Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 4

Vector Operations

Submitted by:
Vallarta, Troy Joaquin G.

Instructor:
Engr. Dylan Josh D. Lopez

November 04, 2020

I. Objectives

This laboratory activity aims to implement the principles and techniques of vector operations and its uses.

II. Methods

- The practices for this activity is to understand how vector operations work
  - The laboratory activity teaches the researchers to understand vector operations and manually compute for some NumPy functions.
- The laboratory delivered results using vector operations.
  - The researcher achieved the result by learning the process of vector operations.

III. Results

In this laboratory activity, the researchers' task was to compute a vector manually for a vector modulus and compare it to the Numpy function. It aims to understand how the modulus formula works in codes. Task one of the laboratory is to compare the numpy.linalg.norm() function and the researcher has created a formula if they are the same. As seen in Figure 1, both Numpy functions and the researcher's created algorithm are the same. It means that the created algorithm is how numpy.linalg.norm() function operates. The researcher used the round() function in printing the modulus of a certain vector to help the readers understand the output more easily, as seen in figure one. The researchers created the vectors to have an element of higher than four to comply with the minimum elements stated in the task's question. The researcher inputted six different vectors to ensure that the formula used by the researcher is consistent.

```python
def inner_product(vect1, vect2):
    sumall = 0
    if len(vect1) == len(vect2):
        for j in range(len(vect1)):
            sumall += vect1[j] * vect2[j]
        return sumall



vect_1 = [8, 56, 0, 7, 3]
vect_2 = [6, 7, 8, 9, 10]
vect_3 = [23, 13, 2, 89, 99, 1]
vect_4 = [8, 1, 78, 21, 9, 55]
vect_5 = [1, 2, 3, 4, 5, 6, 7]
vect_6 = [8, 9, 10, 11, 12, 13, 14]
vect_7 = [99, 98, 97, 96, 95, 94]
vect_8 = [45, 50, 78, 12, 22,66]
vect_9 = [1, 0, 8, 9, 7]
vect_10 = [22, 15, 12, 9, 5]

print("Using inner product method: \n")
print("VECTOR 1 & VECTOR 2: ", inner_product(vect_1,vect_2))
print("VECTOR 3 & VECTOR 4: ", inner_product(vect_3,vect_4))
print("VECTOR 5 & VECTOR 6: ", inner_product(vect_5,vect_6))
print("VECTOR 7 & VECTOR 8: ", inner_product(vect_7,vect_8))
print("VECTOR 9 & VECTOR 10: ", inner_product(vect_9,vect_10))

print("\nUsing np.inner() function: \n")
print("VECTOR 1 & VECTOR 2: ", np.inner(vect_1,vect_2))
print("VECTOR 3 & VECTOR 4: ", np.inner(vect_3,vect_4))
print("VECTOR 5 & VECTOR 6: ", np.inner(vect_5,vect_6))
print("VECTOR 7 & VECTOR 8: ", np.inner(vect_7,vect_8))
print("VECTOR 9 & VECTOR 10: ", np.inner(vect_9,vect_10))
```

Figure 1. Input Laboratory Task 1

The output of using the researcher's modulus created function and Using np.linalg.norm() function are both the same, as seen in figure 2.

```
Using modulus method:

VECTOR 1:  13.38
VECTOR 2:  92.97
VECTOR 3:  116.94
VECTOR 4:  133.72
VECTOR 5:  82.71
VECTOR 6:  21.79

Using np.linalg.norm() function:

VECTOR 1:  13.38
VECTOR 2:  92.97
VECTOR 3:  116.94
VECTOR 4:  133.72
VECTOR 5:  82.71
VECTOR 6:  21.79
```

Figure 2. Output Laboratory Task 1

As seen in figure 3, the first task's flowchart is visualizing the researcher's created function. The start of the function has a parameter that needs a vector. This flowchart shows a for loop visualization to execute the formula of a modulus of a vector. After the condition is satisfied, the program will return the answer.

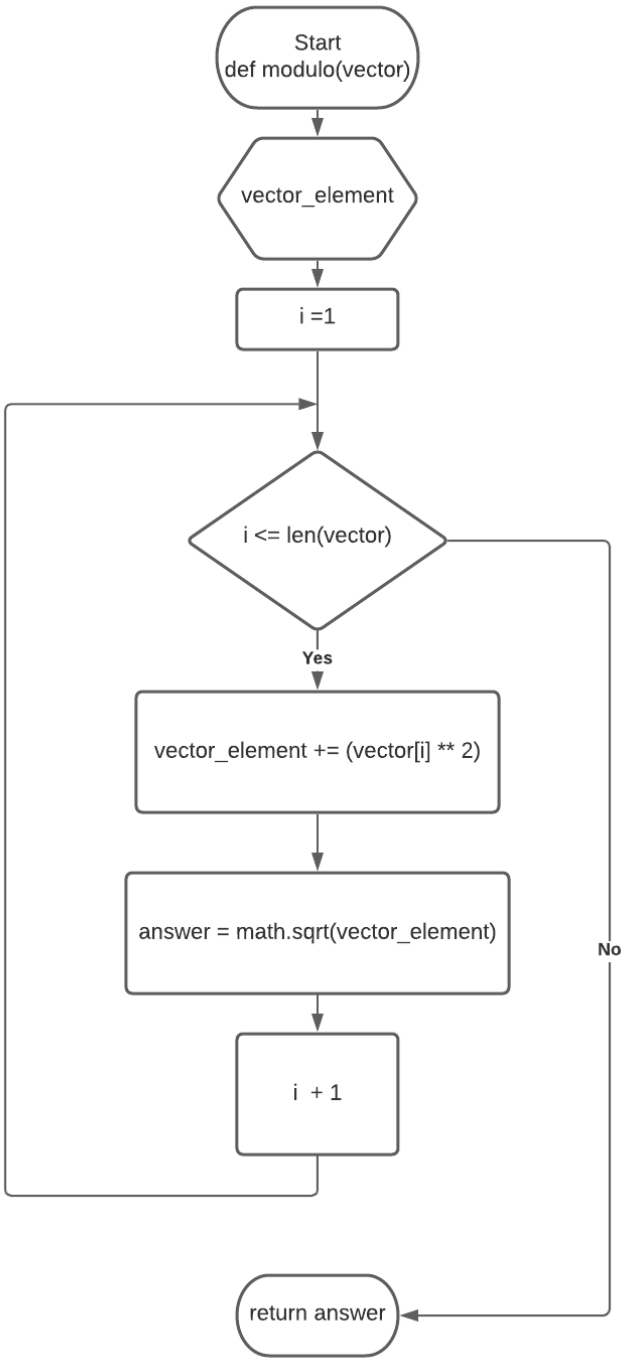

Figure 3. Task 1 Flowchart

Task two of the laboratory activity aims to teach the researchers how to compute for a two vector's inner product, as seen in figure 4. In this inner product of two vectors multiplied the same index element and vector and got the sum of it all. In this laboratory activity, two

vectors need to have the same element count. The program will have an error. To compare if the researcher created the correct algorithm, Numpy has a numpy.inner() function, this function's process is the same as the researcher's created algorithm. It means that the created algorithm of the researcher is correct. Task two of this laboratory activity aims to understand how vector product works.

```
def inner_product(vect1, vect2):
    sumall = 0
    if len(vect1) == len(vect2):
        for j in range(len(vect1)):
            sumall += vect1[j] * vect2[j]
        return sumall



vect_1 = [8, 56, 0, 7, 3]
vect_2 = [6, 7, 8, 9, 10]
vect_3 = [23, 13, 2, 89, 99, 1]
vect_4 = [8, 1, 78, 21, 9, 55]
vect_5 = [1, 2, 3, 4, 5, 6, 7]
vect_6 = [8, 9, 10, 11, 12, 13, 14]
vect_7 = [99, 98, 97, 96, 95, 94]
vect_8 = [45, 50, 78, 12, 22,66]
vect_9 = [1, 0, 8, 9, 7]
vect_10 = [22, 15, 12, 9, 5]

print("Using inner product method: \n")
print("VECTOR 1 & VECTOR 2: ", inner_product(vect_1,vect_2))
print("VECTOR 3 & VECTOR 4: ", inner_product(vect_3,vect_4))
print("VECTOR 5 & VECTOR 6: ", inner_product(vect_5,vect_6))
print("VECTOR 7 & VECTOR 8: ", inner_product(vect_7,vect_8))
print("VECTOR 9 & VECTOR 10: ", inner_product(vect_9,vect_10))

print("\nUsing np.inner() function: \n")
print("VECTOR 1 & VECTOR 2: ", np.inner(vect_1,vect_2))
print("VECTOR 3 & VECTOR 4: ", np.inner(vect_3,vect_4))
print("VECTOR 5 & VECTOR 6: ", np.inner(vect_5,vect_6))
print("VECTOR 7 & VECTOR 8: ", np.inner(vect_7,vect_8))
print("VECTOR 9 & VECTOR 10: ", np.inner(vect_9,vect_10))
```

Figure 4. Task 2 Input

The output shows that the researcher's created algorithm and the Numpy function are the same, as seen in figure five.

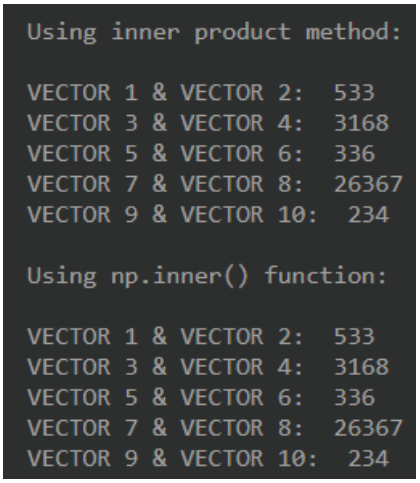

Figure 5. Task 2 Output

As seen in figure 6, the flowchart of task two visualizes the researcher's algorithm. In the function created, it needs two-parameter to create a vector product, the first condition that the flowchart needs is if the two vectors have the same elements inside. If they are not the same, the program will not execute to prevent error when the researcher input uneven elements. After the first condition, figure 6 shows that a condition looping represents the for a loop until it is satisfied. The last process that the flowchart shows is to return the vector product's value in two vectors.

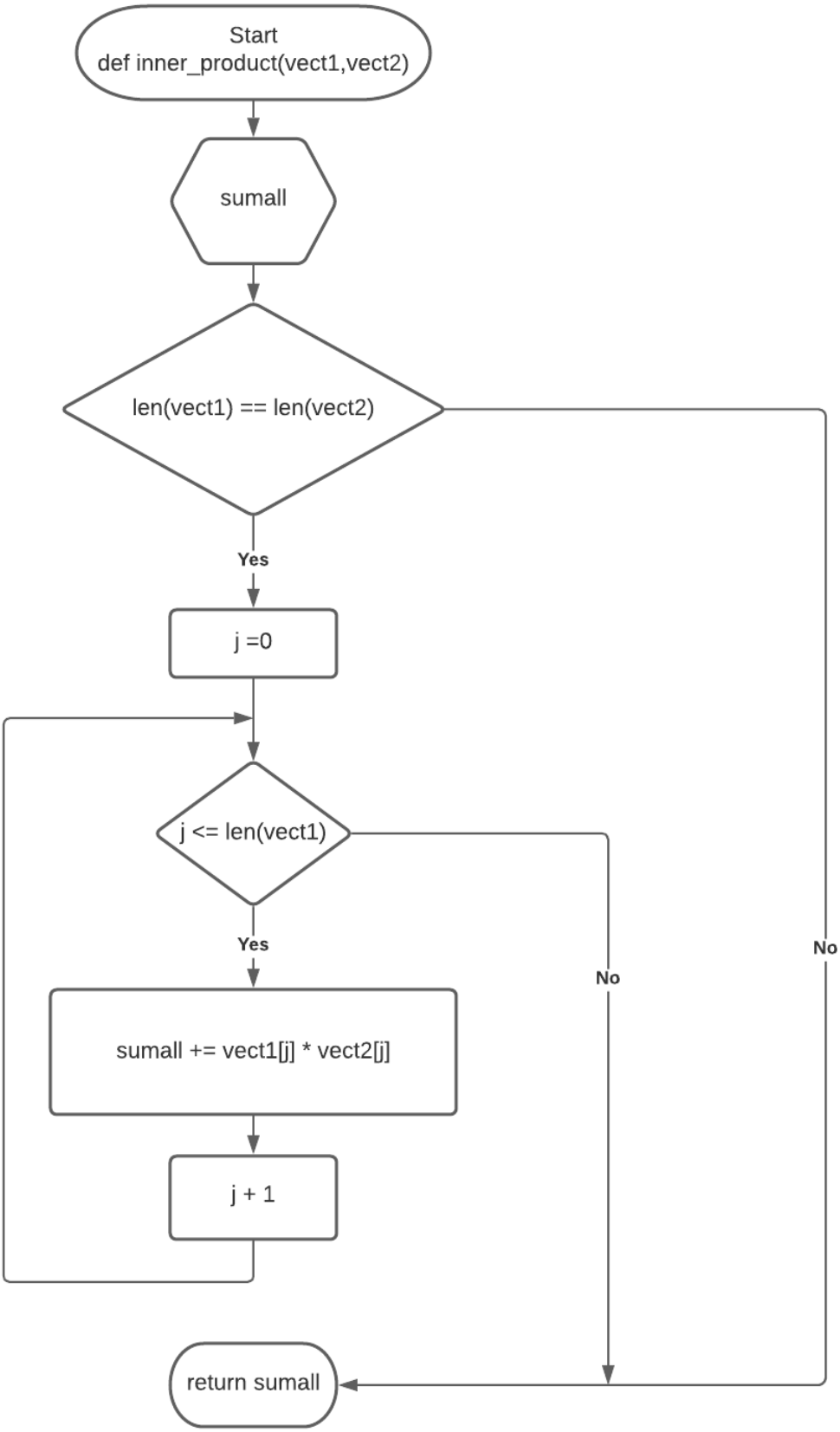

Figure 6. Task 2 Flowchart

The last part of the laboratory activity is a little challenging for the researchers. As seen in figure seven, the researchers are task to make the process and values in code form. The researcher is also tasked to visualize the code in a third-dimensional graph. Also, in this task, the researcher is expected to have the same output given by the professor.

$$((A^2 + B^2 + C^2) * (A * (B + A * B)./C)) * ||A + B + C||$$
$$A = \begin{bmatrix} -0.4 \\ 0.3 \\ -0.6 \end{bmatrix}, B = \begin{bmatrix} -0.2 \\ 0.2 \\ 1 \end{bmatrix}, C = \begin{bmatrix} 0.2 \\ 0.1 \\ -0.5 \end{bmatrix}$$

Figure 7. The question in Task 3

The input value of the matrix is given value in figure seven. In figure eight shows how the researcher made a graph in 3-dimensional. The researcher initialized the 3-dimension graph using plt.axes(projection = '3d') function, it creates a 3D black graph [1]. The np.arrange() function starts in the first parameter, and the second parameter is where the line will stop. Graph limits x,y, and z prevent the graph from going beyond the researcher's needed visualization information. The researcher's process was to copy the given formulas and change the given question into a code format. Using the np.linalg.norm() function to find the modulus like in the given task one. Lastly, print the line in the graph, the researcher multiplied the function np.arrange for the points to be in a specific length only. Lastly is to print the value of the given formula.

```
A = np.array([-0.4,0.3,-0.6])
B = np.array([-0.2,0.2,1])
C = np.array([0.2, 0.1,-0.5])

#initializing graph
graph = plt.axes(projection='3d')
c = np.arange(0,1.5)

graph.set_xlim([0, 1.2])
graph.set_ylim([0, 1.2])
graph.set_zlim([0, 1.2])


first = (A @ A) + (B @ B) + (C @ C)
second = A *(B+(A*B))/C
third = np.linalg.norm(A+B+C)
vector_ans = first * second * third


graph.plot(c*vector_ans[0], c*vector_ans[1], c*vector_ans[2], color='r')
plt.show()
print("Answer: ",vector_ans)
```

Figure 8. Task 3 Input

Figure 9 shows the graph output of the program created by the given formula and its value. The graph is a three-dimensional graph with x, y, and z value, as the readers can see.

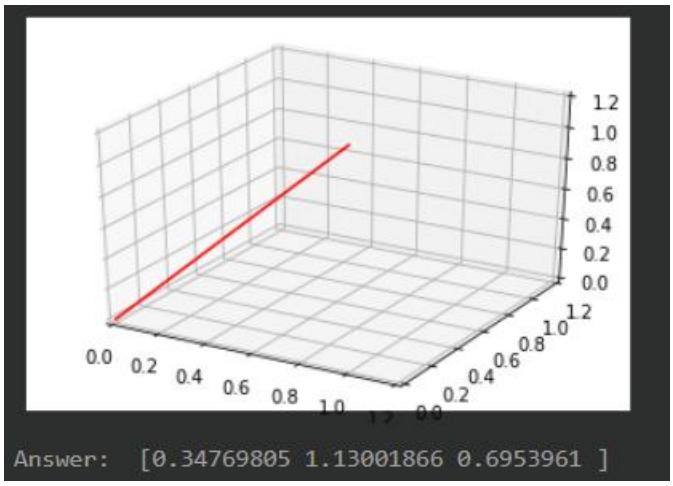

Figure 9. Task 3 Output

To know the process of a certain program, the researchers need to make a flowchart to understand the code in a logically understandable format. In this flowchart, the researcher initialized much value to create the graph's skeleton and the given matrix. After the initializing, the researcher made a process to compute for the given formula format. After initializing and computing the values, the next flowchart shows the graph and the computed graph's values.

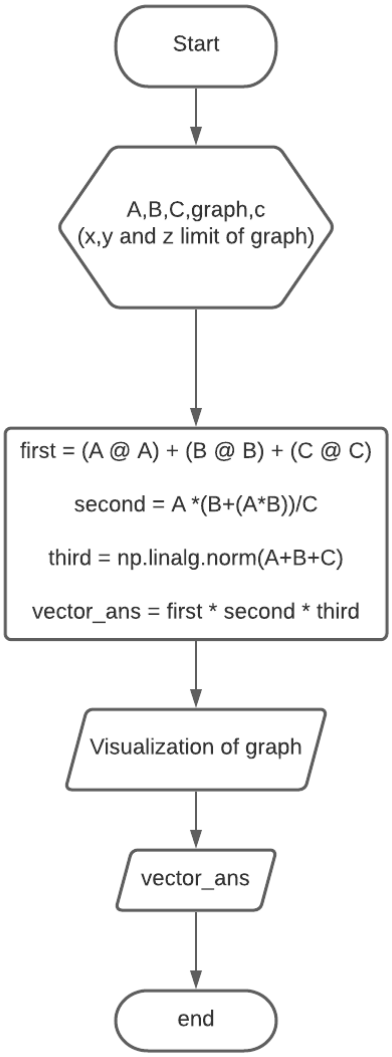

Figure 10. Task 10 Flowchart

IV. Conclusion

To end this laboratory activity, I learned that Numpy functions could be manually made into codes; some codes can be manually made, especially tasks 1 and 2. In this laboratory activity, I learned all about vector operations and its uses. How to compute manually of the modulus and inner product of a vector/vectors, I realized that the Numpy functions are formulas in math that are automated to compute within one access of its function. The use of it in a real-life situation is vast. I think that the world should automate what computers can do, and humans need to push our technological advancement. In this activity, I learned that sometimes we should rely on other works to make our lives/jobs easier.

References

[1] “How to plot 3D graphs in Python using Matplotlib,” CodeSpeedy, 26-Feb-2020. [Online]. Available: https://www.codespeedy.com/plotting-3d-graphs-in-python-using-matplotlib/. [Accessed: 04-Nov-2020].