```
#lab 7 Gauss method :
import numpy as np
import sys

# Reading number of unknowns
n = int(input('Enter number of unknowns: ')) # its saves the shape of matrix

# Making numpy array of n x n+1 size and initializing
# to zero for storing augmented matrix
a = np.zeros((n,n+1)) # its forms the a[][]

# Making numpy array of n size and initializing
# to zero for storing solution vector
x = np.zeros(n) # it assign the argument of x with 0
# Reading augmented matrix coefficients
print('Enter Augmented Matrix Coefficients:') #its reads actual matrix value
for i in range(n): # loop for running upto n -1 if the users ip shape 3 goes to goes to 0 to 2
  for j in range(n+1):
    a[i][j] = float(input( 'a['+str(i)+']['+ str(j)+']='))

# Applying Gauss Elimination
for i in range(n):
  if a[i][i] == 0.0:
    sys.exit('Divide by zero detected!')
  for j in range(i+1, n):
    ratio = a[j][i]/a[i][i]

    for k in range(n+1):
      a[j][k] = a[j][k] - ratio * a[i][k]
# Back Substitution
x[n-1] = a[n-1][n]/a[n-1][n-1]
for i in range(n-2,-1,-1):
  x[i] = a[i][n]
  for j in range(i+1,n):
    x[i] = x[i] - a[i][j]*x[j]
  x[i] = x[i]/a[i][i]

# Displaying solution
print('\nRequired solution is: ')
for i in range(n):
  print('X%d = %0.2f' %(i,x[i]), end = '\t')
```

```
Enter number of unknowns: 3
Enter Augmented Matrix Coefficients:
a[0][0]=1
a[0][1]=2
a[0][2]=3
a[0][3]=6
a[1][0]=2
a[1][1]=-1
a[1][2]=3
a[1][3]=4
a[2][0]=2
a[2][1]=3
a[2][2]=5
a[2][3]=10

Required solution is:
X0 = 1.00       X1 = 1.00       X2 = 1.00
C:\Users\DELL\Desktop\Nummerical Methods  Lab>
```

Lab 8:

```
#Power Method

# Power Method to Find Largest Eigen Value and Eigen Vector
# Importing NumPy Library
import numpy as np
import sys

# Reading order of matrix
n = int(input('Enter order of matrix: '))

# Making numpy array of n x n size and initializing
# to zero for storing matrix
a = np.zeros((n,n))

# Reading matrix
print('Enter Matrix Coefficients:')
for i in range(n):
    for j in range(n):
        a[i][j] = float(input( 'a['+str(i)+']['+ str(j)+']='))

# Making numpy array n x 1 size and initializing to zero
# for storing initial guess vector
x = np.zeros((n))

# Reading initial guess vector
print('Enter initial guess vector: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+']='))

# Reading tolerable error
tolerable_error = float(input('Enter tolerable error: '))

# Reading maximum number of steps
max_iteration = int(input('Enter maximum number of steps: '))

# Power Method Implementation
lambda_old = 1.0
condition =  True
step = 1
while condition:
    # Multiplying a and x
```

```
    x = np.matmul(a,x)
    # Finding new Eigen value and Eigen vector
    lambda_new = max(abs(x))
    x = x/lambda_new
    # Displaying Eigen value and Eigen Vector
    print('\nSTEP %d' %(step))
    print('----------')
    print('Eigen Value = %0.4f' %(lambda_new))
    print('Eigen Vector: ')
    for i in range(n):
        print('%0.3f\t' % (x[i]))
    # Checking maximum iteration
    step = step + 1
    if step > max_iteration:
        print('Not convergent in given maximum
iteration!')
        break
    # Calculating error
    error = abs(lambda_new - lambda_old)
    print('errror='+ str(error))
    lambda_old = lambda_new
    condition = error > tolerable_error
```

```
STEP 7
----------
Eigen Value = 7.6499
Eigen Vector:
0.597
0.485
1.000
errror=0.003930879316461855

STEP 8
----------
Eigen Value = 7.6486
Eigen Vector:
0.597
0.485
1.000
errror=0.0013088801273761774

STEP 9
----------
Eigen Value = 7.6490
Eigen Vector:
0.597
0.485
1.000
errror=0.00043567353822915532
```

Discussion and Conclusion:

The Gauss elimination method systematically reduces a system of linear equations to an upper triangular form, making it efficient for direct solutions, especially for small to medium-sized systems. It ensures accuracy but can become unstable with ill-conditioned matrices unless partial pivoting is used. On the other hand, the Power method is an iterative technique specifically designed to find the dominant eigenvalue and its corresponding eigenvector of a matrix. It is simple to implement and works well when the dominant eigenvalue is significantly larger than the others. However, it may fail or converge slowly if this condition is not met. Both methods highlight the balance between direct and iterative strategies in numerical methods—Gauss focusing on precision and generality, while Power emphasizes simplicity and specific applications in eigenvalue problems.