Department of Computer Science and Engineering
(UG Studies)
PES University, Bangalore-560085

Session : Aug - Dec2017 **Credits :** 0-0-2-01	UE14CS405 : Machine Learning Lab
Lab # : 02	Find solutions to two large Ax= b systems with and without Linear dependence and compute Basis and Eigen Vectors/Values.

Task 1 Solve linear Equations

DataSet:
2x-3y=3
4x-5y+z=7
2x-y-3z=5

Theory

<u>Partial Pivoting</u>

Suppose we have A and b matrices given below

$$\left[\begin{array}{cccc|c} 0.02 & 0.01 & 0 & 0 & 0.02 \\ 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

Step 1: Find the entry in the left column with the largest absolute value.

This entry is called the pivot

Step 2: Perform row interchange (if necessary), so that the pivot

is in the first row.

$$
\left[\begin{array}{cccc|c}
0.02 & 0.01 & 0 & 0 & 0.02 \\
1 & 2 & 1 & 0 & 1 \\
0 & 1 & 2 & 1 & 4 \\
0 & 0 & 100 & 200 & 800
\end{array}\right]
$$

$$\downarrow$$

$$
\left[\begin{array}{cccc|c}
1 & 2 & 1 & 0 & 1 \\
0.02 & 0.01 & 0 & 0 & 0.02 \\
0 & 1 & 2 & 1 & 4 \\
0 & 0 & 100 & 200 & 800
\end{array}\right]
$$

Step 1: Gaussian Elimination

$$
\left[\begin{array}{cccc|c}
1 & 2 & 1 & 0 & 1 \\
0.02 & 0.01 & 0 & 0 & 0.02 \\
0 & 1 & 2 & 1 & 4 \\
0 & 0 & 100 & 200 & 800
\end{array}\right]
$$

$$\downarrow$$

$$
\left[\begin{array}{cccc|c}
1 & 2 & 1 & 0 & 1 \\
0 & -0.03 & -0.02 & 0 & 0 \\
0 & 1 & 2 & 1 & 4 \\
0 & 0 & 100 & 200 & 800
\end{array}\right]
$$

Step 2: Find new pivot

Step 3: Switch rows (if necessary)

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & -0.03 & -0.02 & 0 & 0 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & -0.03 & -0.02 & 0 & 0 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

Step 4: Gaussian Elimination

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & -0.03 & -0.02 & 0 & 0 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 0.04 & 0.03 & 0.12 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

Step 5: Find new pivot

Step 6: Switch rows (if necessary)

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 0.04 & 0.03 & 0.12 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 100 & 200 & 800 \\ 0 & 0 & 0.04 & 0.03 & 0.12 \end{array}\right]$$

Step 7: Gaussian Elimination

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 100 & 200 & 800 \\ 0 & 0 & 0.04 & 0.03 & 0.12 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 100 & 200 & 800 \\ 0 & 0 & 0 & -0.05 & -0.2 \end{array}\right]$$

Step 8: Back Substitute

-0.2x4= -0.05;x4= 4

$$100x_3 + 200x_4 = 800; x_3 = 0,$$

$$x_2 + 2x_3 + x_4 = 4; x_2 = 0,$$

$$x_1 + 2x_2 + x_3 = 1; x_1 = 1$$

There fore

$x_1=1, x_2=0, x_3=0, x_4=4$

Code for Gaussian elimination without partial pivoting:

```
import numpy as np
def Gauss(A, b):
    '''
    Gaussian elimination with no pivoting.
    % input: A is an n x n nonsingular matrix
    %        b is an n x 1 vector
    % output: x is the solution of Ax=b.

    '''
    n =  len(A)
    if b.size != n:
        raise ValueError("Invalid argument: incompatible sizes between A & b.", b.size, n)
    for pivot_row in xrange(n-1):
        for row in xrange(pivot_row+1, n):
            multiplier = A[row][pivot_row]/A[pivot_row][pivot_row]
            #the only one in this column since the rest are zero
            A[row][pivot_row] = multiplier
            for col in xrange(pivot_row + 1, n):
                A[row][col] = A[row][col] - multiplier*A[pivot_row][col]
            #Equation solution column
            b[row] = b[row] - multiplier*b[pivot_row]
    print 'REUSLTS AFTER GUASSIAN ELIMINATION'
    print '-------------------------------------'
    print A
    print b
```

```
    x = np.zeros(n)

    #print 'before',x

    k = n-1

    x[k] = b[k]/A[k,k]

    #print 'b value is ',b[k]

    while k >= 0:

        x[k] = (b[k] - np.dot(A[k,k+1:],x[k+1:]))/A[k,k]

        k = k-1

    return x


if __name__ == "__main__":

    # Take matrix A

    # Take Matrix b
```

To Do list for solving linear equations

1. Execute the given code with A and b matrices (Input the matrices A and b)for cosistent Systems
2. Compare the output with matrices for Inconsistent Systems
3. What happens if the pivot element is zero?
4. Modify the given code with partial pivoting(Example for partial pivoting is given)
5.Check the results manually with output got from python code

Task 2: Finding Eigen values and Eigen vectors

Code:

```
#eigen value and eigen vector finding without linalg.eig

import numpy as np
a=np.matrix([[4,3],[-2,-3]])
print(a)
def sumOfDiagonals(arr):
    sum = 0
    for i in range(len(arr)):
        sum += arr[i][i]
    return sum

print('sum of diagonal values is',sumOfDiagonals([[4,3],[-2,-3]]))
```

```
print('determinent of a is ',np.linalg.det(a)) # computes determinent of matrix

print(a)



# computing roots of a characteristic equation
coeff=[1,-1,-6]
print('eigen values are',np.roots(coeff))

#eigen values are 3 and -2

b=np.matrix([[4,3],[-2,-3]])

c=np.matrix([[3,0],[0,3]])

z=b-c #A-3I
print('eigen vectors are',z)
p=np.matrix([[4,3],[-2,-3]])

q=np.matrix([[2,0],[0,2]])

y=p+q #A+2I
print('eigen vectors are',y)
# (1, -2) can be taken as an eigenvector associated with the eigenvalue -2, and (3, -1) as an eigenvector associated with the eigenvalue 3, as can be verified by multiplying them by A.
```

To do for eigen values and eigen vectors

Determine eigen values and eigen vectors using python for the matrix

8 -6 2
-6 7 -4
2 -4 3

Learning outcome:

1. Eigenvectors and eigenvalues have many important applications in computer vision and machine learning in general.
2. Well known examples are PCA (Principal Component Analysis) for dimensionality reduction or EigenFaces for face recognition.
3. Furthermore, eigendecomposition forms the base of the geometric interpretation of covariance matrices.