# 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]:
```
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)
```

```python
        n += 1;
    return [(a+b)/2,n]
```

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

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

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

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

In [6]:
```python
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]:
```python
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]:
```python
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]:
```
f = lambda x: x**4 - 3
regulaFalsiMethod(-1,2,f)
```

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

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

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

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

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

### FIXED POINT METHOD

In [13]:
```
def fixedPointMethod(a,f:'function' = lambda x:0, g:'function' = lambda x:0, error
    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]:
```
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]:
```
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]:
```
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]:
```
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]:
```
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]:
```
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 [3]:
```
import numpy as np
from typing import *
```

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

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

Out[4]:
```
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 [8]:
```
A
A[0,0:2].T
```

Out[8]:
```
matrix([[ 4],
        [-2]])
```

**Date: 16-01-24**

> **CRAMERS RULE**
>
> $$X = \frac{A\ with\ X\ replaced\ with\ B}{|A|}$$

In [4]:
```
def cramersRule(A,B):
    X = [0]*len(A)
    modA = np.linalg.det(A)
    A = A.T
    for i in range(len(A)):
        X[i] = np.round(np.linalg.det((np.append(np.append(A[0:i],B,axis=0),A[i+1:]
    return X
```

In [5]:
```
A = np.matrix([[4,3,-2],[3,-7,5],[1,3,-2]])
B = np.matrix([1,2,7])
```

In [6]:
```
A,B
```

Out[6]:
```
(matrix([[ 4,  3, -2],
         [ 3, -7,  5],
         [ 1,  3, -2]]),
 matrix([[1, 2, 7]]))
```

In [7]:
```
cramersRule(A,B)
```

Out[7]:
```
[-2.0, 61.0, 87.0]
```

In [8]:
```
np.linalg.solve(A,B.T)
```

Out[8]:
```
matrix([[-2.],
        [61.],
        [87.]])
```

**Date: 20-01-24**

> **Gauss Elemination Method**

In [9]:
```
def rowEchelon(A):
    A = np.matrix(A,dtype=float)
```

```
    mat = A.copy()
    for i in range(min(mat.shape)):
        row = mat[i,:].A1
        if(row[i] == 0):
            continue;
        row = row/row[i];
        mat[i,:] = row
        for j in range(i+1,mat.shape[0]):
            mat[j,:] = mat[j,:] - (mat[j,i]*row)
    return mat

def gaussMethod(A,B):
    mat = np.insert(arr=A,obj=A.shape[1],values=B.A1,axis=1)
    mat = rowEchelon(mat)
    #print(mat)
    A_ = mat[:,:-1].A #Get Without Last Columns
    B_ = mat[:,-1:] #Get Last Column
    X = np.matrix(np.zeros(B.shape),float)
    varSolved = []
    solving = [list(i).count(0) for i in A_]
    for i in range(A_.shape[1])[::-1]:
        rowIndex = solving.index(i)
        row = A_[rowIndex,:]
        for j in range(A_.shape[1]):
            if((row[j] != 0) and (j not in varSolved)):
                varSolved.append(j)
                break;
        X[varSolved[-1],0] = (B_[rowIndex,0] - np.sum([row[k]*X[k,0] for k in range
    #X = np.linalg.solve(mat.T[:-1,:].T,mat.T[-1,:].T)
    return X
```

> $2x + 5y = 7$ </br> $4x + 7y = 14$ </br>

In [10]:
```
A = np.matrix([[2,5],[4,7]])
B = np.matrix([7,14]).T
gaussMethod(A,B)
```

Out[10]:
```
matrix([[ 3.5],
        [-0. ]])
```

In [11]:
```
np.linalg.solve(A,B)
```

Out[11]:
```
matrix([[3.5],
        [0. ]])
```

> $2x + y + z = 10$</br> $5x - 9y + 3z = 5$</br> $x + 4y + 9z = 16$</br>

In [12]:
```
A = np.matrix([[2,1,1],[3,5,3],[1,4,9]])
B = np.matrix([10,18,16]).T
A,B,dict(GE = gaussMethod(A,B),Inverse = A.I * B)
```

Out[12]:
```
(matrix([[2, 1, 1],
         [3, 5, 3],
         [1, 4, 9]]),
 matrix([[10],
         [18],
         [16]]),
 {'GE': matrix([[4.24489796],
          [0.36734694],
          [1.14285714]]),
  'Inverse': matrix([[4.24489796],
          [0.36734694],
          [1.14285714]])})
```

In [86]:
```
A = np.matrix([[12,-8,5],[-9,5,7],[2,13,8]])
B = np.matrix([24,-4,15]).T
gaussMethod(A,B)
```

Out[86]:
```
matrix([[1.52713178],
        [0.11111111],
        [1.3126615 ]])
```

In [87]:
```
A = np.matrix([[4,3,2],[2,3,4],[1,2,1]])
B = np.matrix([16,20,8]).T
gaussMethod(A,B)
```

Out[87]:
```
matrix([[1.],
        [2.],
        [3.]])
```

**Date: 23-01-24**

In [15]:
```
A = np.matrix([[2,1,1],[1,3,1],[1,2,3]])
B = np.matrix([7,13,13]).T
A.I*B - np.matrix([0.8774,3.422953,1.758897]).T
```

Out[15]:
```
matrix([[ 0.03169091],
        [ 0.03159245],
        [-0.03162427]])
```

> ### Iterative Methods

In [16]:
```
import sympy as sp
```

In [22]:
```
def stringToEquation(VARS: str,EQ: str):

    #Making Sympy Variables
    X_vars = VARS.split(",")
    X_symb = sp.symbols(VARS)
    X = dict(zip(X_vars,X_symb))

    #SPLIT -> Convert comma separted equations to list
    #FUNC -> SPLIT into LHS and RHS > Convert LHS and RHS to sympy Expression > Cor
    SP_EQ = list(map(lambda eq: sp.Eq(*[sp.simplify(i,locals = X) for i in eq.split
    # {Variable:Symbol},[Equations]
    return X,SP_EQ
```

In [23]:
```
stringToEquation("a,b,c","a + 2*b = 2, 20*a + b - 2*c = 17")
```

Out[23]:
```
({'a': a, 'b': b, 'c': c}, [Eq(a + 2*b, 2), Eq(20*a + b - 2*c, 17)])
```

> ### Gauss Jacobi Method

In [354…

```
def gaussJacobi(variables:str, equations:str,steps:bool = False, error: float = 1e-
    iter = 0;
    X,EQ = stringToEquation(variables,equations)
    X = list(X.values())

    #To Make Diagonaly Dominant
    D = [eq.args[0].as_coefficients_dict(*X) for eq in EQ]
    indices = [np.argmax([d[x] for x in X]) for d in D]
    EQ = list(np.array(EQ)[indices])

    funcs = [sp.lambdify(X,sp.solve(EQ[i],X[i]),"numpy") for i in range(len(X))]
    X = np.zeros((2,len(X)))
    X[0] += 1
    while((np.max(abs(np.diff(X,axis=0))) > error) and (iter<itermax)):
        X[0] = X[1]
        X[1] = [f(*X[0])[0] for f in funcs]
        iter += 1;
        if(steps):
            print(iter," ",[f"{x:.5f}" for x in X[-1]])
    return list(map(lambda x: round(x,3),X[-1])),iter
```

> $20x + y - 2z = 17$</br> $3x + 20y - z = -18$</br> $2x - 3y + 20z = 25$
> </br>

In [355…

```
gaussJacobi("a,b,c","20*a + b - 2*c = 17, 3*a + 20*b - c = -18, 2*a - 3*b + 20*c =
```

```
1   ['0.85000', '-0.90000', '1.25000']
2   ['1.02000', '-0.96500', '1.03000']
3   ['1.00125', '-1.00150', '1.00325']
4   ['1.00040', '-1.00002', '0.99965']
5   ['0.99997', '-1.00008', '0.99996']
6   ['1.00000', '-1.00000', '0.99999']
7   ['1.00000', '-1.00000', '1.00000']
8   ['1.00000', '-1.00000', '1.00000']
9   ['1.00000', '-1.00000', '1.00000']
```

Out[355]:

```
([1.0, -1.0, 1.0], 9)
```

> $27x + 6y - z = 85$</br> $x + y + 54z = 110$</br> $6x + 15y + 2z = 72$
> </br>

In [334…

```
gaussJacobi("x,y,z",
            "27*x + 6*y - z = 85, x + y + 54*z = 110, 6*x + 15*y + 2*z = 72",steps
```

```
1   ['3.14815', '4.80000', '2.03704']
2   ['2.15693', '3.26914', '1.88985']
3   ['2.49167', '3.68525', '1.93655']
4   ['2.40093', '3.54513', '1.92265']
5   ['2.43155', '3.58328', '1.92692']
6   ['2.42323', '3.57046', '1.92565']
7   ['2.42603', '3.57395', '1.92604']
8   ['2.42527', '3.57278', '1.92593']
9   ['2.42553', '3.57310', '1.92596']
10   ['2.42546', '3.57299', '1.92595']
11   ['2.42548', '3.57302', '1.92595']
12   ['2.42547', '3.57301', '1.92595']
13   ['2.42548', '3.57302', '1.92595']
14   ['2.42548', '3.57302', '1.92595']
```

Out[334]:

```
([2.425, 3.573, 1.926], 14)
```

> $6x + y + z = 20$</br> $x + 4y - z = 6$</br> $x - y + 5z = 7$</br>

In [335…

```
gaussJacobi("x,y,z",
            "6*x + y + z = 20, x + 4*y - z = 6, x - y + 5*z = 7")
```

Out[335]:

```
([3.0, 1.0, 1.0], 15)
```

**Date: 27-01-24**

> **Gauss Seidel Method**

In [359…

```
def gaussSeidel(variables:str, equations:str,steps:bool = False, error: float = 1e-
    iter = 0;
    X,EQ = stringToEquation(variables,equations)
    X = list(X.values())

    #To Make Diagonaly Dominant
    D = [eq.args[0].as_coefficients_dict(*X) for eq in EQ]
    indices = [np.argmax([d[x] for x in X]) for d in D]
    EQ = list(np.array(EQ)[indices])

    funcs = [sp.lambdify(X,sp.solve(EQ[i],X[i]),"numpy") for i in range(len(X))]
    X = np.zeros((2,len(X)))
    X[0] += 1
    while((np.max(abs(np.diff(X,axis=0))) > error) and (iter<itermax)):
        X[0] = X[1]
        for i in range(len(X[1])):
            X[1][i] = funcs[i](*X[1])[0]
        iter += 1;
        if(steps):
            print(iter," ",[f"{x:.5f}" for x in X[-1]])
    return list(map(lambda x: round(x,2),X[-1])),iter
```

> WAP to solve using Gauss Jacobi and Gauss Seidel, comment which method converges faster
> $4x + 2y + z = 14$
> $x + 5y - z = 10$
> $x + y + 8z = 20$

In [368…

```
equations = "4*x + 2*y + z = 14, x + 5*y - z = 10, x + y + 8*z = 20"
vars = "x,y,z"
print("Gauss Jacobi")
n1 = gaussJacobi(vars,equations,steps=True,error=1e-8)
print("Gauss Seidel")
n2 = gaussSeidel(vars,equations,steps=True,error=1e-8)
n1,n2,"GAUSS-SEIDEL CONVERGES FASTER THAN GAUSS-JACOBI"
```

```
Gauss Jacobi
1   ['3.50000', '2.00000', '2.50000']
2   ['1.87500', '1.80000', '1.81250']
3   ['2.14688', '1.98750', '2.04062']
4   ['1.99609', '1.97875', '1.98320']
5   ['2.01482', '1.99742', '2.00314']
6   ['2.00050', '1.99766', '1.99847']
7   ['2.00155', '1.99959', '2.00023']
8   ['2.00015', '1.99974', '1.99986']
9   ['2.00017', '1.99994', '2.00001']
10   ['2.00003', '1.99997', '1.99999']
11   ['2.00002', '1.99999', '2.00000']
12   ['2.00000', '2.00000', '2.00000']
13   ['2.00000', '2.00000', '2.00000']
14   ['2.00000', '2.00000', '2.00000']
15   ['2.00000', '2.00000', '2.00000']
16   ['2.00000', '2.00000', '2.00000']
17   ['2.00000', '2.00000', '2.00000']
18   ['2.00000', '2.00000', '2.00000']
19   ['2.00000', '2.00000', '2.00000']
Gauss Seidel
1   ['3.50000', '1.30000', '1.90000']
2   ['2.37500', '1.90500', '1.96500']
3   ['2.05625', '1.98175', '1.99525']
4   ['2.01031', '1.99699', '1.99909']
5   ['2.00173', '1.99947', '1.99985']
6   ['2.00030', '1.99991', '1.99997']
7   ['2.00005', '1.99998', '2.00000']
8   ['2.00001', '2.00000', '2.00000']
9   ['2.00000', '2.00000', '2.00000']
10   ['2.00000', '2.00000', '2.00000']
11   ['2.00000', '2.00000', '2.00000']
12   ['2.00000', '2.00000', '2.00000']
13   ['2.00000', '2.00000', '2.00000']
```

Out[368]:
```
(([2.0, 2.0, 2.0], 19),
 ([2.0, 2.0, 2.0], 13),
 'GAUSS-SEIDEL CONVERGES FASTER THAN GAUSS-JACOBI')
```

> $x + 3y + 52z = 173.61$ </br> $x - 27y + 2z = 71.31$ </br> $41x - 2y + 3z = 65.46$ </br>

In [375…
```
gaussSeidel("x,y,z",
            "x + 3*y + 52*z = 173.61, x - 27*y + 2*z = 71.31, 41*x - 2*y + 3*z = 65.
```

```
1   ['1.59659', '0.00000', '3.30795']
2   ['1.35454', '0.00000', '3.31260']
3   ['1.35420', '0.00000', '3.31261']
4   ['1.35420', '0.00000', '3.31261']
5   ['1.35420', '0.00000', '3.31261']
6   ['1.35420', '0.00000', '3.31261']
```

Out[375]:
```
([1.35, 0.0, 3.31], 6)
```

In [ ]:
```

```