Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 7

Matrix Operations

Submitted by:
Vallarta, Troy Joaquin G.

Instructor:
Engr. Dylan Josh D. Lopez

November 29, 2020

I. Objectives

This laboratory activity aims to implement the principles and techniques of matrix operations and learn the process of each matrix operation.

Methods

- The practice of this laboratory activity is learning how each matrix operations work.
    - The laboratory activity implies to teach the researchers to understand the matrix operations by doing it manually.
- The laboratory activity provides the researchers in learning an in-depth understanding of how matrix operations work.
    - The researchers achieve the functions by learning how each matrix operation works and learning about Dot product special properties

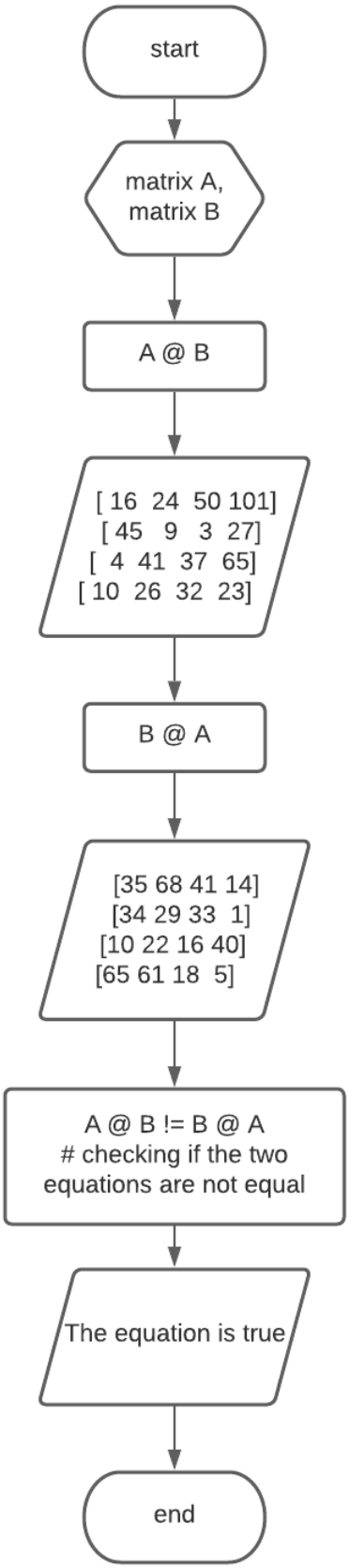

Figure 1. Matrix multiplication property 1

Figure 1 shows that the created flowchart shows the first matrix multiplication property of $A \ @ \ B \ ! = B \ @ \ A$, the first equation shows that the first matrix A is multiplied to the column of matrix B. The result of the first index of $A \ @ \ B$ will be a combination of the first **row** of matrix A and the first **column** of matric B. Therefore, the two equations would not be the same because it reverses its matrix.

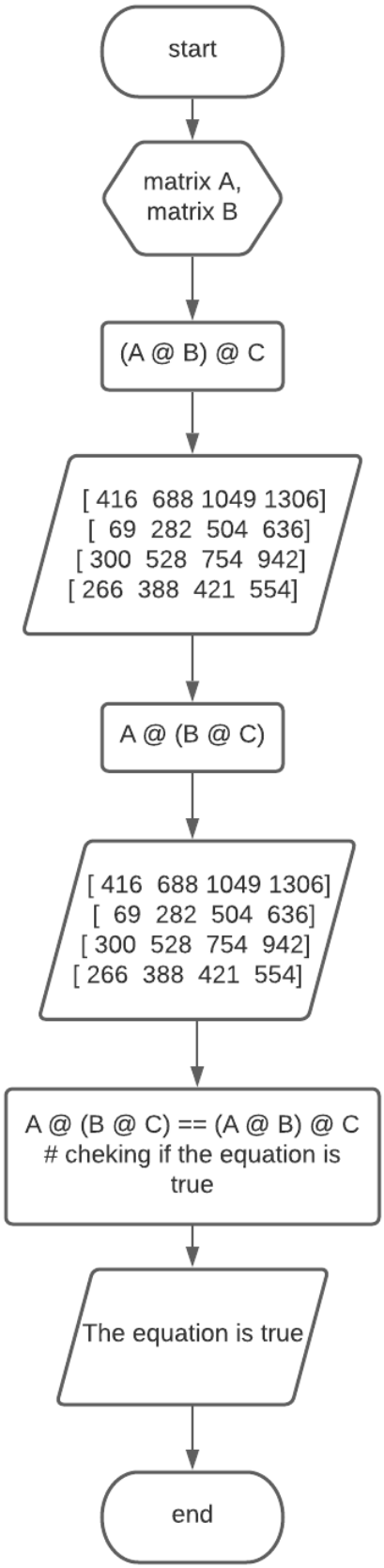

Figure 2. Matrix multiplication property 2

Figure 2 shows the 2nd multiplication property wherein $A @ (B @ C) == (A @ B) @ C$ it shows in this property that both created equations its result are the same. Its property is the same as commutative property in algebra. It shows that no matter what the order of multiplication in the matrix will not change the answer, but the matrix should not change the position of the matrix, for example, changing the position of matrix A to B. In this case, the matrices did not change the position, therefore, it will not change the result.

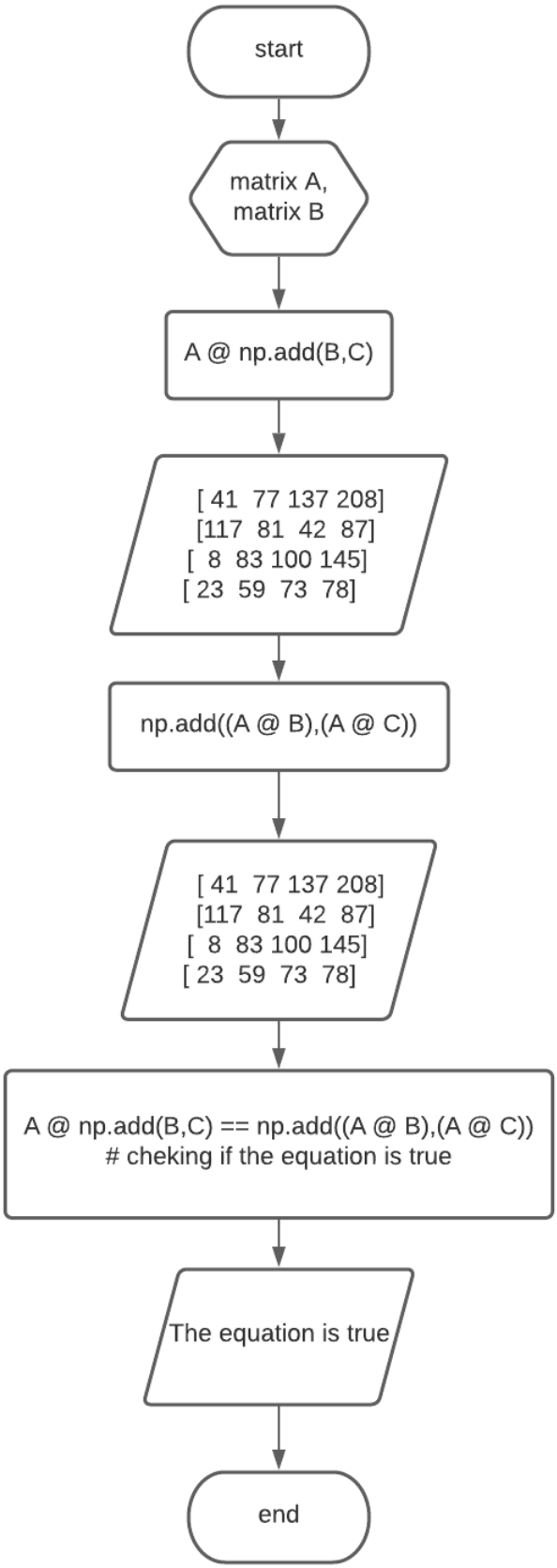

Figure 3. Matrix multiplication property 3

Figure 3 show the 3rd multiplication property of matrices, it show that in this equation $A @ np.add(B, C) == np.add((A@B), (A @ C))$ are the properties of distributive property in algebra. It shows that the first equation distrutes the matrix a to matrix B and C. The result is the same with the left side of the equation.

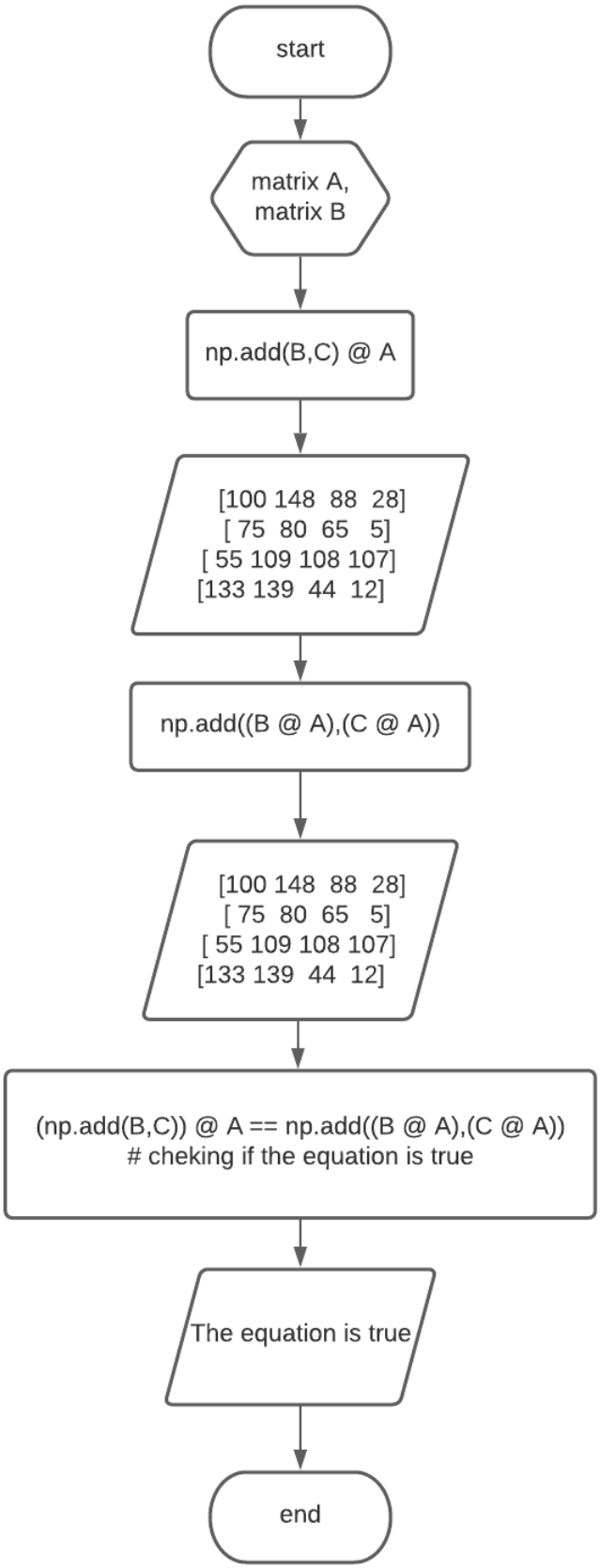

Figure 4. Matrix multiplication property 4

Figure 4 shows that the equation $np.add(B, C) \ @ \ A == np.add((B \ @ \ A), (C \ @ \ A))$ is the same property as figure 3 shown above. In this case, it shows that this is a distributive property. This is shown that Figure 3 & 4 does not have the same result. It combines the first multiplication property shown in figure 1 that the matrices cannot change places.

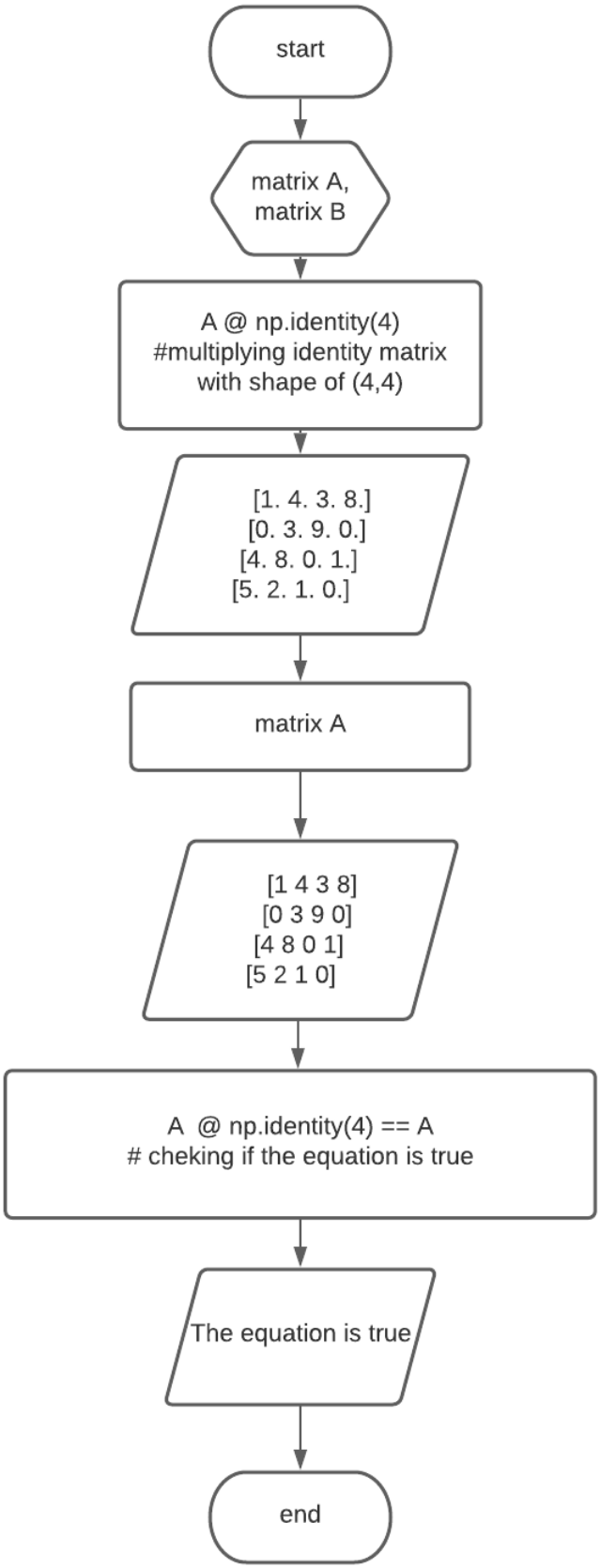

Figure 5. Matrix multiplication property 5

Figure 5 shows that the equation $A \ @ \ np.identity(4) == A$ is true, it shows that any inner product of matrices and an indentity matrix is always the same matrix. An identity matrix consist of $\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$ value one diagonaly and zero in between. Therefore any result of inner product in matrices, it will give the same matrix.

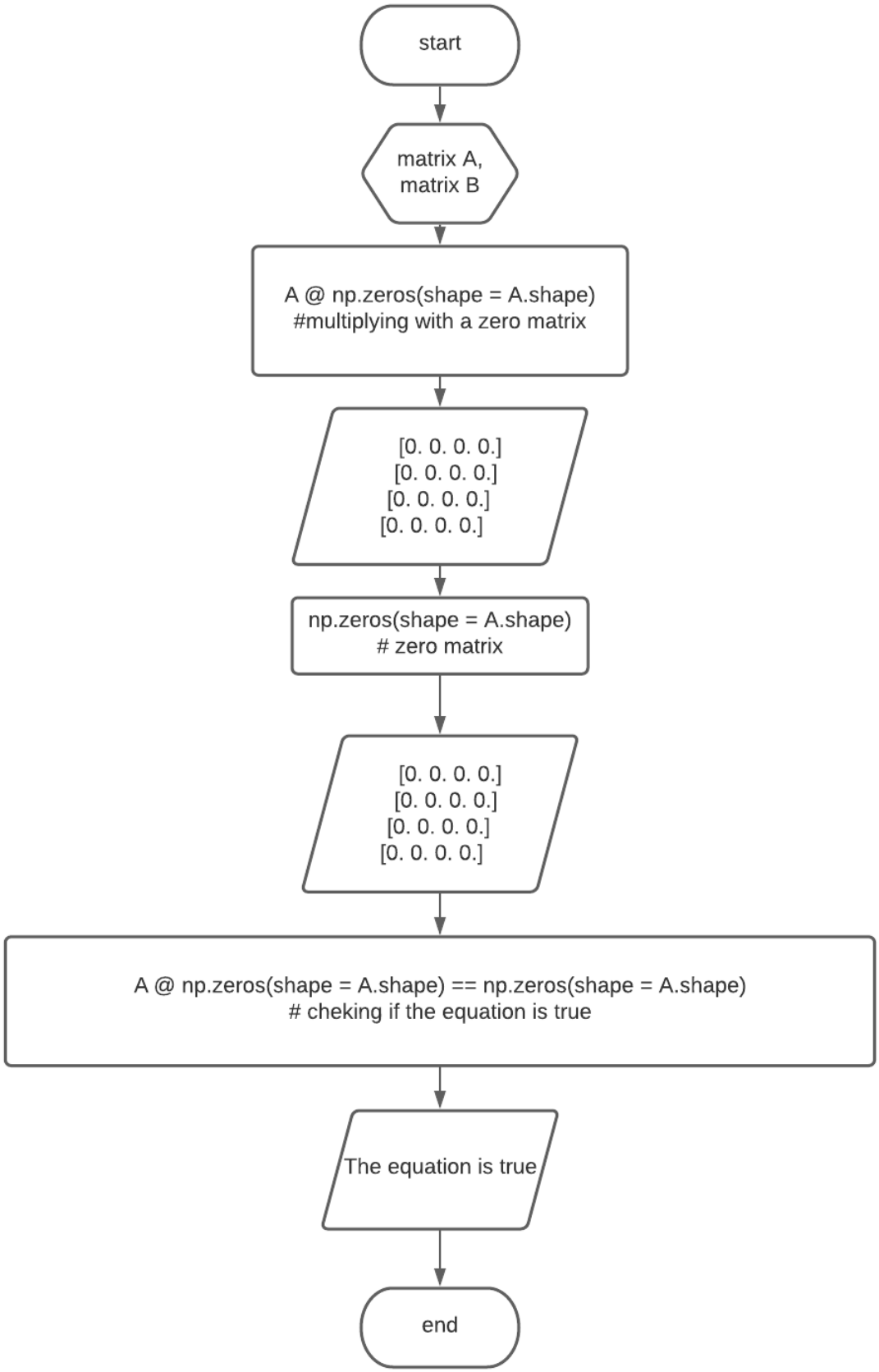

Figure 6. Matrix multiplication property 6

Figure 6 is the last multiplication property of the matrix and it shows any inner product of a matrix and a null matrix will result to a null matrix. A null matrix only consist of zeroes, therefore, any number multiplied with zero is still zero.

II. Results

```
A = np.array([
    [1,4,3,8],
    [0,3,9,0],
    [4,8,0,1],
    [5,2,1,0]
])
B = np.array([
    [1,4,6,2],
    [0,3,1,6],
    [5,0,0,1],
    [0,1,5,9]
])
C = np.array([
    [1,4,6,8],
    [0,3,4,5],
    [8,7,3,5],
    [0,2,7,8]
])
```

Figure 7. Initialization of matrices

Figure 7 shows the initialization of the matrix and to understand the results of the multiplication properties in matrices. It also shows that the matrices are in a shape of (4,4) because the laboratory activity clearly shows that the matrices should not be lower than (3,3).

```
print(A @ B ,'\n')
print(B @ A)
print("\n They not are Equal: \n")
print(A @ B != B @ A)
```

```
[[ 16  24  50 101]
 [ 45   9   3  27]
 [  4  41  37  65]
 [ 10  26  32  23]]

[[35 68 41 14]
 [34 29 33  1]
 [10 22 16 40]
 [65 61 18  5]]

 They not are Equal:

[[ True  True  True  True]
 [ True  True  True  True]
 [ True  True  True  True]
 [ True  True  True  True]]
```

Figure 8. Multiplication property 1

Figure 8 shows the first multiplication property, it shows in this code that they two equations are not equal with each other. The last matrix shown in figure one is the checking method, it prints in a Boolean data type if the results satisfy the argument it will return a True value and False if not.

```
# NUMBER 2
print(A @ (B @ C),'\n')
print((A @ B) @ C)
print("\n They are Equal: \n")
A @ (B @ C) == (A @ B) @ C
```

```
[[ 416  688 1049 1306]
 [  69  282  504  636]
 [ 300  528  754  942]
 [ 266  388  421  554]]

[[ 416  688 1049 1306]
 [  69  282  504  636]
 [ 300  528  754  942]
 [ 266  388  421  554]]

 They are Equal:


array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Figure 9. Multiplication property 2

Figure 9 shows the created equations are equal to each other. It also shows in the code that the argument is satisfied, therefore the code returns True value. It also shows the commutative property in matrices.

```
print(A @ np.add(B,C),'\n')
print(np.add((A @ B),(A @ C)))
print("\n They are Equal: \n")
A @ np.add(B,C) == np.add((A @ B),(A @ C))
```

```
[[ 41  77 137 208]
 [117  81  42  87]
 [  8  83 100 145]
 [ 23  59  73  78]]

[[ 41  77 137 208]
 [117  81  42  87]
 [  8  83 100 145]
 [ 23  59  73  78]]

 They are Equal:



array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Figure 10. Multiplication property 3

```
print((np.add(B,C) @ A), "\n")
print( np.add((B @ A),(C @ A)) )
print("\nEqual\n")
(np.add(B,C)) @ A == np.add((B @ A),(C @ A))
```

```
[[100 148  88  28]
 [ 75  80  65   5]
 [ 55 109 108 107]
 [133 139  44  12]]

[[100 148  88  28]
 [ 75  80  65   5]
 [ 55 109 108 107]
 [133 139  44  12]]

Equal
```

```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Figure 11. Multiplication property 4

Figure 10 and 11 shows how the distributive property works in matrices. In both figures, it shows that they don't have the same results because matrix A changed it place from the front (Figure 10) to the back (Figure 11). In this case, it applied the property of Figure 1 when Figure 10 and 11 compared to each other.

```
print(A @ np.identity(4), "\n")
print(A, "\n")
print("They are the same: \n")
A  @ np.identity(4) == A
```

```
[[1. 4. 3. 8.]
 [0. 3. 9. 0.]
 [4. 8. 0. 1.]
 [5. 2. 1. 0.]]

[[1 4 3 8]
 [0 3 9 0]
 [4 8 0 1]
 [5 2 1 0]]

They are the same:



array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Figure 12. Multiplication property 5

Figure 12 shows that when the inner multiplication of any matrix and an identity matrix, it will result to that said matrix. In this multiplication property, the code implemented a new function called np.identity(), in this function it will create an identity matrix, it only needs one paremeter to create the desired size matrix.

```
print(A @ np.zeros(shape = A.shape), "\n")
print(np.zeros(shape = A.shape), "\n")
print("\nThey are equal:\n")

A @ np.zeros(shape = A.shape) == np.zeros(shape = A.shape)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]


They are equal:


array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Figure 13. Multiplication property 6

Figure 13 shows any inner product of a matrix and null matrix, the result will be a null matrix. A null matrix consist of all zero, and any number multiplied by a zero is zero. The code used a new function called np.zeros(), its use is to create a zero matrix the only parameter it needs is the shape of the zero or null matrix.

III. Conclusion

As a computer engineering student, this laboratory activity is one of the most useful topic in linear algebra. It shows how the operations work in matrices, it also shows that using some operations in matrices are a little bit different from a addition, subtraction, multiplication, and division. Operations in matrices work in element-wise or other methods like dot product or inner product. In this laboratory activity, the researchers learned all about that.

The application of matrix operation in real-life has a wide uses, the first thing that came up with my mind is in creating video games, using matrices in creating a character data and especially in 3d creation. The other application of matrix operation is in healthcare situations. For example. [1] Healthcare problems are when a team is organized veritically (meaning hierarchy) it can get confusing for a people handling the healthcare. In this case, they used a matrix team in that has something like this Person in-charge = $\begin{bmatrix} Person\ A & Peson\ B \\ Person\ C & Person\ D \end{bmatrix}$. It can provide a fast collection of data. In example, person A is in-charge of a certain healthcare area, in this method everyone can work individually and fast to help the healthcare teams.

References

[1] “Matrix teams in healthcare,” *Global Integration*, 14-May-2020. [Online]. Available: https://www.global-integration.com/insights/matrix-teams-in-healthcare-2/. [Accessed: 13-Dec-2020].

Appendix

https://github.com/Zofserif/Linear-Algebra/blob/master/Lab7/Lab7.ipynb