Adamson University
College of Engineering
Computer Engineering Department

LINEAR ALGEBRA

Laboratory No. # 9

MATRIX ALGEBRA

Score

CRITERIA	Exceeds Expectations	Meets Expectations	Needs Improvement	Unsatisfactory
Functionality (60 points)				
Completeness (20 points)				
Structure (20 points)				

Remarks: ____

____

Submitted by:
Ugot, Aaron Paul M.
<Monday 7:00-10:00> / <58013>

Submitted to
<Ms. Maria Rizette Sayo>
<Facilitator>

Date Performed:
09-10-2023

Date Submitted
13-10-2023

Objective

1. Be familiar with the fundamental matrix operations.
2. Apply the operations to solve intermediate equations.
3. Apply matrix algebra in engineering solutions.

Algorithm

1. Type the main title of this activity as "Matrix Algebra”
2. On your GitHub, create a repository name Linear Algebra 58013
3. On your Colab, name your activity as Python Exercise 9.ipynb and save a copy to your GitHub repository

Discussion

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Determinant

A determinant is a scalar value derived from a square matrix. The determinant is a fundamental and important value used in matrix algebra. Although it will not be evident in this laboratory on how it can be used practically, but it will be reatly used in future lessons.

The determinant of some matrix A is denoted as det(A) or |A|. So let's say A is represented as:

$$A = \begin{bmatrix} a_{(0,\ 0)} & a_{(0,\ 1)} \\ a_{(1,\ 0)} & a_{(1,\ 1)} \end{bmatrix}$$

We can compute for the determinant as:

$$|A| = a_{(0,\ 0)} * a_{(1,\ 1)} - a_{(1,\ 0)} * a_{(0,\ 1)}$$

But you might wonder how about square matrices beyond the shape $(2,2)$? We can approach this problem by using several methods such as co-factor expansion and the minors method. This can be taught in the lecture of the laboratory but we can achieve the strenuous computation of high-dimensional matrices programmatically using Python. We can achieve this by using np.linalg.det().

```
A = np.array([
    [1,4],
    [0,3]
])
np.linalg.det(A)
```

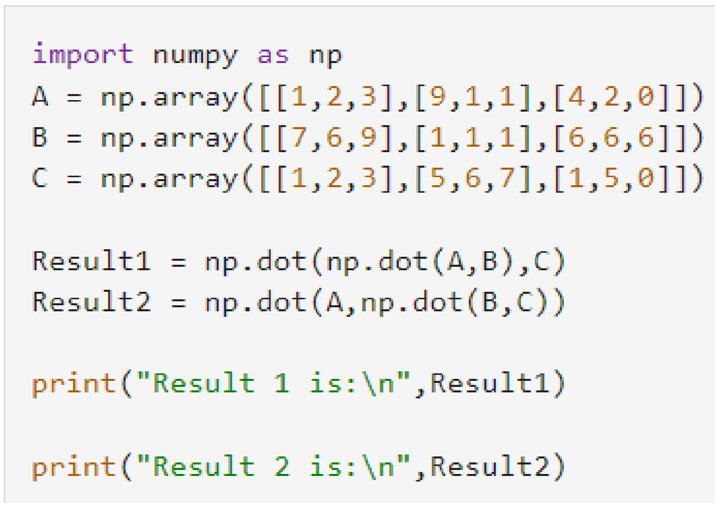
```
## Now other mathematics classes would require you to solve this by hand,
## and that is great for practicing your memorization and coordination skills
## but in this class we aim for simplicity and speed so we'll use programming
## but it's completely fine if you want to try to solve this one by hand.
B = np.array([
    [1,3,5,6],
    [0,3,1,3],
    [3,1,8,2],
    [5,2,6,8]
])
np.linalg.det(B)
```

Adamson University
College of Engineering
Computer Engineering Department

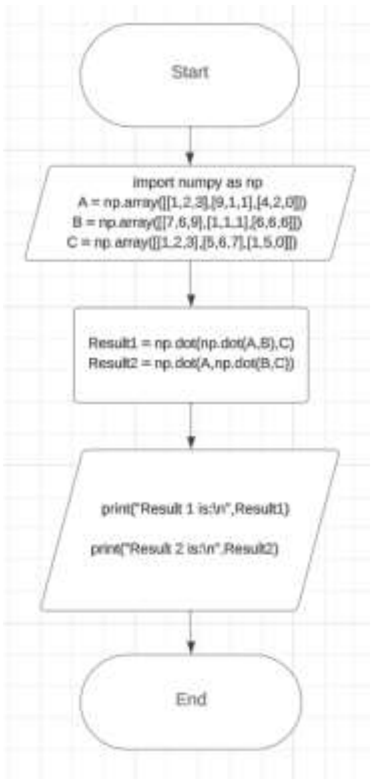
<u>Coding Activity 9</u>

Prove and implement the remaining 6 matrix multiplication properties. You may create your own matrices in which their shapes should not be lower than (3,3). In your methodology, create individual flowcharts for each property and discuss the property you would then present your proofs or validity of your implementation in the results section by comparing your result to present functions from NumPy.

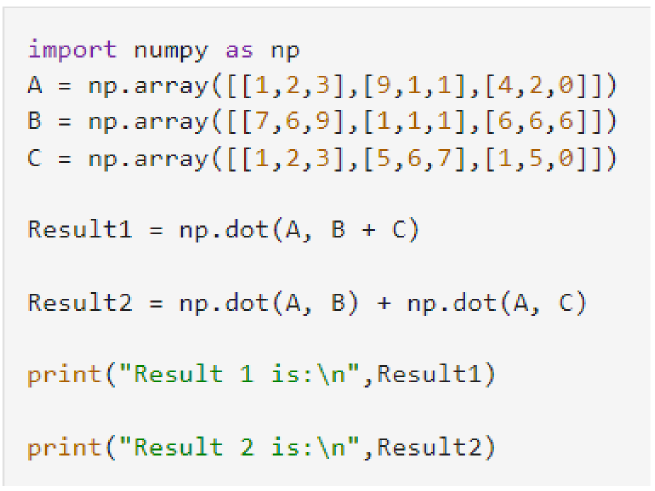
Associative



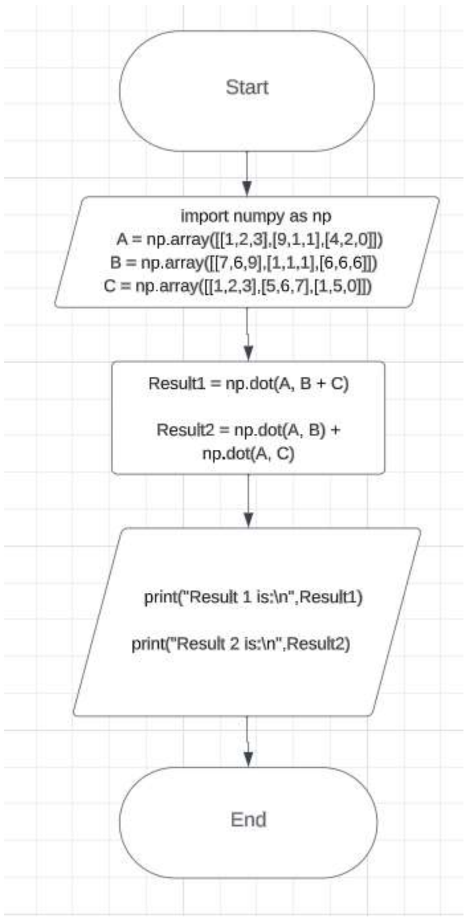
```
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])

Result1 = np.dot(np.dot(A,B),C)
Result2 = np.dot(A,np.dot(B,C))

print("Result 1 is:\n",Result1)

print("Result 2 is:\n",Result2)
```


Start
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])
Result1 = np.dot(np.dot(A,B),C)
Result2 = np.dot(A,np.dot(B,C))
print("Result 1 is:\n",Result1)
print("Result 2 is:\n",Result2)
End

Distributive

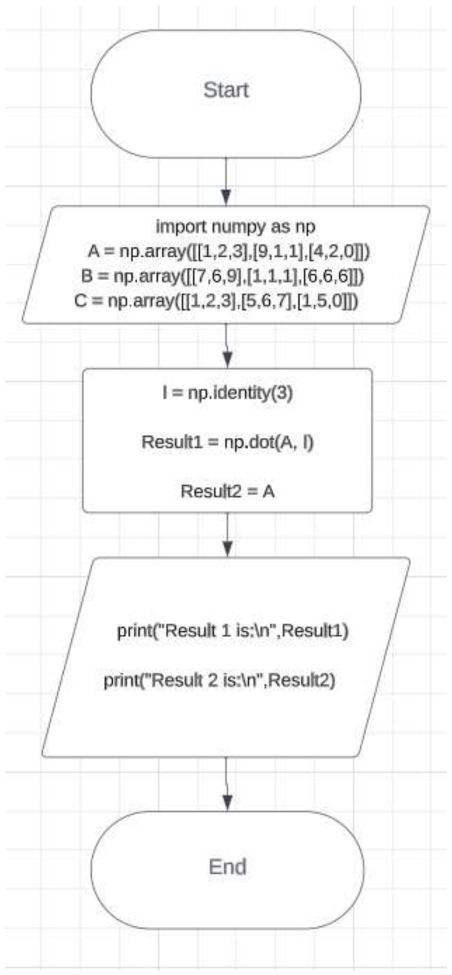


```
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])

Result1 = np.dot(A, B + C)

Result2 = np.dot(A, B) + np.dot(A, C)

print("Result 1 is:\n",Result1)

print("Result 2 is:\n",Result2)
```


Start
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])
Result1 = np.dot(A, B + C)
Result2 = np.dot(A, B) + np.dot(A, C)
print("Result 1 is:\n",Result1)
print("Result 2 is:\n",Result2)
End

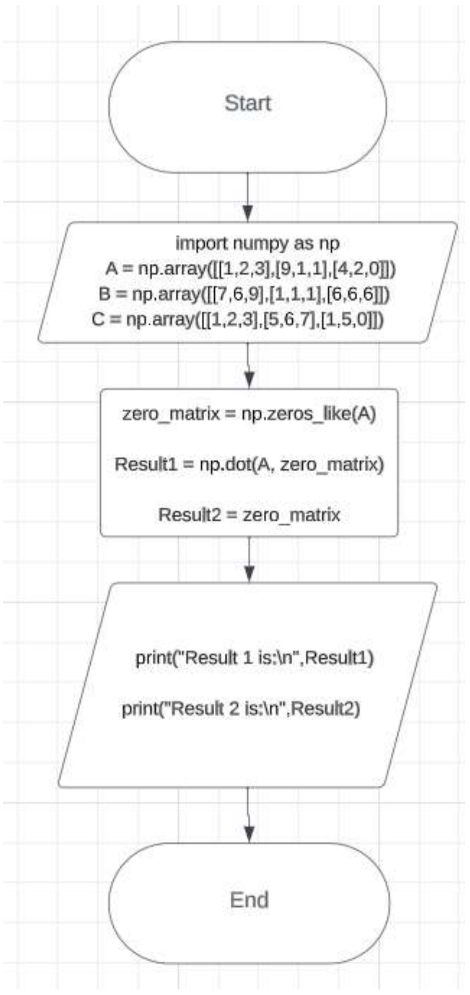
Identity

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

I = np.identity(3)

Result1 = np.dot(A, I)

Result2 = A

print("Result 1 is:\n",Result1)

print("Result 2 is:\n",Result2)
```

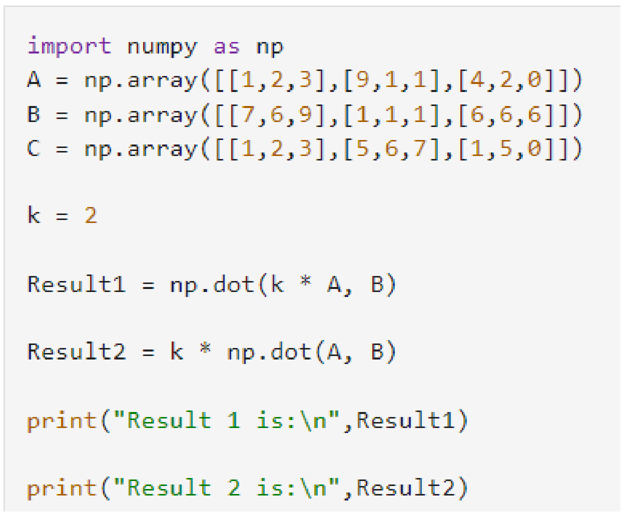

Start
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])
I = np.identity(3)
Result1 = np.dot(A, I)
Result2 = A
print("Result 1 is:\n",Result1)
print("Result 2 is:\n",Result2)
End

Zero

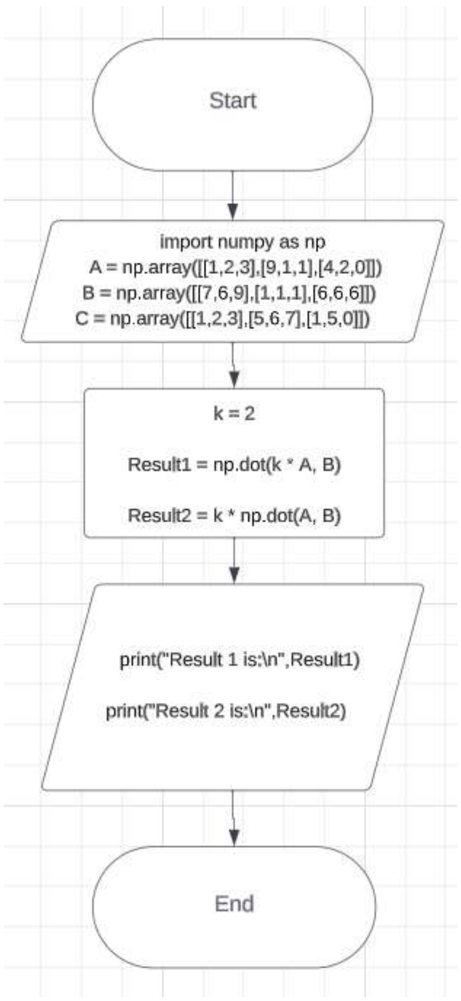
```
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])

zero_matrix = np.zeros_like(A)

Result1 = np.dot(A, zero_matrix)

Result2 = zero_matrix

print("Result 1 is:\n",Result1)

print("Result 2 is:\n",Result2)
```


Start
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])
zero_matrix = np.zeros_like(A)
Result1 = np.dot(A, zero_matrix)
Result2 = zero_matrix
print("Result 1 is:\n",Result1)
print("Result 2 is:\n",Result2)
End

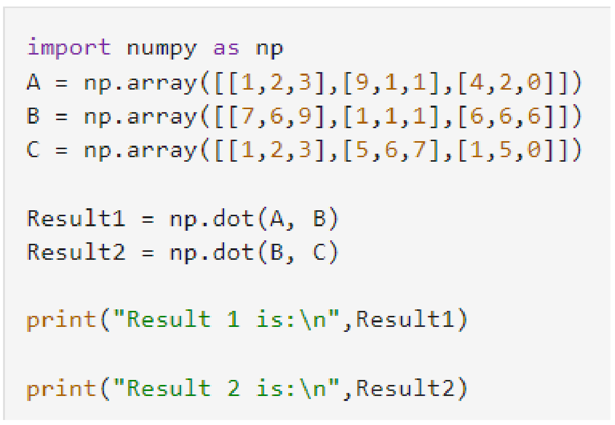
Scalar



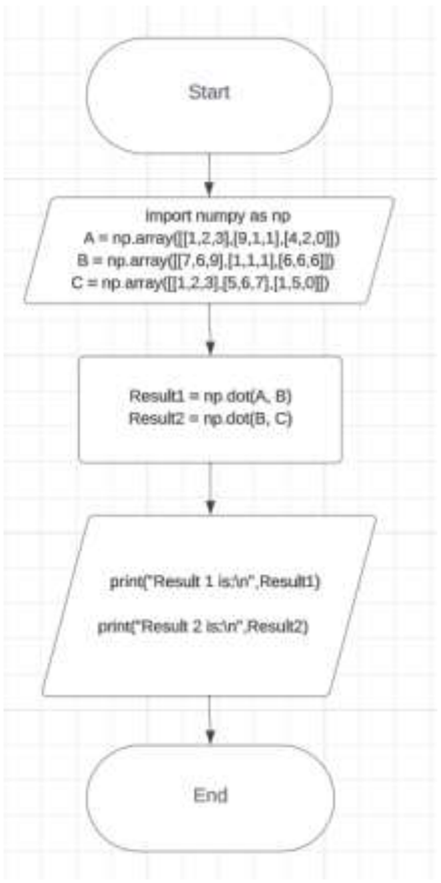
```
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])

k = 2

Result1 = np.dot(k * A, B)

Result2 = k * np.dot(A, B)

print("Result 1 is:\n",Result1)

print("Result 2 is:\n",Result2)
```


Start
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])
k = 2
Result1 = np.dot(k * A, B)
Result2 = k * np.dot(A, B)
print("Result 1 is:\n",Result1)
print("Result 2 is:\n",Result2)
End

Closure



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

Result1 = np.dot(A, B)
Result2 = np.dot(B, C)

print("Result 1 is:\n",Result1)

print("Result 2 is:\n",Result2)
```


Start
import numpy as np
A = np.array([[1,2,3],[9,1,1],[4,2,0]])
B = np.array([[7,6,9],[1,1,1],[6,6,6]])
C = np.array([[1,2,3],[5,6,7],[1,5,0]])
Result1 = np.dot(A, B)
Result2 = np.dot(B, C)
print("Result 1 is:\n",Result1)
print("Result 2 is:\n",Result2)
End

Github link:
https://github.com/Ugot-Aaron-Paul/58013-LinearAlgebra