Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 7

Matrix Operation

Submitted by:
Joaquin, Marc Christopher C.

Instructor:
Engr. Dylan Josh D. Lopez

December, 14, 2020

I. Objectives

This laboratory activity aims to implement the principles of operations of matrix algebra in engineering solutions to solve intermediate equations. and techniques of the matrix operation.

II. Methods

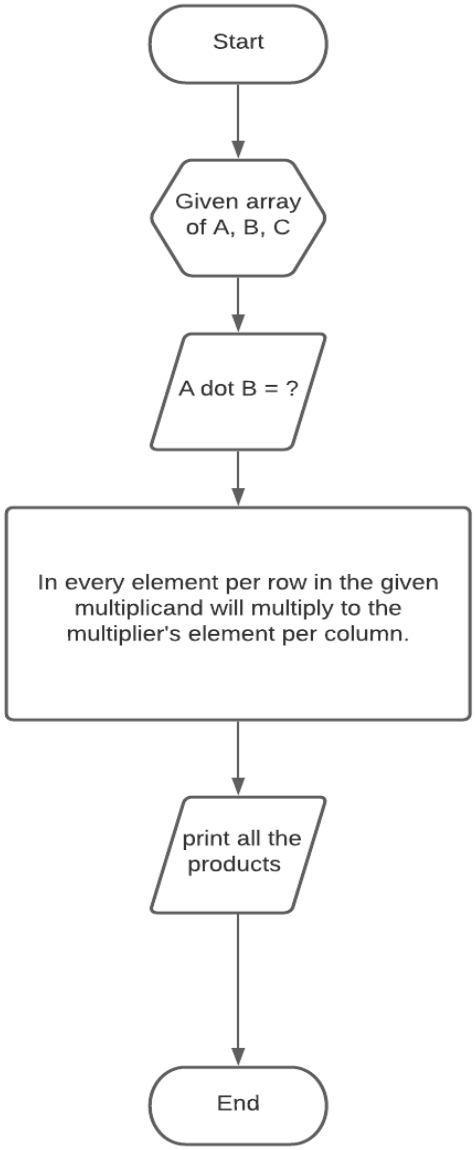

Figure 1: Property 1 Flowchart {A dot B)

In the first property, the programmer made his own formula to achieve the expected output. The strategy that he used is in every element of every row of A array will multiply to the B array to all elements per column. Once it has all the products on that row, it will all add

up and the output must have 3 elements in a row. Once he already got it, he proceeds to the next row of A array and multiply with the same procedure. That procedure will do until to the last row of A array. When he do the same procedure until to the last row, he came up for 9 elements overall. Thus, that's the output, which he achieved the expected result.

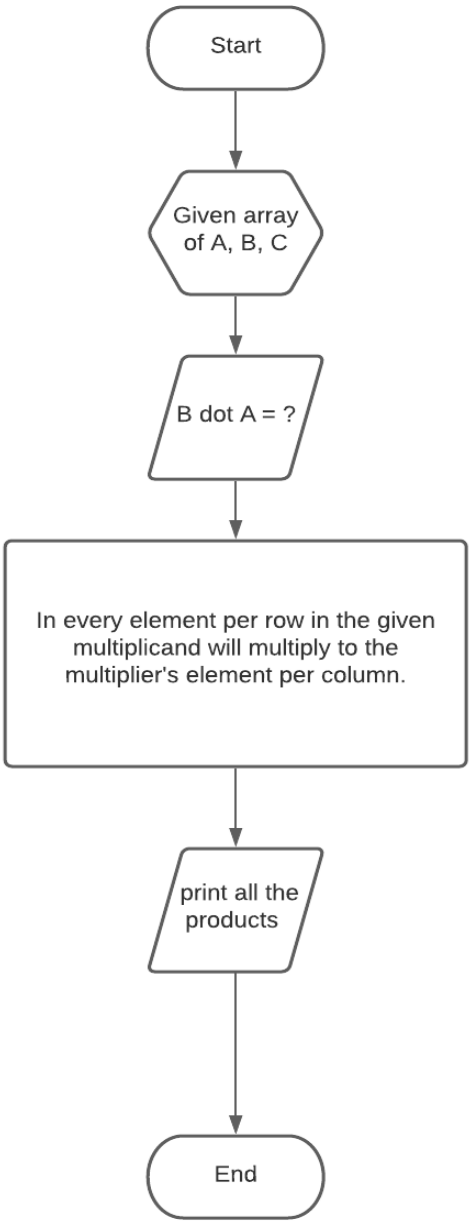

Figure 2: Property 1 Flowchart (B dot A)

The programmer did the same procedure yet he changed the position of the values. Once he did not change the position, the answer will be the same as A dot B, which is not

supposed to be equal to those values. Thus, he changed and the result he got the expected result. However, when you changed the position of every elements to the pattern, the answers will be totally different.

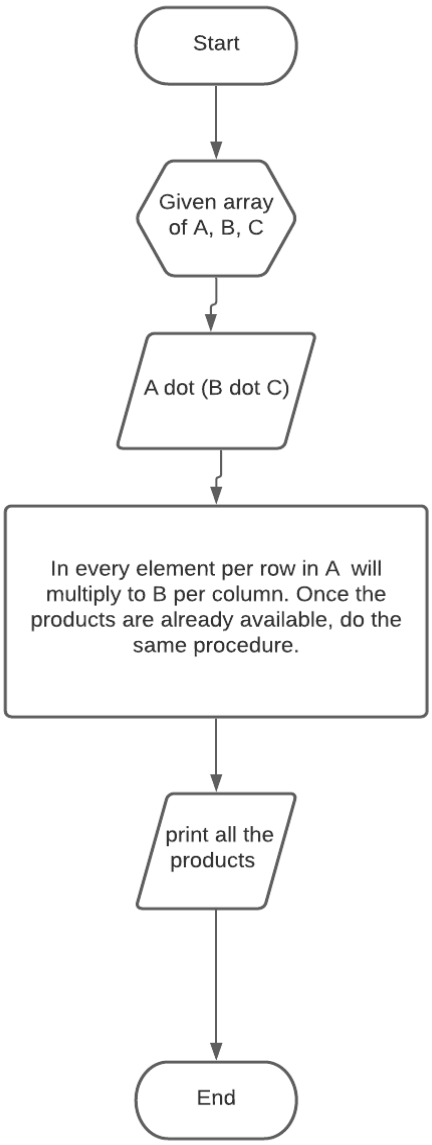

Figure 3: Property 2 Flowchart

The programmer used his built-in formula just like what he did to the first property. The catch in this property, the given is no longer only two only given array but three array. He multiply first the A array and B array and once it is all done, a new array will prosper and that new array will do the same procedure of multiplying towards to C and with that, he got the expected answer.

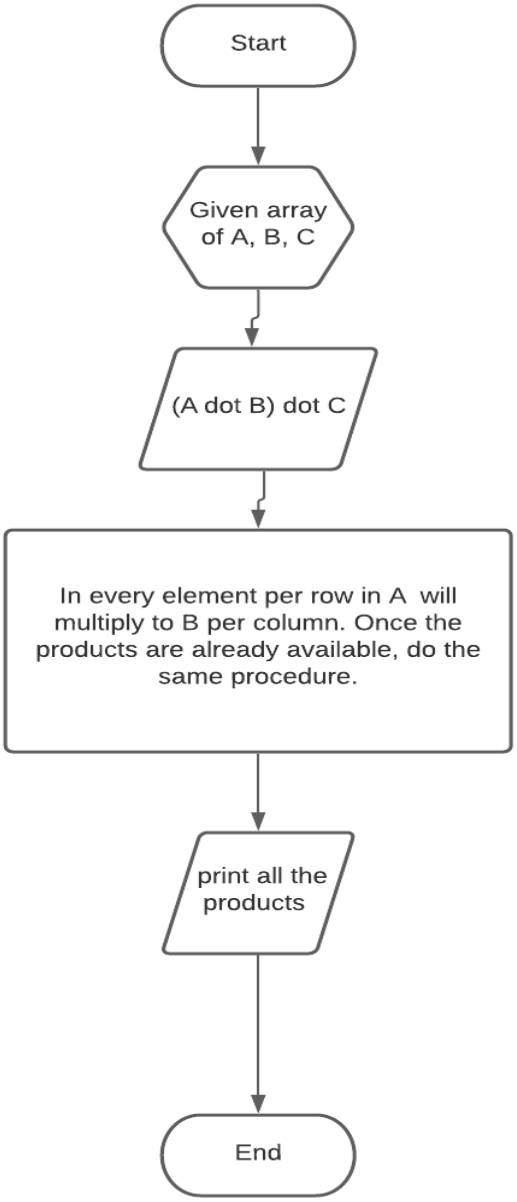

Figure 4: Property 2 Flowchart

The programmer did the same procedure like what he did in the first given of second property. He changed nothing there because if he try to change like the position, the answer will be different from the expected value. Thus, he stay with that position.

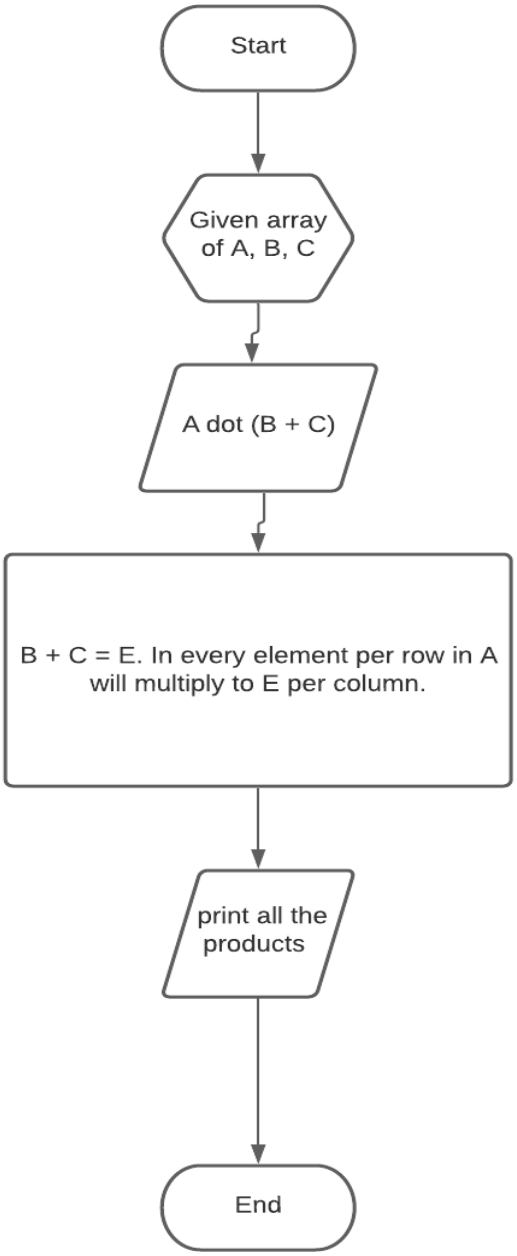

Figure 5: Property 3 Flowchart

The programmer did in this property, he add first all the arrays inside of the parenthesis. Once he got the sum of all the arrays, the new array served as the multiplier of A array and with that, the programmer achieved the expected result.

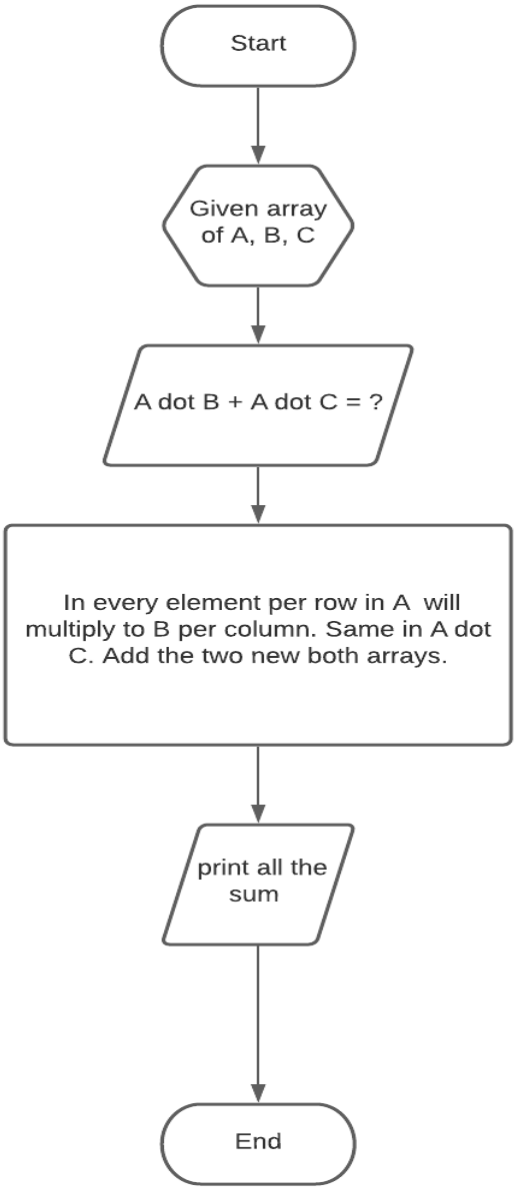

Figure 6: Property 3 Flowchart

In this scenario, the programmer got too much time on thinking how it will function yet, he made it through. What he did is to do his built-in multiplication on A and B & A and C. Once it is already finished, two new arrays are born. Those two arrays will add to each other and the output achieved the expected result of the programmer.

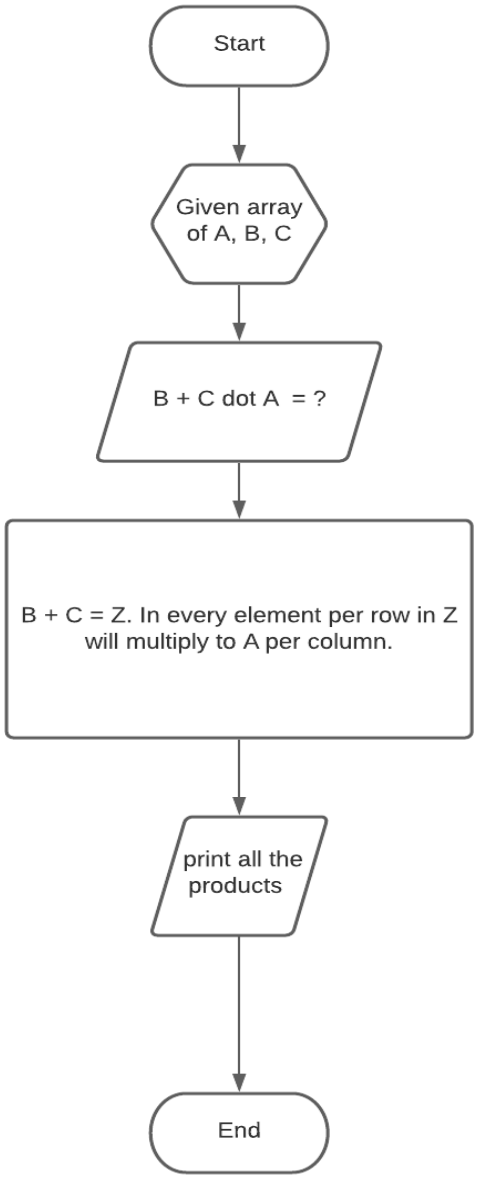

Figure 7: Property 4 Flowchart

The programmer just used the same procedure on what he did in property 3 (A dot(B +C)) yet the catch is he changed the position to his built-in multiplication method. As the result, the programmer achieved the expected output.

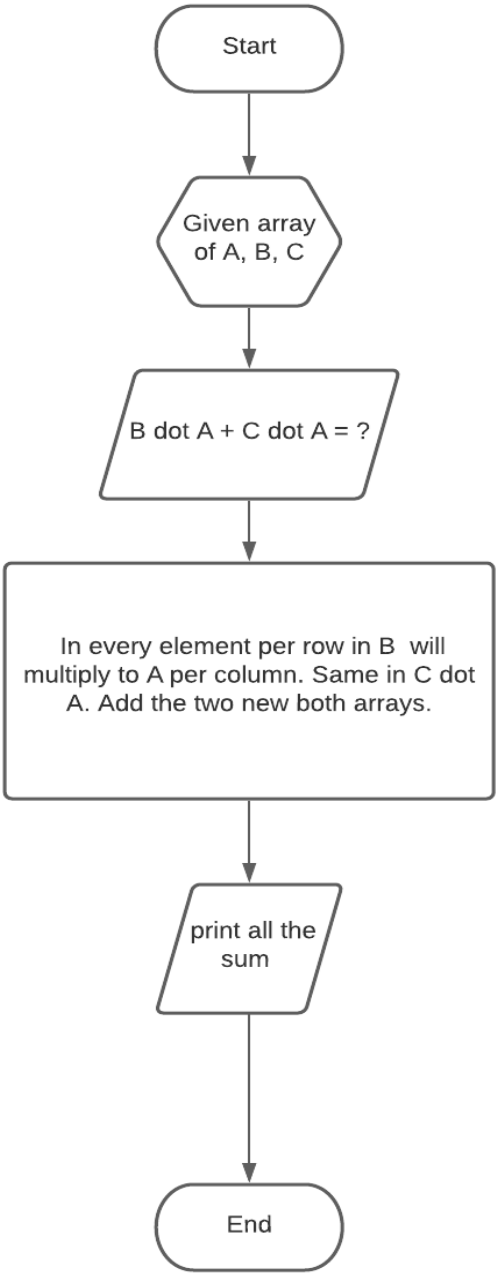

Figure 8: Property 4 Flowchart

The programmer just used the same procedure in property 3 (A dot B + A dot C)) yet the catch is he changed the position to his built-in multiplication method. If he don't change the position on his formula, the answer is too far compare to the expected output. As the result, the programmer achieved it.

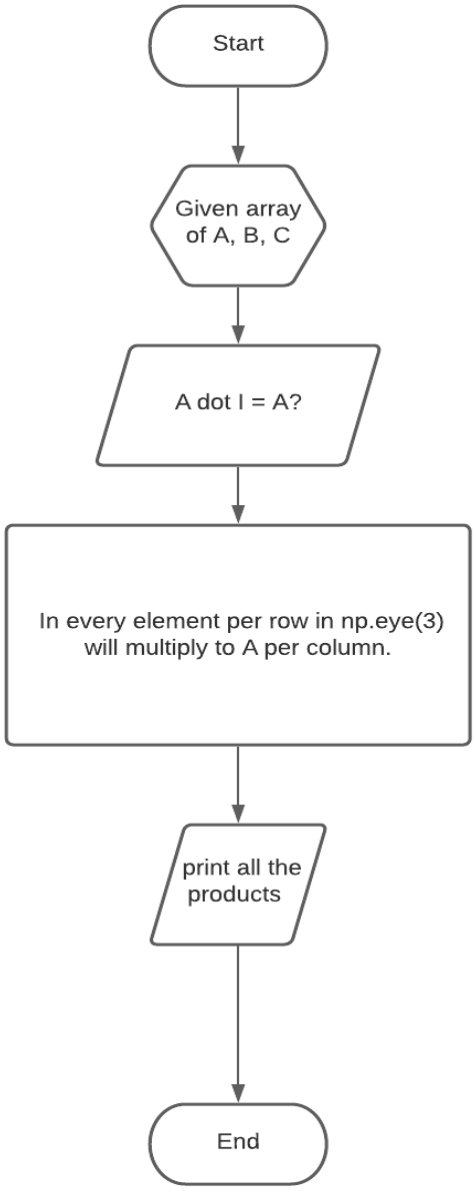

Figure 9: Property 5 Flowchart

In this property, I used *np.eye()* function in order to have an 1 diagonally in an array. This function return a 2-D array with ones on the diagonal and zeros elsewhere[1]. The next thing what the programmer did is he used the same procedure on other property, which is his built-in multiplication method. With that, he came up to the expected result.

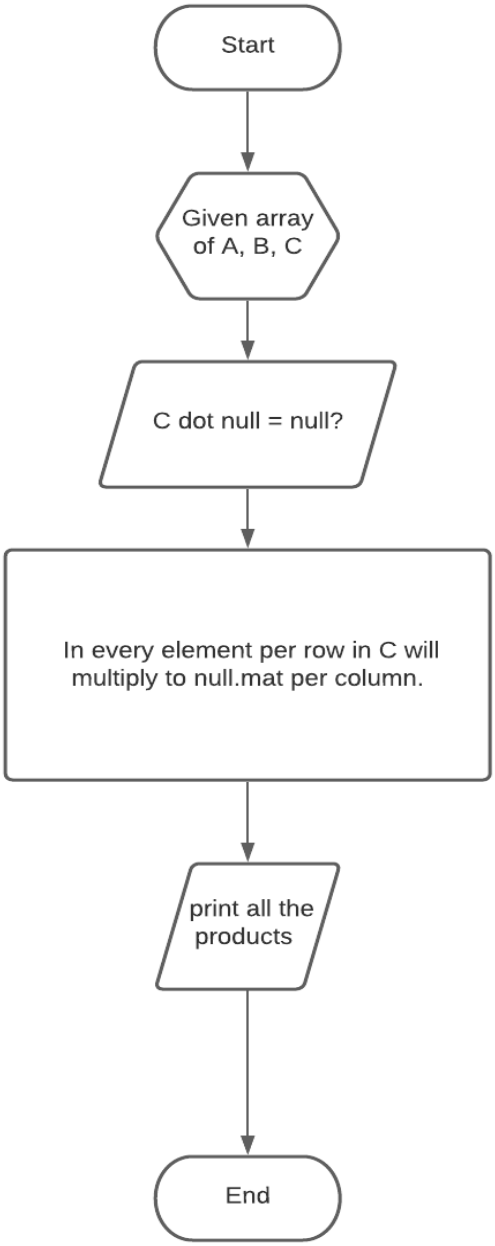

Figure 10: Property 6 Flowchart

The programmer did in this property is first to create zeros in all elements in an array. He used *np.zeros()* function for that, it Return a new array of given shape and type, filled with zeros[2]. The programmer did after that is used his built-in multiplication method, it served as the multiplier of C and the result achieved what he expected.

III. Results

In [23]:
```
#My function in A dot B
#first row of array
ABzerobyzeroelement = ((A[0,0] * B[0,0])+(A[0,1] * B[1,0])+(A[0,2]*B[2,0]))
#print(ABzerobyzeroelement)
ABzerobyoneelement = ((A[0,0] * B[0,1])+(A[0,1] * B[1,1])+(A[0,2]*B[2,1]))
#print(ABzerobyoneelement)
ABzerobytwoelement = ((A[0,0] * B[0,2])+(A[0,1] * B[1,2])+(A[0,2]*B[2,2]))
#print(ABzerobytwoelement)

#second row of array
ABonebyzeroelement = ((A[1,0] * B[0,0])+(A[1,1] * B[1,0])+(A[1,2]*B[2,0]))
#print(ABonebyzeroelement)
ABonebyoneelement = ((A[1,0] * B[0,1])+(A[1,1] * B[1,1])+(A[1,2]*B[2,1]))
#print(ABonebyoneelement)
ABonebytwoelement = ((A[1,0] * B[0,2])+(A[1,1] * B[1,2])+(A[1,2]*B[2,2]))
#print(ABonebytwoelement)

#third row of array
ABtwobyzeroelement = ((A[2,0] * B[0,0])+(A[2,1] * B[1,0])+(A[2,2]*B[2,0]))
#print(ABtwobyzeroelement)
ABtwobyoneelement = ((A[2,0] * B[0,1])+(A[2,1] * B[1,1])+(A[2,2]*B[2,1]))
#print(ABtwobyoneelement)
ABtwobytwoelement = ((A[2,0] * B[0,2])+(A[2,1] * B[1,2])+(A[2,2]*B[2,2]))
#print(ABtwobytwoelement)

print(f'The product of A and B are:\n [{[ABzerobyzeroelement, ABzerobyoneelement, ABzerobytwoelement]},\n {[ABonebyzeroelement, ABonebyoneelement, ABonebytwoelement]},\n {[ABtwobyzeroelement, ABtwobyoneelement, ABtwobytwoelement]}]')
```

```
23
The product of A and B are:
 [[10, 36, 23],
 [28, 87, 53],
 [46, 138, 83]]
```

Figure 11: Property 1 A dot B Proof (Programmer's Built in Multiplication

In [34]:
```
#My function in B dot A
#first row of array
BAzerobyzeroelement = ((B[0,0] * A[0,0])+(B[0,1] * A[1,0])+(B[0,2]*A[2,0]))
#print(BAzerobyzeroelement)
BAzerobyoneelement = ((B[0,0] * A[0,1])+(B[0,1] * A[1,1])+(B[0,2]*A[2,1]))
#print(BAzerobyoneelement)
BAzerobytwoelement = ((B[0,0] * A[0,2])+(B[0,1] * A[1,2])+(B[0,2]*A[2,2]))
#print(BAzerobytwoelement)

#second row of array
BAonebyzeroelement = ((B[1,0] * A[0,0])+(B[1,1] * A[1,0])+(B[1,2]*A[2,0]))
#print(BAonebyzeroelement)
BAonebyoneelement = ((B[1,0] * A[0,1])+(B[1,1] * A[1,1])+(B[1,2]*A[2,1]))
#print(BAonebyoneelement)
BAonebytwoelement = ((B[1,0] * A[0,2])+(B[1,1] * A[1,2])+(B[1,2]*A[2,2]))
#print(BAonebytwoelement)

#third row of array
BAtwobyzeroelement = ((B[2,0] * A[0,0])+(B[2,1] * A[1,0])+(B[2,2]*A[2,0]))
#print(BAtwobyzeroelement)
BAtwobyoneelement = ((B[2,0] * A[0,1])+(B[2,1] * A[1,1])+(B[2,2]*A[2,1]))
#print(BAtwobyoneelement)
BAtwobytwoelement = ((B[2,0] * A[0,2])+(B[2,1] * A[1,2])+(B[2,2]*A[2,2]))
#print(BAtwobytwoelement)

print(f'The product of B and A are:\n [{[BAzerobyzeroelement, BAzerobyoneelement, BAzerobytwoelement]},\n {[BAonebyzeroelement, BAonebyoneelement, BAonebytwoelement]},\n {[BAtwobyzeroelement, BAtwobyoneelement, BAtwobytwoelement]}]')
```

```
The product of B and A are:
 [[33, 42, 51],
 [51, 63, 75],
 [60, 72, 84]]
```

Figure 12: Property 1 B dot A Proof (Programmer's Built in Multiplication Method)

Property 1. $\boldsymbol{A \cdot B \neq B \cdot A}$

[4]:
```
np.dot(A,B)
```

[4]:
```
array([[ 10,  36,  23],
       [ 28,  87,  53],
       [ 46, 138,  83]])
```

[5]:
```
np.dot(B,A)
```

[5]:
```
array([[33, 42, 51],
       [51, 63, 75],
       [60, 72, 84]])
```

[6]:
```
np.dot(A,B) == np.dot(B,A)
```

[6]:
```
array([[False, False, False],
       [False, False, False],
       [False, False, False]])
```

Figure 13: Property 1 Proof (Numpy Multiplication)

In this property, as the reader will see, when the programmer did not used numpy multiplication method, he had coded a lot of operations in order to achieve the answer given in the expected result in *Figure 13*. It isw much easier to code with Numpy Multiplication rather than doing it manually.

In [40]:
```
#My function in A dot (B dot C)
#first row of array
BCAzerobyzeroelement = ((ABzerobyzeroelement * C[0,0])+(ABzerobyoneelement * C[1,0])+(ABzerobytwoelement *C[2,0]))
#print(BCAzerobyzeroelement)
BCAzerobyoneelement = ((ABzerobyzeroelement * C[0,1])+(ABzerobyoneelement * C[1,1])+(ABzerobytwoelement*C[2,1]))
#print(BCAzerobyoneelement)
BCAzerobytwoelement = ((ABzerobyzeroelement * C[0,2])+(ABzerobyoneelement * C[1,2])+(ABzerobytwoelement*C[2,2]))
#print(BCAzerobytwoelement)

#second row of array
BCAonebyzeroelement = ((ABonebyzeroelement * C[0,0])+(ABonebyoneelement * C[1,0])+(ABonebytwoelement*C[2,0]))
#print(BCAonebyzeroelement)
BCAonebyoneelement = ((ABonebyzeroelement * C[0,1])+(ABonebyoneelement * C[1,1])+(ABonebytwoelement*C[2,1]))
#print(BCAonebyoneelement)
BCAonebytwoelement = ((ABonebyzeroelement * C[0,2])+(ABonebyoneelement * C[1,2])+(ABonebytwoelement*C[2,2]))
#print(BCAonebytwoelement)

#third row of array
BCAtwobyzeroelement = ((ABtwobyzeroelement * C[0,0])+(ABtwobyoneelement * C[1,0])+(ABtwobytwoelement*C[2,0]))
#print(BCAtwobyzeroelement)
BCAtwobyoneelement = ((ABtwobyzeroelement * C[0,1])+(ABtwobyoneelement * C[1,1])+(ABtwobytwoelement*C[2,1]))
#print(BCAtwobyoneelement)
BCAtwobytwoelement = ((ABtwobyzeroelement * C[0,2])+(ABtwobyoneelement * C[1,2])+(ABtwobytwoelement*C[2,2]))
#print(BCAtwobytwoelement)

print(f'The product of A and B are:\n [{[BCAzerobyzeroelement, BCAzerobyoneelement, BCAzerobytwoelement]},\n {[BCAonebyzeroelement, BCAonebyoneelement, BCAonebytwoelement]},\n {[BCAtwobyzeroelement, BCAtwobyoneelement, BCAtwobytwoelement]}]')
```

```
The product of A and B are:
 [[175, 352, 421],
 [442, 880, 1030],
 [709, 1408, 1639]]
```

Figure 14: Property 2 A dot B dot C Proof (Programmer's Built in Multiplication Method)

```
#My function in (A dot B) dot C
#first row of array
ABCzerobyzeroelement = ((ABzerobyzeroelement * C[0,0])+(ABzerobyoneelement * C[1,0])+(ABzerobytwoelement *C[2,0]))
#print(ABzerobyzeroelement)
ABCzerobyoneelement = ((ABzerobyzeroelement * C[0,1])+(ABzerobyoneelement * C[1,1])+(ABzerobytwoelement*C[2,1]))
#print(ABzerobyoneelement)
ABCzerobytwoelement = ((ABzerobyzeroelement * C[0,2])+(ABzerobyoneelement * C[1,2])+(ABzerobytwoelement*C[2,2]))
#print(ABzerobytwoelement)

#second row of array
ABConebyzeroelement = ((ABonebyzeroelement * C[0,0])+(ABonebyoneelement * C[1,0])+(ABonebytwoelement*C[2,0]))
#print(ABonebyzeroelement)
ABConebyoneelement = ((ABonebyzeroelement * C[0,1])+(ABonebyoneelement * C[1,1])+(ABonebytwoelement*C[2,1]))
#print(ABonebyoneelement)
ABConebytwoelement = ((ABonebyzeroelement * C[0,2])+(ABonebyoneelement * C[1,2])+(ABonebytwoelement*C[2,2]))
#print(ABonebytwoelement)

#third row of array
ABCtwobyzeroelement = ((ABtwobyzeroelement * C[0,0])+(ABtwobyoneelement * C[1,0])+(ABtwobytwoelement*C[2,0]))
#print(ABtwobyzeroelement)
ABCtwobyoneelement = ((ABtwobyzeroelement * C[0,1])+(ABtwobyoneelement * C[1,1])+(ABtwobytwoelement*C[2,1]))
#print(ABtwobyoneelement)
ABCtwobytwoelement = ((ABtwobyzeroelement * C[0,2])+(ABtwobyoneelement * C[1,2])+(ABtwobytwoelement*C[2,2]))
#print(ABtwobytwoelement)

print(f'The product of A and B are:\n [{[ABCzerobyzeroelement, ABCzerobyoneelement, ABCzerobytwoelement]},\n {[ABConebyzeroelement, ABConebyoneelement, ABConebytwoelement]},\n {[ABCtwobyzeroelement, ABCtwobyoneelement, ABCtwobytwoelement]}]')
```

```
The product of A and B are:
 [[175, 352, 421],
 [442, 880, 1030],
 [709, 1408, 1639]]
```

Figure 15: Property 2 A dot B dot C Proof (Programmer's Built in Multiplication Method)

Property 2. $\boldsymbol{A} \cdot (\boldsymbol{B} \cdot \boldsymbol{C}) = (\boldsymbol{A} \cdot \boldsymbol{B}) \cdot \boldsymbol{C}$

```
[7]: A @ (B @ C)
```

```
[7]: array([[ 175,  352,  421],
            [ 442,  880, 1030],
            [ 709, 1408, 1639]])
```

```
[8]: (A @ B) @ C
```

```
[8]: array([[ 175,  352,  421],
            [ 442,  880, 1030],
            [ 709, 1408, 1639]])
```

```
[9]: A @ (B @ C) == (A @ B) @ C
```

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

Figure 16: Property 2 Proof (Numpy Multiplication)

As the reader can see, the programmer made his built in method doubled for the property has three array that must be multiplied. Thus, the long method that he did made his code more longer and complicated unlike with numpy multiplication, with just one line, it achieved the expected and precise output.

```
#For A at (B plus C)
#add B and C
E = B+C
print(f'The sum of B and C are:\n{E}')
#A at BC
#first row of array
AEzerobyzeroelement = ((A[0,0] * E[0,0])+(A[0,1] * E[1,0])+(A[0,2]*E[2,0]))
#print(AEzerobyzeroelement)
AEzerobyoneelement = ((A[0,0] * E[0,1])+(A[0,1] * E[1,1])+(A[0,2]*E[2,1]))
#print(AEzerobyoneelement)
AEzerobytwoelement = ((A[0,0] * E[0,2])+(A[0,1] * E[1,2])+(A[0,2]*E[2,2]))
#print(AEzerobytwoelement)

#second row of array
AEonebyzeroelement = ((A[1,0] * E[0,0])+(A[1,1] * E[1,0])+(A[1,2]*E[2,0]))
#print(AEonebyzeroelement)
AEonebyoneelement = ((A[1,0] * E[0,1])+(A[1,1] * E[1,1])+(A[1,2]*E[2,1]))
#print(AEonebyoneelement)
AEonebytwoelement = ((A[1,0] * E[0,2])+(A[1,1] * E[1,2])+(A[1,2]*E[2,2]))
#print(AEonebytwoelement)

#third row of array
AEtwobyzeroelement = ((A[2,0] * E[0,0])+(A[2,1] * E[1,0])+(A[2,2]*E[2,0]))
#print(AEtwobyzeroelement)
AEtwobyoneelement = ((A[2,0] * E[0,1])+(A[2,1] * E[1,1])+(A[2,2]*E[2,1]))
#print(AEtwobyoneelement)
AEtwobytwoelement = ((A[2,0] * E[0,2])+(A[2,1] * E[1,2])+(A[2,2]*E[2,2]))
#print(AEtwobytwoelement)

print(f'The product of A and E are:\n [{[AEzerobyzeroelement, AEzerobyoneelement, AEzerobytwoelement]},\n {[AEonebyzeroelement, AEonebyoneelement, AEonebytwoelement]},\n {[AEtwobyzeroelement, AEtwobyoneelement, AEtwobytwoelement]}]')
```

```
The sum of B and C are:
[[ 9 15  9]
 [ 5  9 10]
 [ 3 10 11]]
The product of A and E are:
 [[28, 63, 62],
 [79, 165, 152],
 [130, 267, 242]]
```

Figure 15: Property 3 A dot B plus C Proof (Programmer's Built in Multiplication Method)

```
#For the product of A and B add to the product of A and C
#Product of A and B
#first row of array
ABzerobyzeroelement = ((A[0,0] * B[0,0])+(A[0,1] * B[1,0])+(A[0,2]*B[2,0]))
#print(ABzerobyzeroelement)
ABzerobyoneelement = ((A[0,0] * B[0,1])+(A[0,1] * B[1,1])+(A[0,2]*B[2,1]))
#print(ABzerobyoneelement)
ABzerobytwoelement = ((A[0,0] * B[0,2])+(A[0,1] * B[1,2])+(A[0,2]*B[2,2]))
#print(ABzerobytwoelement)

#second row of array
ABonebyzeroelement = ((A[1,0] * B[0,0])+(A[1,1] * B[1,0])+(A[1,2]*B[2,0]))
#print(ABonebyzeroelement)
ABonebyoneelement = ((A[1,0] * B[0,1])+(A[1,1] * B[1,1])+(A[1,2]*B[2,1]))
#print(ABonebyoneelement)
ABonebytwoelement = ((A[1,0] * B[0,2])+(A[1,1] * B[1,2])+(A[1,2]*B[2,2]))
#print(ABonebytwoelement)

#third row of array
ABtwobyzeroelement = ((A[2,0] * B[0,0])+(A[2,1] * B[1,0])+(A[2,2]*B[2,0]))
#print(ABtwobyzeroelement)
ABtwobyoneelement = ((A[2,0] * B[0,1])+(A[2,1] * B[1,1])+(A[2,2]*B[2,1]))
#print(ABtwobyoneelement)
ABtwobytwoelement = ((A[2,0] * B[0,2])+(A[2,1] * B[1,2])+(A[2,2]*B[2,2]))
#print(ABtwobytwoelement)

print(f'The product of A and B are:\n [{[ABzerobyzeroelement, ABzerobyoneelement, ABzerobytwoelement]},\n {[ABonebyzeroelement, ABonebyoneelement, ABonebytwoelement]},\n {[ABtwobyzeroelement, ABtwobyoneelement, ABtwobytwoelement]}]')

#Product of A and C
#first row of array
AClerobyzeroelement = ((A[0,0] * C[0,0])+(A[0,1] * C[1,0])+(A[0,2]*C[2,0]))
#print(AClerobyzeroelement)
AClerobyoneelement = ((A[0,0] * C[0,1])+(A[0,1] * C[1,1])+(A[0,2]*C[2,1]))
#print(AClerobyoneelement)
AClerobytwoelement = ((A[0,0] * C[0,2])+(A[0,1] * C[1,2])+(A[0,2]*C[2,2]))
#print(AClerobytwoelement)

#second row of array
AConebyzeroelement = ((A[1,0] * C[0,0])+(A[1,1] * C[1,0])+(A[1,2]*C[2,0]))
#print(AConebyzeroelement)
AConebyoneelement = ((A[1,0] * C[0,1])+(A[1,1] * C[1,1])+(A[1,2]*C[2,1]))
#print(AConebyoneelement)
AConebytwoelement = ((A[1,0] * C[0,2])+(A[1,1] * C[1,2])+(A[1,2]*C[2,2]))
#print(AConebytwoelement)

#third row of array
ACtwobyzeroelement = ((A[2,0] * C[0,0])+(A[2,1] * C[1,0])+(A[2,2]*C[2,0]))
#print(ACtwobyzeroelement)
ACtwobyoneelement = ((A[2,0] * C[0,1])+(A[2,1] * C[1,1])+(A[2,2]*C[2,1]))
#print(ACtwobyoneelement)
ACtwobytwoelement = ((A[2,0] * C[0,2])+(A[2,1] * C[1,2])+(A[2,2]*C[2,2]))
#print(ACtwobytwoelement)

print(f'The product of A and C are:\n [{[AClerobyzeroelement, AClerobyoneelement, AClerobytwoelement]},\n {[AConebyzeroelement, AConebyoneelement, AConebytwoelement]},\n {[ACtwobyzeroelement, ACtwobyoneelement, ACtwobytwoelement]}]')
```

Figure 16: Property 3 A and B plus A and C Proof (Programmer's Built in Multiplication Method)

```
#For the sum
SUMone = ABzerobyzeroelement + AClerobyzeroelement
SUMtwo = ABzerobyoneelement + AClerobyoneelement
SUMthree = ABzerobytwoelement + AClerobytwoelement
SUMfour = ABonebyzeroelement + AConebyzeroelement
SUMfive = ABonebyoneelement + AConebyoneelement
SUMsix = ABonebytwoelement + AConebytwoelement
SUMseven = ABtwobyzeroelement + ACtwobyzeroelement
SUMeight = ABtwobyoneelement + ACtwobyoneelement
SUMnine = ABtwobyoneelement + ACtwobytwoelement

print(f'Overall are:\n [{[SUMone, SUMtwo, SUMthree]},\n {[SUMfour, SUMfive, SUMsix]},\n {[SUMseven, SUMeight, SUMnine]}]')
```

```
The product of A and B are:
 [[10, 36, 23],
 [28, 87, 53],
 [46, 138, 83]]
The product of A and C are:
 [[18, 27, 39],
 [51, 78, 99],
 [84, 129, 159]]
Overall are:
 [[28, 63, 62],
 [79, 165, 152],
 [130, 267, 297]]
```

Figure 17: Property 3 A and B plus A and C Proof (Programmer's Built in Multiplication Method)

Property 3. $A \cdot (B + C) = A \cdot B + A \cdot C$

```
[10]: A @ (B+C)

[10]: array([[ 28,  63,  62],
             [ 79, 165, 152],
             [130, 267, 242]])

[11]: A@B + A@C

[11]: array([[ 28,  63,  62],
             [ 79, 165, 152],
             [130, 267, 242]])

[13]: (A @ (B+C)) == (A@B + A@C)

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

Figure 18: Property 3 Proof (Numpy Multiplication)

The programmer took so much time in building his own formula in this property. He concludes that this is the proof that it is much more covinient using numpy multiplication rather than doing it manually.

```
#Add B and C
Z = B+C
print(f'The sum of B and C are:\n{E}')

#Z at A
#first row of array
ZAzerobyzeroelement = ((Z[0,0] * A[0,0])+(Z[0,1] * A[1,0])+(Z[0,2]*A[2,0]))
#print(ZAzerobyzeroelement)
ZAzerobyoneelement = ((Z[0,0] * A[0,1])+(Z[0,1] * A[1,1])+(Z[0,2]*A[2,1]))
#print(ZAzerobyoneelement)
ZAzerobytwoelement = ((Z[0,0] * A[0,2])+(Z[0,1] * A[1,2])+(Z[0,2]*A[2,2]))
#print(ZAzerobytwoelement)

#second row of array
ZAonebyzeroelement = ((Z[1,0] * A[0,0])+(Z[1,1] * A[1,0])+(Z[1,2]*A[2,0]))
#print(ZAonebyzeroelement)
ZAonebyoneelement = ((Z[1,0] * A[0,1])+(Z[1,1] * A[1,1])+(Z[1,2]*A[2,1]))
#print(ZAonebyoneelement)
ZAonebytwoelement = ((Z[1,0] * A[0,2])+(Z[1,1] * A[1,2])+(Z[1,2]*A[2,2]))
#print(ZAonebytwoelement)

#third row of array
ZAtwobyzeroelement = ((Z[2,0] * A[0,0])+(Z[2,1] * A[1,0])+(Z[2,2]*A[2,0]))
#print(ZAtwobyzeroelement)
ZAtwobyoneelement = ((Z[2,0] * A[0,1])+(Z[2,1] * A[1,1])+(Z[2,2]*A[2,1]))
#print(ZAtwobyoneelement)
ZAtwobytwoelement = ((Z[2,0] * A[0,2])+(Z[2,1] * A[1,2])+(Z[2,2]*A[2,2]))
#print(ZAtwobytwoelement)

print(f'The product of Z and A are:\n [{[ZAzerobyzeroelement, ZAzerobyoneelement, ZAzerobytwoelement]},\n {[ZAonebyzeroelement, ZAonebyoneelement, ZAonebytwoelement]},\n {[ZAtwobyzeroelement, ZAtwobyoneelement, ZAtwobytwoelement]}]')
```

```
The sum of B and C are:
[[ 9 15  9]
 [ 5  9 10]
 [ 3 10 11]]
The product of Z and A are:
 [[132, 165, 198],
 [111, 135, 159],
 [120, 144, 168]]
```

Figure 19: Property 4 B plus C dot A Proof (Programmer's Built in Multiplication Method)

```
#For the product of B and A add to the product of C and A
#Product of B and A
#first row of array
BAzerobyzeroelement = ((B[0,0] * A[0,0])+(B[0,1] * A[1,0])+(B[0,2]*A[2,0]))
#print(BAzerobyzeroelement)
BAzerobyoneelement = ((B[0,0] * A[0,1])+(B[0,1] * A[1,1])+(B[0,2]*A[2,1]))
#print(BAzerobyoneelement)
BAzerobytwoelement = ((B[0,0] * A[0,2])+(B[0,1] * A[1,2])+(B[0,2]*A[2,2]))
#print(BAzerobytwoelement)

#second row of array
BAonebyzeroelement = ((B[1,0] * A[0,0])+(B[1,1] * A[1,0])+(B[1,2]*A[2,0]))
#print(BAonebyzeroelement)
BAonebyoneelement = ((B[1,0] * A[0,1])+(B[1,1] * A[1,1])+(B[1,2]*A[2,1]))
#print(BAonebyoneelement)
BAonebytwoelement = ((B[1,0] * A[0,2])+(B[1,1] * A[1,2])+(B[1,2]*A[2,2]))
#print(BAonebytwoelement)

#third row of array
BAtwobyzeroelement = ((B[2,0] * A[0,0])+(B[2,1] * A[1,0])+(B[2,2]*A[2,0]))
#print(BAtwobyzeroelement)
BAtwobyoneelement = ((B[2,0] * A[0,1])+(B[2,1] * A[1,1])+(B[2,2]*A[2,1]))
#print(BAtwobyoneelement)
BAtwobytwoelement = ((B[2,0] * A[0,2])+(B[2,1] * A[1,2])+(B[2,2]*A[2,2]))
#print(BAtwobytwoelement)

print(f'The product of B and A are:\n [{[BAzerobyzeroelement, BAzerobyoneelement, BAzerobytwoelement]},\n {[BAonebyzeroelement, BAonebyoneelement, BAonebytwoelement]},\n {[BAtwobyzeroelement, BAtwobyoneelement, BAtwobytwoelement]}]')

#Product of C and A
#first row of array
CAzerobyzeroelement = ((C[0,0] * A[0,0])+(C[0,1] * A[1,0])+(C[0,2]*A[2,0]))
#print(CAzerobyzeroelement)
CAzerobyoneelement = ((C[0,0] * A[0,1])+(C[0,1] * A[1,1])+(C[0,2]*A[2,1]))
#print(CAzerobyoneelement)
CAzerobytwoelement = ((C[0,0] * A[0,2])+(C[0,1] * A[1,2])+(C[0,2]*A[2,2]))
#print(CAzerobytwoelement)

#second row of array
CAonebyzeroelement = ((C[1,0] * A[0,0])+(C[1,1] * A[1,0])+(C[1,2]*A[2,0]))
#print(CAonebyzeroelement)
CAonebyoneelement = ((C[1,0] * A[0,1])+(C[1,1] * A[1,1])+(C[1,2]*A[2,1]))
#print(CAonebyoneelement)
CAonebytwoelement = ((C[1,0] * A[0,2])+(C[1,1] * A[1,2])+(C[1,2]*A[2,2]))
#print(CAonebytwoelement)

#third row of array
CAtwobyzeroelement = ((C[2,0] * A[0,0])+(C[2,1] * A[1,0])+(C[2,2]*A[2,0]))
#print(CAtwobyzeroelement)
CAtwobyoneelement = ((C[2,0] * A[0,1])+(C[2,1] * A[1,1])+(C[2,2]*A[2,1]))
#print(CAtwobyoneelement)
CAtwobytwoelement = ((C[2,0] * A[0,2])+(C[2,1] * A[1,2])+(C[2,2]*A[2,2]))
#print(CAtwobytwoelement)

print(f'The product of C and A are:\n [{[CAzerobyzeroelement, CAzerobyoneelement, CAzerobytwoelement]},\n {[CAonebyzeroelement, CAonebyoneelement, CAonebytwoelement]},\n {[CAtwobyzeroelement, CAtwobyoneelement, CAtwobytwoelement]}]')
```

Figure 20: Property 4 B and A plus C and A Proof (Programmer's Built in Multiplication Method)

```
#For the sum
sumone = BAzerobyzeroelement + CAzerobyzeroelement
sumtwo = BAzerobyoneelement + CAzerobyoneelement
sumthree = BAzerobytwoelement + CAzerobytwoelement
sumfour = BAonebyzeroelement + CAonebyzeroelement
sumfive = BAonebyoneelement + CAonebyoneelement
sumsix = BAonebytwoelement + CAonebytwoelement
sumseven = BAtwobyzeroelement + CAtwobyzeroelement
sumeight = BAtwobyoneelement + CAtwobyoneelement
sumnine = BAtwobyoneelement + CAtwobytwoelement

print(f'Overall are:\n [{[sumone, sumtwo, sumthree]},\n {[sumfour, sumfive, sumsix]},\n {[sumseven, sumeight, sumnine]}]')
```

```
The product of B and A are:
 [[33, 42, 51],
 [51, 63, 75],
 [60, 72, 84]]
The product of C and A are:
 [[99, 123, 147],
 [60, 72, 84],
 [60, 72, 84]]
Overall are:
 [[132, 165, 198],
 [111, 135, 159],
 [120, 144, 156]]
```

Figure 21: Property 4 B and A plus C and A Proof (Programmer's Built in Multiplication Method)

Property 4. $(B + C) \cdot A = B \cdot A + C \cdot A$

```
[14]: (B+C) @ A

[14]: array([[132, 165, 198],
             [111, 135, 159],
             [120, 144, 168]])

[15]: B@A + C@A

[15]: array([[132, 165, 198],
             [111, 135, 159],
             [120, 144, 168]])

[16]: (B+C) @ A  == B@A + C@A

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

Figure 22: Property 4 Proof (Numpy Multiplication)

The programmer noticed that it is slightly similar in property, just the positioning made this property different. As the reader will observed, the programmer's built -in formula are way too long rather than to numpy, and that's the reason why it's more convenient to used numpy multiplication.

```
#For A at I
O = np.eye(3)
O

#A at O
#first row of array
AOzerobyzeroelement = ((A[0,0] * O[0,0])+(A[0,1] * O[1,0])+(A[0,2]*O[2,0]))
#print(AOzerobyzeroelement)
AOzerobyoneelement = ((A[0,0] * O[0,1])+(A[0,1] * O[1,1])+(A[0,2]*O[2,1]))
#print(AOzerobyoneelement)
AOzerobytwoelement = ((A[0,0] * O[0,2])+(A[0,1] * O[1,2])+(A[0,2]*O[2,2]))
#print(AOzerobytwoelement)

#second row of array
AOonebyzeroelement = ((A[1,0] * O[0,0])+(A[1,1] * O[1,0])+(A[1,2]*O[2,0]))
#print(AOonebyzeroelement)
AOonebyoneelement = ((A[1,0] * O[0,1])+(A[1,1] * O[1,1])+(A[1,2]*O[2,1]))
#print(AOonebyoneelement)
AOonebytwoelement = ((A[1,0] * O[0,2])+(A[1,1] * O[1,2])+(A[1,2]*O[2,2]))
#print(AOonebytwoelement)

#third row of array
AOtwobyzeroelement = ((A[2,0] * O[0,0])+(A[2,1] * O[1,0])+(A[2,2]*O[2,0]))
#print(AOtwobyzeroelement)
AOtwobyoneelement = ((A[2,0] * O[0,1])+(A[2,1] * O[1,1])+(A[2,2]*O[2,1]))
#print(AOtwobyoneelement)
AOtwobytwoelement = ((A[2,0] * O[0,2])+(A[2,1] * O[1,2])+(A[2,2]*O[2,2]))
#print(AOtwobytwoelement)

print(f'The product of A and I are:\n [{[AOzerobyzeroelement, AOzerobyoneelement, AOzerobytwoelement]},\n {[AOonebyzeroelement, AOonebyoneelement, AOonebytwoelement]},\n {[AOtwobyzeroelement, AOtwobyoneelement, AOtwobytwoelement]}]')
```

```
The product of A and I are:
 [[1.0, 2.0, 3.0],
 [4.0, 5.0, 6.0],
 [7.0, 8.0, 9.0]]
```

Figure 23: Property 5 A and I Proof (Programmer's Built in Multiplication Method)

Property 5. $\boldsymbol{A} \cdot \boldsymbol{I} = \boldsymbol{A}$

```
[17]: A

[17]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])

[18]: np.eye(3)

[18]: array([[1., 0., 0.],
             [0., 1., 0.],
             [0., 0., 1.]])

[19]: A @ np.eye(3)

[19]: array([[1., 2., 3.],
             [4., 5., 6.],
             [7., 8., 9.]])

[20]: A @ np.eye(3) == A

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

Figure 24: Property 5 A and I Proof and Validity (Numpy multiplication function)

In this property, it is much easier using the numpy multiplication still even though it is sort of short. It is still convenient to use numpy multiplication.

```
#For C at Null

#first row of array
Cnullzerobyzeroelement = ((C[0,0] * null_mat[0,0])+(C[0,1] * null_mat[1,0])+(C[0,2]*null_mat[2,0]))
#print(Cnullzerobyzeroelement)
Cnullzerobyoneelement = ((C[0,0] * null_mat[0,1])+(C[0,1] * null_mat[1,1])+(C[0,2]*null_mat[2,1]))
#print(Cnullzerobyoneelement)
Cnullzerobytwoelement = ((C[0,0] * null_mat[0,2])+(C[0,1] * null_mat[1,2])+(C[0,2]*null_mat[2,2]))
#print(Cnullzerobytwoelement)

#second row of array
Cnullonebyzeroelement = ((C[1,0] * null_mat[0,0])+(C[1,1] * null_mat[1,0])+(C[1,2]*null_mat[2,0]))
#print(Cnullonebyzeroelement)
Cnullonebyoneelement = ((C[1,0] * null_mat[0,1])+(C[1,1] * null_mat[1,1])+(C[1,2]*null_mat[2,1]))
#print(Cnullonebyoneelement)
Cnullonebytwoelement = ((C[1,0] * null_mat[0,2])+(C[1,1] * null_mat[1,2])+(C[1,2]*null_mat[2,2]))
#print(Cnullonebytwoelement)

#third row of array
Cnulltwobyzeroelement = ((C[2,0] * null_mat[0,0])+(C[2,1] * null_mat[1,0])+(C[2,2]*null_mat[2,0]))
#print(Cnulltwobyzeroelement)
Cnulltwobyoneelement = ((C[2,0] * null_mat[0,1])+(C[2,1] * null_mat[1,1])+(C[2,2]*null_mat[2,1]))
#print(Cnulltwobyoneelement)
Cnulltwobytwoelement = ((C[2,0] * null_mat[0,2])+(C[2,1] * null_mat[1,2])+(C[2,2]*null_mat[2,2]))
#print(Cnulltwobytwoelement)

print(f'The product of C and Null are:\n [{[Cnullzerobyzeroelement, Cnullzerobyoneelement, Cnullzerobytwoelement]},\n {[Cnullonebyzeroelement, Cnullonebyoneelement, Cnullonebytwoelement]},\n {[Cnulltwobyzeroelement, Cnulltwobyoneelement, Cnulltwoby
```

```
The product of C and Null are:
 [[0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0]]
```

Figure 25: Property 6 C and null Proof (Programmer's Built in Multiplication Method)

Property 6. $C \cdot \varnothing = \varnothing$

```
[21]: C
```

```
[21]: array([[7, 9, 8],
             [1, 6, 5],
             [3, 2, 7]])
```

```
[22]: null_mat = np.zeros((3,3))
      print(null_mat)
```

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

```
[23]: C @ null_mat
```

```
[23]: array([[0., 0., 0.],
             [0., 0., 0.],
             [0., 0., 0.]])
```

```
[24]: C @ null_mat == null_mat
```

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

Figure 26: Property 5 C and null Proof and Validity (Numpy multiplication function)

In this property, this null property is much easier still to use numpy multiplication rather than the programmer's built in formula and it can be shown the proof above this statement.

Link

https://github.com/chrisjoaquin29/Laboratory-Activity-7

IV. Conclusion

On the given activities, the programmer learned how to used operations, multiplication to be exact, by means of using codes. It is much easier than computing manually. There's a lot of codes that are needed to be encoded to achieved the expected result by the programmer yet with this functionality, it makes his life to do this codes easier than his built-n function. He achieved the same precise values every time he tried to compute different values in a specific array. Moreover, it is not time consuming even though, he will compute several data and the programmer came up to the conclusionh that it can be more easier accurate and efficient to use programming for computation compare to manually type every value in the calculator. Nonetheless, this matrix operations solve problems in healthcare so we can compute the cases here in the Philippines if it increases or decreases. With this matrix operation, it can help to compute a tons of datas of the healthcare so that our healthcare facility won't consume their time too much to count the numbers of patients, cases and many more.

References

[1]*numpy.eye — NumPy v1.19 Manual*. (n.d.). Retrieved December 18, 2020, from https://numpy.org/doc/stable/reference/generated/numpy.eye.html

[2]*numpy.zeros — NumPy v1.19 Manual*. (n.d.). Retrieved December 18, 2020, from https://numpy.org/doc/stable/reference/generated/numpy.zeros.html