Lab 5 Report

Amlan Nayak MS18197

24th Feb 2021

1 Multiply Matrix A and B

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 1 & 4 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 1 \\ 1 & 2 \\ 2 & 1 \end{bmatrix}$$

1.1 Algorithm

An basic algorithm was written to ensure the multiplication with arrays in numpy module.

The code has been written for a general system. By changing the input arrays, one can find the resultant matrix given the conditions of dimensions is satisfied.

1.2 Code and Output

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$ python3 MS
[[10.  8.]
 [13.  8.]
 [12. 12.]]
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$
```

Figure 1: Output

```
File Edit Options Buffers Tools Python Virtual Envs Elpy Flym
1import numpy as np
2A= np.array([[1,2,3],[2,1,4],[1,4,3]],dtype=float)
3B= np.array([[2,1],[1,2],[2,1]],dtype=float)
4def mat_mul(A,B):
5  if A.shape[1]==B.shape[0]:
6    C= np.zeros(shape=(A.shape[0],B.shape[1]),dtype=float)
7  else:
8    print("incorrect dimensions")
9  for i in range(0,A.shape[0]):
10   for j in range(0,B.shape[1]):
11     a= A[i,:]*B[:,j]
12     sum=0
13     for k in a:
14       sum = sum + k
15     C[i,j]= sum
16 print(C)
17mat_mul(A,B)
```

Figure 2: Multiplication Code in Python in emacs

2 Use partial pivoting to solve the following equations using elimination

$$2x_2 + x_3 = 5$$

$$4x_1 + x_2 - x_3 = -3$$

$$-2x_1 + 3x_2 - 3x_3 = 5$$

2.1 Algorithm

The augmented matrix $[A|B]$ is the input array. The algorithm iterates through the columns of A in the augmented matrix one by one. In each iteration, we use the partial pivoting procedure to find the row which has maximum absolute value in the chosen column and set it as the new pivot row, interchanging the rows in the process. After this is done, we set the pivot element as 1 by dividing the pivot row by the pivot element. We then commence the Gaussian elimination.

Once this is done, we use a simple back substitution method to find the solutions.

The code has been written for a general system. By changing the input arrays, one can find the solutions for the required system of equations.

2.2 Code and Output

```
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASnippet He
import numpy as np
from numpy.linalg import matrix_rank
A= np.array([[0,2,1,5],[4,1,-1,-3],[-2,3,-3,5]], dtype=float)
rows = A.shape[0]
for i in range(rows):
  A_copy= A.copy()
  #pivoting procedure
  maxi = A[:,i].tolist().index(max(A[:,i].tolist(),key=abs))
  if maxi > i:
    mid = A_copy[i,:]
    A[i,:]=A[maxi,:]
    A[maxi,:] = mid
  else:
    pass
  #making pivots 1 and starting elimination
  A[i,:]= A[i,:]/A[i,i]
  for j in range(i+1,rows):
    if A[j,i]!=0:
      A[j,:] =   A[j,:] -  A[j,i]*A[i,:]
    else:
      pass
#check rank for consistency
if matrix_rank(A) == matrix_rank(A[:,0:rows]):
#back substituition
  sol= np.zeros((rows,1))
  for i in range(rows-1, -1, -1):
    c = A[i,rows]
    for j in range(rows-1, i, -1):
      c = c - sol[j]*A[i,j]
    sol[i] = c/A[i,i]
  print("The solutions are (" + str(["x"+str(i) for i in range(1,rows+1)])
  + "):",[float(i) for i in sol])
else:
  print("Equations not consistent")
```

Figure 3: Gaussian method with partial pivoting code

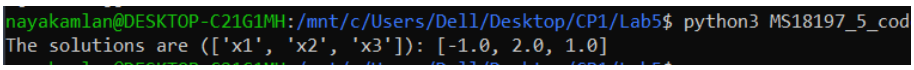

Figure 4: Output

3 Use Gauss Jordan method to solve the following equations:

$$2x_1 + 4x_2 + x_3 = 3$$

$$3x_1 + 2x_2 - 2x_3 = -2$$

$$3x_1 - 3x_2 + 3x_3 = 18$$

3.1 Algorithm

The augmented matrix $[A|B]$ is the input array. The algorithm iterates through the columns of A in the augmented matrix one by one. In each iteration, we use the partial pivoting procedure to find the row which has maximum absolute value in the chosen column and set it as the new pivot row, interchanging the rows in the process. After this is done, we set the pivot element as 1 by dividing the pivot row by the pivot element. We then commence the Gaussian elimination. Here we not only eliminate the entries below the pivot, but also entries above the pivot.

Once this is done, the B part in the modified augmented matrix $[A|B]$ is the solution.

The code has been written for a general system. By changing the input arrays, one can find the solutions for the required system of equations.

3.2 Code and Output

```
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASnippet
import numpy as np
A= np.array([[2,4,1,3],[3,2,-2,-2],[3,-3,3,18]], dtype=float)
rows = A.shape[0]
for i in range(rows):
    A_copy= A.copy()
    #pivoting procedure
    maxi = A[:,i].tolist().index(max(A[:,i].tolist(),key=abs))
    if maxi > i:
        mid = A_copy[i,:]
        A[i,:]=A[maxi,:]
        A[maxi,:] = mid
    else:
        pass
    #making pivots 1 and starting elimination
    A[i,:]= A[i,:]/A[i,i]
    for j in range(i+1,rows):
        if A[j,i]!=0:
            A[j,:] =   A[j,:] -  A[j,i]*A[i,:]
        else:
            pass
    if i>0:
        for k in range(i):
            A[k,:] =   A[k,:] -  A[k,i]*A[i,:]
    else:
        pass
print("The solutions are (" + str(["x"+str(i) for i in range(1,rows+1)])
+ "): ",[round(i,4) for i in A[:,rows]])
```

Figure 5: Gauss Jordan method code

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$ python3 MS18197_5_code3
The solutions are (['x1', 'x2', 'x3']):  [2.0, -1.0, 3.0]
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$
```

Figure 6: output

4 Solve the following equations using Dolittle and Crout-decomposition:

$$2x_1 + x_2 + 4x_3 = 12$$

$$8x_1 - 3x_2 + 2x_3 = 20$$

$$4x_1 + 11x_2 - x_3 = 33$$

4.1 Algorithm

Here we try to determine two matrices L,U such that we can write $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. We use two methods to find the these matrices.

- Crout Method - In this method, the diagonal entries of the U matrix are set to be 1. Then, as discussed in class lectures use use the following relations to find the elements of the matrix:

$$U_{ii} = 1$$

$$L_{ii} = A_{ii} - \sum_{k=1}^{i-1} L_{ik}U_{ki}$$

$$L_{ij} = A_{ij} - \sum_{k=1}^{j-1} L_{ik}U_{kj}, i > j$$

$$U_{ij} = \frac{A_{ij} - \sum_{k=1}^{j-1} L_{ik}U_{kj}}{L_{ii}}, i > j$$

- Dolittle Method - In this method, the diagonal entries of the L matrix are set to be 1. Then, as discussed in class lectures use use the following relations to find the elements of the matrix:

$$L_{ii} = 1$$

$$U_{ii} = A_{ii} - \sum_{k=1}^{i-1} L_{ik}U_{ki}$$

$$U_{ij} = A_{ij} - \sum_{k=1}^{j-1} L_{ik}U_{kj}, i > j$$

$$L_{ij} = \frac{A_{ij} - \sum_{k=1}^{j-1} L_{ik}U_{kj}}{U_{jj}}, i > j$$

Once we determine L and U using either of the two methods mentioned above, we use forward substitution method to determine Z such that $LZ = B$. Once we determine Z, we use back substitution method to determine our solution set X such that $UX = Z$.

The code has been written for a general system. By changing the input arrays A and B in $AX = B$, one can find the solutions for the required system of equations.

For finding inverse of A, we use the matrices L and U determined above. We create two augmented matrices $[L|I]$ and $[U|I]$, where I is the identity matrix, we then perform Gauss Jordan elimination method described before on the rows of L and U part of the augmented matrices. We end up with $[I|L^{-1}]$ and $[I|U^{-1}]$, where L^{-1} and U^{-1} are the inverse of L and U.

We then multiply A^{-1} from U^{-1} and L^{-1} using the relation:

$$A^{-1} = U^{-1}L^{-1}$$

The inverse code has been written for a general system. By changing the input arrays A, one can find the inverse for the matrix.

4.2 Code and Output

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$ python3 MS18197_5_code4_Crout.p
The solutions through Crout method are (['x1', 'x2', 'x3']): [3.0, 2.0, 1.0]
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$
```

Figure 7: output for Crout method

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$ python3 MS18197_5_code4_Dolittle
The solutions through Dolittle method are (['x1', 'x2', 'x3']): [3.0, 2.0, 1.0]
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$
```

Figure 8: output for Dolittle method

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$ python3 MS18197_5_code4_inverse
[[-0.05026455  0.11904762  0.03703704]
 [ 0.04232804 -0.04761905  0.07407407]
 [ 0.26455026 -0.04761905 -0.03703704]]
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab5$
```

Figure 9: output for finding Inverse

```
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASni
import numpy as np
#A= np.array([[2,-3,1],[1,2,-3],[4,-1,-2]], dtype=float)
A= np.array([[2,1,4],[8,-3,2],[4,11,-1]], dtype=float)
B=np.array([12,20,33],dtype=float)
rows = A.shape[0]
U = np.zeros((rows, rows), dtype=float)
L = np.zeros((rows, rows), dtype=float)
for k in range(rows):
   L[k, k] = A[k, k] - np.dot( L[k, :] , U[:, k])
   U[k, k:] = (A[k, k:] -  np.dot(L[k, :k] , U[:k, k:])) / L[k, k]
   L[(k+1):,k] = (A[(k+1):,k]-np.dot(L[(k+1):,:],U[:,k]))/U[k,k]
sol= np.zeros((rows,1))
for i in range(rows):
    c = B[i]
    for j in range(i):
        c = c - sol[j]*L[i,j]
    sol[i] = c/L[i,i]
for i in range(rows-1, -1, -1):
    c = sol[i]
    for j in range(rows-1, i, -1):
        c = c - sol[j]*U[i,j]
    sol[i] = c
print("The solutions through Crout method are ("
+str(["x"+str(i) for i in range(1,rows+1)])+"):"
,[float(i) for i in sol])
```

Figure 10: Crout method in Python

```
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YAS
 1import numpy as np
 2A= np.array([[2,1,4],[8,-3,2],[4,11,-1]], dtype=float)
 3B=np.array([12,20,33],dtype=float)
 4rows = A.shape[0]
 5U = np.zeros((rows, rows), dtype=float)
 6L = np.eye(rows, dtype=float)
 7for k in range(rows):
 8    U[k, k:] = A[k, k:] -  np.dot(L[k, :k] , U[:k, k:])
 9    L[(k+1):, k] = (A[(k+1):,k]-np.dot(L[(k+1):,:],U[:,k]))/U[k,k]
10sol= np.zeros((rows,1))
11#Solves LZ=B for Z
12for i in range(rows):
13    c = B[i]
14    for j in range(i):
15        c = c - sol[j]*L[i,j]
16    sol[i] = c
17#Solves UX=Z for X
18for i in range(rows-1, -1, -1):
19    c = sol[i]
20    for j in range(rows-1, i, -1):
21        c = c - sol[j]*U[i,j]
22    sol[i] = c/U[i,i]
23print("The solutions through Dolittle method are ("
24+str(["x"+str(i) for i in range(1,rows+1)])+"):"
25,[float(i) for i in sol])
```

Figure 11: Dolittle method in Python

```
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASnippet Help
import numpy as np
A= np.array([[2,1,4],[8,-3,2],[4,11,-1]], dtype=float)
B=np.array([12,20,33],dtype=float)
rows = A.shape[0]
U = np.zeros((rows, rows), dtype=float)
L = np.zeros((rows, rows), dtype=float)
for k in range(rows):
        L[k, k] = A[k, k] - np.dot( L[k, :] , U[:, k])
        U[k, k:] = (A[k, k:] -  np.dot(L[k, :k] , U[:k, k:])) / L[k, k]
        L[(k+1):, k] = (A[(k+1):, k] -  np.dot(L[(k+1):, :], U[:, k])) / U[k, k]
def inverse(a,rows): #inverse of a matrix
  D=np.zeros((rows,2*rows),dtype=float)
  D[:,:rows]=a
  D[:,rows:]=np.eye(rows,dtype=float)
  D_copy=D.copy()
  for i in range(rows):
    D_copy= D.copy()
    #making pivots 1 and starting elimination
    D[i,:]= D[i,:]/D[i,i]
    for j in range(i+1,rows):
      if D[j,i]!=0:
        D[j,:] =  D[j,:] -  D[j,i]*D[i,:]
      else:
        pass
    if i>0:
      for k in range(i):
        D[k,:] =   D[k,:] -  D[k,i]*D[i,:]
    else:
      pass
  return D[:,rows:]
```

Figure 12: Python code for finding inverse

```
def mat_mul(A,B): #Multiplies two matrix
  C= np.zeros(shape=(A.shape[0],B.shape[1]),dtype=float)
  for i in range(0,A.shape[0]):
    for j in range(0,B.shape[1]):
      a= A[i,:]*B[:,j]
      sum=0
      for k in a:
        sum = sum + k
      C[i,j]= sum
  print(C)
mat_mul(inverse(U,rows),inverse(L,rows)) #inverse of U,L multiplied
```

Figure 13: Python code for finding inverse

5 Summary

Lab 5 explored various methods of solving systems of linear equations. Problem 1 was related to multiply two matrices together. This was used later in other problems' code. Problem 2 was using Gaussian elimination method with partial pivoting and Problem 3 was Gauss Jordan method. Problem 4 explored decomposition of matrix into product of a lower triangular matrix and an upper triangular matrix. Problem 4 also explored finding inverse of a matrix.

All codes used were written for any generalized square matrix.