Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 6

Matrices

Submitted by:
Canoza, Cherrylyn S.

Instructor:
Engr. Dylan Josh D. Lopez

December 09, 2020

I. Objectives

The goal of this laboratory activity is to project different function using matrices as a set of arraw that is held by the numpy python library to run the output. It aims to understand different methods how to work on codes with matrices depends on the output desired.

II. Methods

This laboratory activity again use the functions of matrices from the numpy library which is so wide and different creative stuffs can be found. There is no compicated code needed but if verifying if which it is a true type of matrix, it will deals with such analyzations and declarations to identify the answer.

In task one, it is requested to have a flowchat showing the method how it verify with the certain function they are using like in square matrix and null matrix, it can be display if they are tru square and null. This methods need an if statement in order to process some condition for it to run the code.

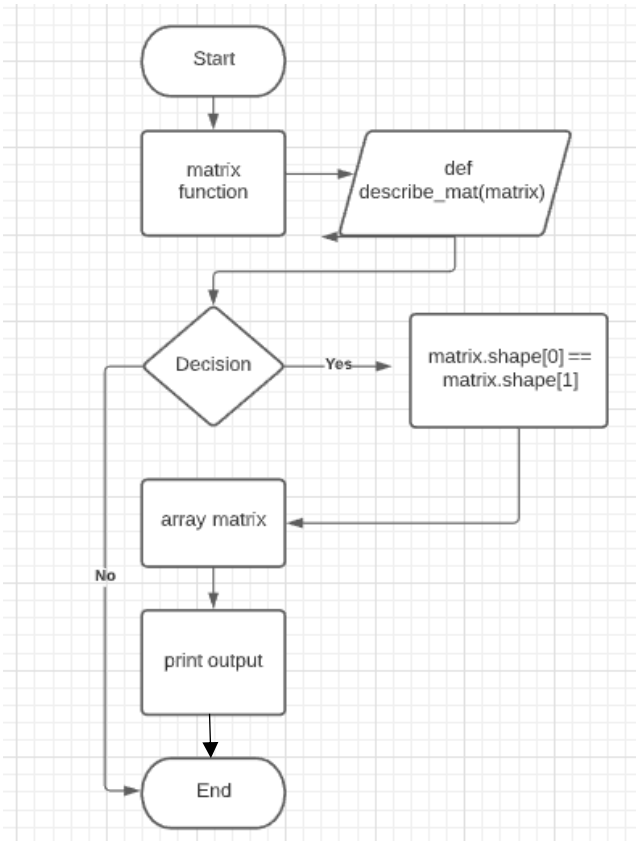

Figure 1 Task 1 Square Matrix Flowchart

This is a sample of a square flowchart by which it has condition that if the shape of the matrix array key value of zero is equal to the shape of matrix key value one then it is true that its shape is a square then else it is not. It is just a simple representation of the flowchart of verifying if the shape of a matrix make sense to each other as it calls the function of it.

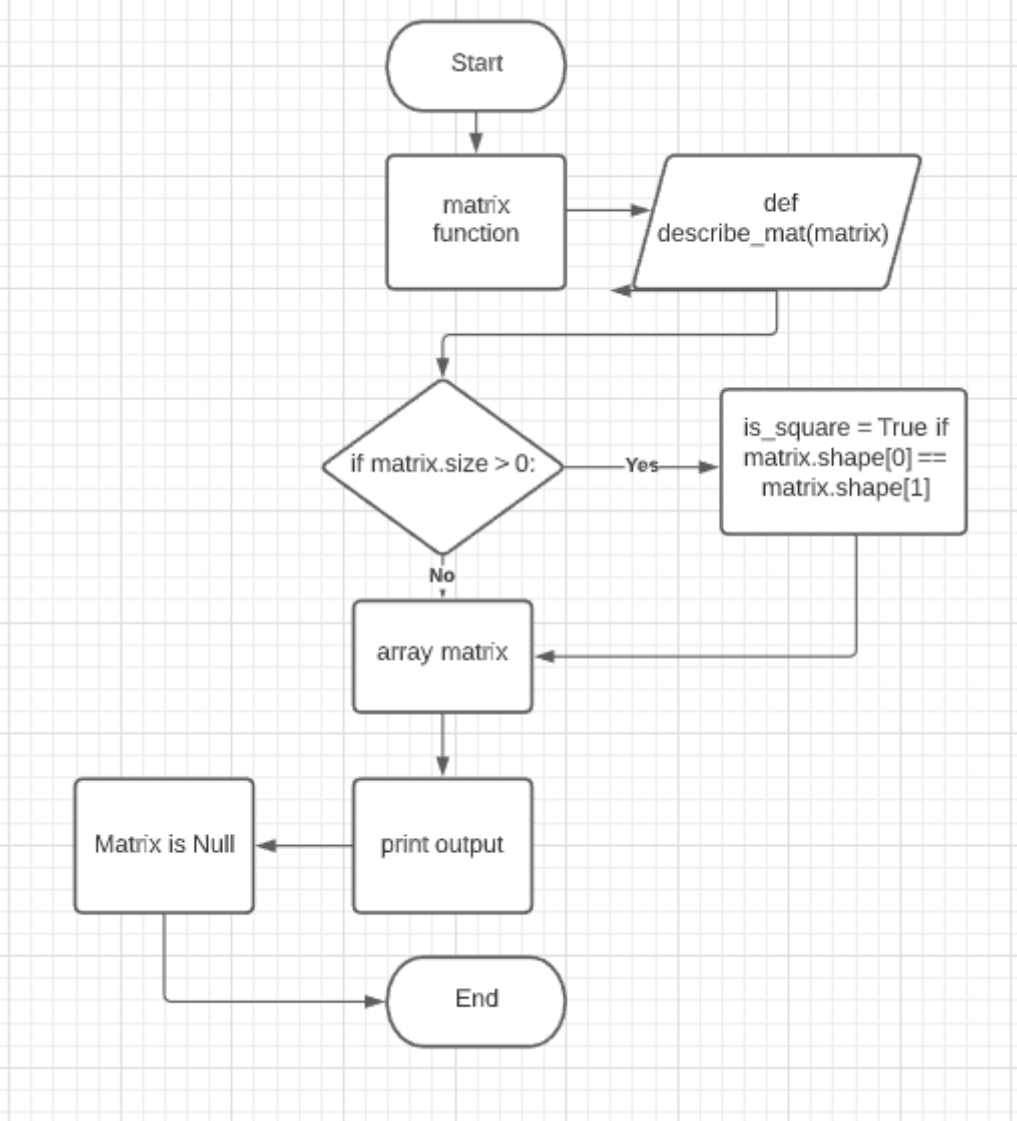

Figure 2 Task 1 Null Matrix Flowchart

Another example is the null type of matrix by which it projects an array without values inside of it and it just also simply verifying it by the size but now it is declared that if the size is greater than zero then condition inside of it saying that the shapes of each keys must be equals also hence it is false and will print out an output.

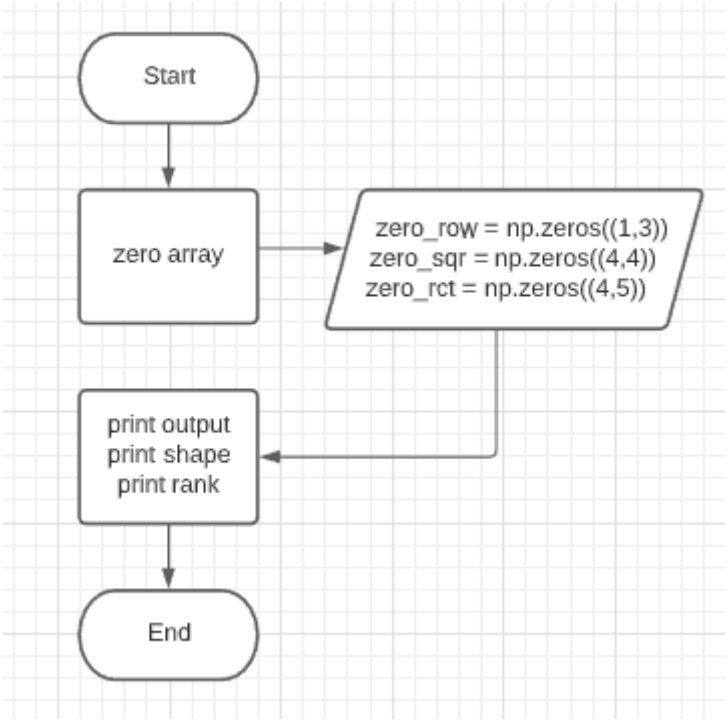

Figure 3 Task 1 Zero Matrix Flowchart

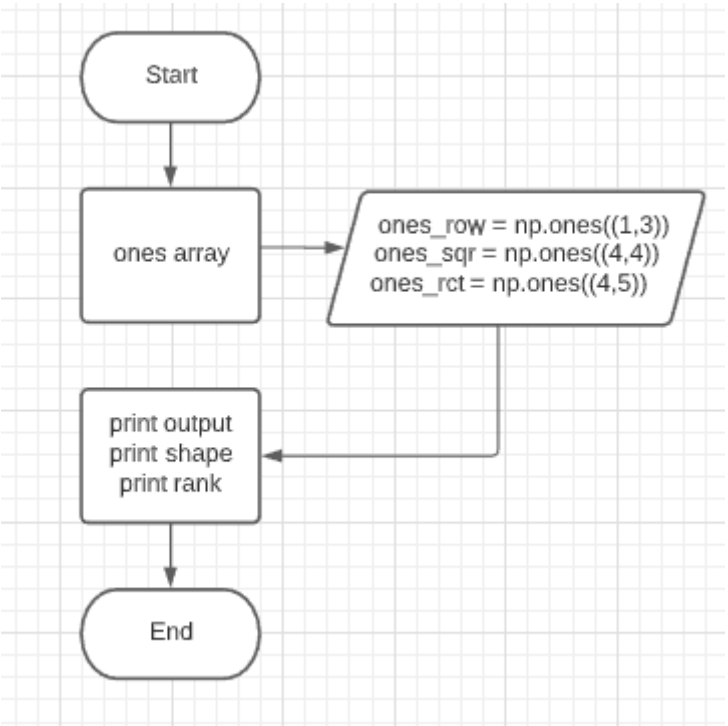

Figure 4 Task 1 Ones Matrix Flowchart

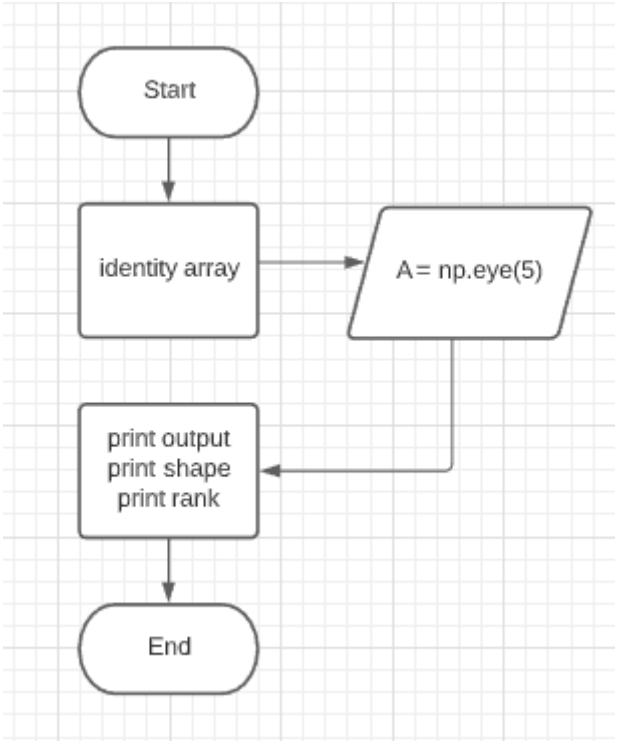

Figure 5 Task 1 Ones Matrix Flowchart

The other types of function of matrix are simply showing them as is it since it is more on the concept and so the flowchart just simple and directly to output only.

III. Results

The task one consists of different many codes of how to access matrices using different functions that can is also related to numpy since it is the library that consists of different functions in order to process an array. There is different amazing type of functions in manipulating an array or matrices such as the zeros, squares, identity and other more. There are no such complicated codes needed since it just needed to declare matrix as an array and then call the corresponding function and then call the variable and it’s now ready to run.

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as la
%matplotlib inline


def describe_mat(matrix):
    print(f'Matrix:\n{matrix}\n\nShape:\t{matrix.shape}\nRank:\t{matrix.ndim}\n')
A = np.array([
    [5, 7, 1],
    [3, 6, 2],
    [9, 1, 4]
])
describe_mat(A)
```

```
Matrix:
[[5 7 1]
 [3 6 2]
 [9 1 4]]

Shape:  (3, 3)
Rank:   2
```

Figure 6 Task 1 Matrix

The figure 5 is just a simple type of matrix array which also provides the size and what type of rank of matrix it is belong to.

In [29]:
```
def describe_mat(matrix):
    is_square = True if matrix.shape[0] == matrix.shape[1] else False
    print(f'Matrix:\n{matrix}\n\nShape:\t{matrix.shape}\nRank:\t{matrix.ndim}\nIs Square: {is_square}\n')


square = np.array([
    [1,2,3],
    [4,5,6],
    [7,8,9]
])

non_square = np.array([
    [6,5,4],
    [3,2,1]
])
describe_mat(square)
describe_mat(non_square)
```

```
Matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Shape:  (3, 3)
Rank:   2
Is Square: True

Matrix:
[[6 5 4]
 [3 2 1]]

Shape:  (2, 3)
Rank:   2
Is Square: False
```

Figure 7 Task 1 Square Matrix

The next one is the square matrix which should have the same number of column and row to attain a type of square matrix. Also, it provides verification if it is true square or not and it is actually obvious when you see the shape part, it should be the same to be considered as a square function.

```
In [49]: #Zero Matrix

         zero_row = np.zeros((1,3))
         zero_sqr = np.zeros((4,4))
         zero_rct = np.zeros((4,5))

         print(f'Matrix:\n{zero_row}\n\nShape:\t{zero_row.shape}\nRank:\t{zero_row.ndim}\n')
         print(f'Matrix:\n{zero_sqr}\n\nShape:\t{zero_sqr.shape}\nRank:\t{zero_sqr.ndim}\n')
         print(f'Matrix:\n{zero_sqr}\n\nShape:\t{zero_sqr.shape}\nRank:\t{zero_rct.ndim}\n')

         Matrix:
         [[0. 0. 0.]]

         Shape:  (1, 3)
         Rank:   2

         Matrix:
         [[0. 0. 0. 0.]
          [0. 0. 0. 0.]
          [0. 0. 0. 0.]
          [0. 0. 0. 0.]]

         Shape:  (4, 4)
         Rank:   2

         Matrix:
         [[0. 0. 0. 0.]
          [0. 0. 0. 0.]
          [0. 0. 0. 0.]
          [0. 0. 0. 0.]]

         Shape:  (4, 4)
         Rank:   2
```

Figure 8 Task 1 Zero Matrix

This function of matrix array shows an output of full zero by calling the function np.zero. The shape will be depending on what you want to input and it will matter as it changes but still, it will create a good output because it focuses on one number only.

In [12]:
```
#Ones Matrix

ones_row = np.ones((1,3))
ones_sqr = np.ones((4,4))
ones_rct = np.ones((4,5))

print(f'Matrix:\n{ones_row}\n\nShape:\t{ones_row.shape}\nRank:\t{ones_row.ndim}\n')
print(f'Matrix:\n{ones_sqr}\n\nShape:\t{ones_sqr.shape}\nRank:\t{ones_sqr.ndim}\n')
print(f'Matrix:\n{ones_rct}\n\nShape:\t{ones_rct.shape}\nRank:\t{ones_rct.ndim}\n')
```

```
Matrix:
[[1. 1. 1.]]

Shape:  (1, 3)
Rank:   2

Matrix:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

Shape:  (4, 4)
Rank:   2

Matrix:
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]

Shape:  (4, 5)
Rank:   2
```

Figure 9 Task 1 Ones Matrix

The other type of function is the ones matrix which is the same as the zero matrix but here, the number one is used instead of zero by calling the function np.ones, it will now create an output of matrix full of one.

In [34]:
```
#Null Matrix
def describe_mat(matrix):
    if matrix.size > 0:
        is_square = True if matrix.shape[0] == matrix.shape[1] else False
        print(f'Matrix:\n{matrix}\n\nShape:\t{matrix.shape}\nRank:\t{matrix.ndim}\nIs Square: {is_square}\n')
    else:
        print('Matrix is Null')

null = np.array([

])
describe_mat(null)
```

```
Matrix is Null
```

Figure 10 Task 1 Null Matrix

A null matrix is literally a matrix without a value inside and in here to declare a null, you should input no value inside your array also. In order to identify if it is null, if statement was created that has condition for its size then if satisfied that no values are found then it printed out that the matrix is a type of null matrix.

```
In [15]: A = np.eye(5)
         print(f'Matrix:\n{A}\n\nShape:\t{A.shape}\nRank:\t{A.ndim}\n')

Matrix:
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]

Shape:  (5, 5)
Rank:   2
```

Figure 11 Task 1 Identity Matrix

As shown in the output, an identity matrix is composed a two specific number but the major focus here is the slant one cover by zeroes around it. It identifies specific number and it is just simply calling the function np.eye and it will not give a unique slant effect of the specific number.

The task two consist of different operations in mathematics which are the addition, subtraction, multiplication, and division. In order to have operations, it needs to matrices or can be written as array in two different variable names so that on the codes, it will just be easily be accessed and manipulate it with different opeations.

```
In [67]: import numpy as np
         import matplotlib.pyplot as plt
         import scipy.linalg as la
         %matplotlib inline


         def describe_mat(matrix):
             print(f'Matrix:\n{matrix}\n\nShape:\t{matrix.shape}\nRank:\t{matrix.ndim}\n')
         X = np.array([
             [5, 7, 1],
             [3, 6, 2],
             [9, 1, 4]
         ])
         Y = np.array([
             [-2, 1, 4],
             [8, -3, 5],
             [3, 2, 6]
         ])
         describe_mat(X)
         describe_mat(Y)

Matrix:
[[5 7 1]
 [3 6 2]
 [9 1 4]]

Shape:  (3, 3)
Rank:   2

Matrix:
[[-2  1  4]
 [ 8 -3  5]
 [ 3  2  6]]

Shape:  (3, 3)
Rank:   2
```

Figure 12 Task 2 Matrix

This is the declared two array that will be used for all of the operations that will be done later on the codes. It also previews the size and the shape of the matrix array in order to analyze and understand how can it be done when operations are included.

```
In [69]: describe_mat(X+Y)

         Matrix:
         [[ 3  8  5]
          [11  3  7]
          [12  3 10]]

         Shape:  (3, 3)
         Rank:   2
```

Figure 13 Task 2 Addition Matrix

Here, it just simply called the function name and called the variables of the array inside the parameter and simply do the addition operation.

```
In [70]: describe_mat(X-Y)

         Matrix:
         [[ 7  6 -3]
          [-5  9 -3]
          [ 6 -1 -2]]

         Shape:  (3, 3)
         Rank:   2
```

Figure 14 Task 2 Subtraction Matrix

The method done here inside the function is the subtraction and still, the set of has been used to do the operation.

```
In [73]: describe_mat(X*Y)

         Matrix:
         [[-10   7   4]
          [ 24 -18  10]
          [ 27   2  24]]

         Shape:  (3, 3)
         Rank:   2
```

Figure 15 Task 2 Multiplication Matrix

Here, the multiplication operation has been called and done by which the multiplication operation was done by '*' or asterisk since it is the usual symbol for multiplication when being coded.

```
In [87]: X/Y

Out[87]: array([[-2.5       ,  7.        ,  0.25      ],
                [ 0.375     , -2.        ,  0.4       ],
                [ 3.        ,  0.5       ,  0.66666667]])
```

Figure 16 Task 2 Division Matrix

Lastly, this is the division operation and '/' or the slash key was used to the operation since it is the usual symbol for division to occur. The arrays from above are still being used to finish the division operation and this is the final ouput.

IV. Conclusion

Matrices are widely known with its useful application in the real life since these matrices are easily to be manipulate with using arrays and functions, operations and other different amazing things can be created or calculate using these. In coding, to attain the functions and different methods, a numpy library must be used so that terms can be accessed as function and it will now project the desired exact output it is meant to be as what function was declared. Matrices also can be related and helpful to the agriculture of the country by using the different opeations of it. It will have an easy access for the data using these matrices like for example, there's a certan data of information related to agriculture, and with this, it can be converted to an array type which can now be a matrix and with this matrix, you can now easily done different operations like adding of some data, multiplying some stock and others or evern subtracting what's been released already and dividing some necessary goods. These matrices actually helpful for different things if it is used properly and creatively, there are still many ways how to manipulate a matrix so that different outputs can be done and new concepts and innovations are attainable.

References

[1] D.J.D. Lopez. "Adamson University Computer Engineering Department Honor Code," AdU-CpE Departmental Policies, 2020.

Code Link

https://github.com/cherrylyncanoza/Linear-Algebra/tree/main/LinAlg_Lab6