve(A,B)
```

Out[5]:
```
matrix([[ 1.],
        [-2.],
        [ 3.]])
```

In [6]:
```
#Solution
A.I @ B
```

Out[6]:
```
matrix([[ 1.],
        [-2.],
        [ 3.]])
```

WAP to print the 2^{nd} row of the given matrix.

In [7]:
```
A[1]
```

Out[7]:
```
matrix([[-2,  4, -2]])
```

WAP to print col[4 - 2]

In [19]:
```
A
A[0,0:2].T
```

Out[19]:
```
matrix([[ 4],
        [-2]])
```

CRAMERS RULE

$$X = \frac{A\ with\ X\ replaced\ with\ B}{|A|}$$

In [23]:
```
def cramersRule(A,B):
    X = np.ones(len(A))
    for i in range(len(A)):
        X[i] = A.T[0:i];
```