Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 4

Vector Operations

Submitted by:
Reyes, Carl vincent G.

Instructor:
Engr. Dylan Josh D. Lopez

November 6, 2020

I. Objectives

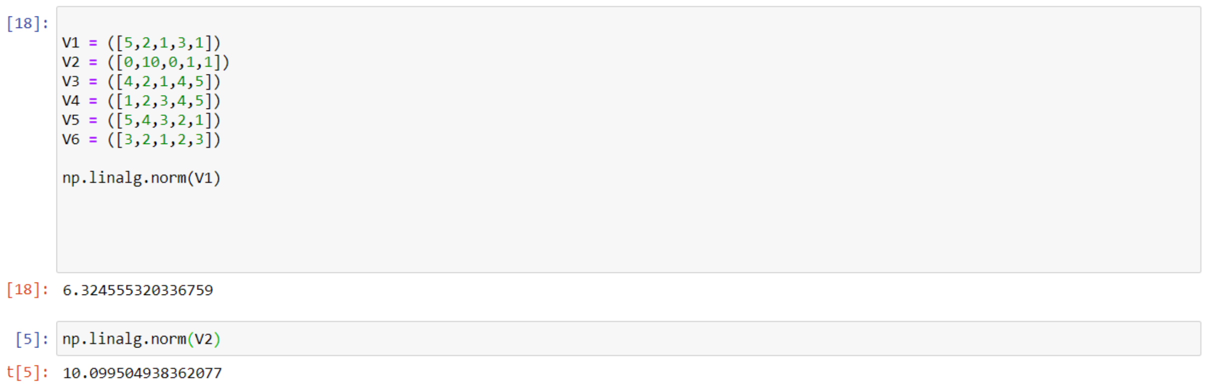
This laboratory activity aims to give the students fundamental knowledge about vector operations and perform coding using Python.

II. Methods

This activity focuses of how to perform different vector operations in Python, such as addition, subtraction, division, and advance operations, like dot product and modulus of a vector. Achieving this laboratory consists of focus and understanding about the topic that the professor taught and fundamental knowledge about the previous lessons like linear combination and 3-Dimensional plots, since the last task asks to visually represent the answers the programmers came up with 3-Dimensional plots.

III. Results

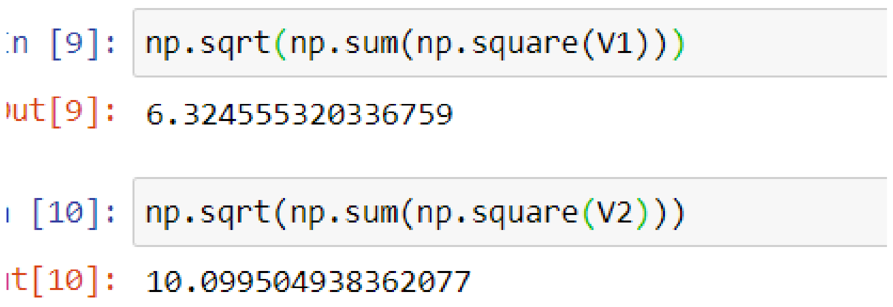
The first task was not that difficult, but having so much vectors and elements in just one task was very confusing for the programmer, and the result of it is more time being wasted thinking about what he or she should do as such:



```
[18]:
V1 = ([5,2,1,3,1])
V2 = ([0,10,0,1,1])
V3 = ([4,2,1,4,5])
V4 = ([1,2,3,4,5])
V5 = ([5,4,3,2,1])
V6 = ([3,2,1,2,3])

np.linalg.norm(V1)

[18]: 6.324555320336759

 [5]: np.linalg.norm(V2)
t[5]: 10.099504938362077
```

Figure 1: Result and codes of Task 1

Similar to the given example from the laboratory report, the programmer based his codes in those examples whereas, each arrays was tested and printed using np.linalg.norm(), which is by definition takes 1/8 of the matrix of the vector or calculates one of the vector norms, [1] which can be seen above.



```
In [9]: np.sqrt(np.sum(np.square(V1)))
Out[9]: 6.324555320336759

In [10]: np.sqrt(np.sum(np.square(V2)))
Out[10]: 10.099504938362077
```

Figure 2: Task 1, Programmers code

On the first task, the programmers are tasked to create their own way of computing the given and comparing it to using the np.linalg.norm. Based on the observation of the programmer, both codes showed or printed the same result, but the way the are coded is different, having the np.linalg.norm() is similar to simplifying the code that the programmer used to code the first task. Since the programmers code consists of having a squaring the elements of the array then adding, and finally the square root, which is a bit complicated, rather than using the np.linalg.norm().

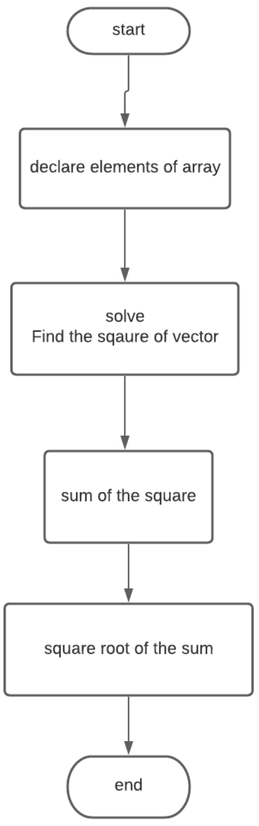
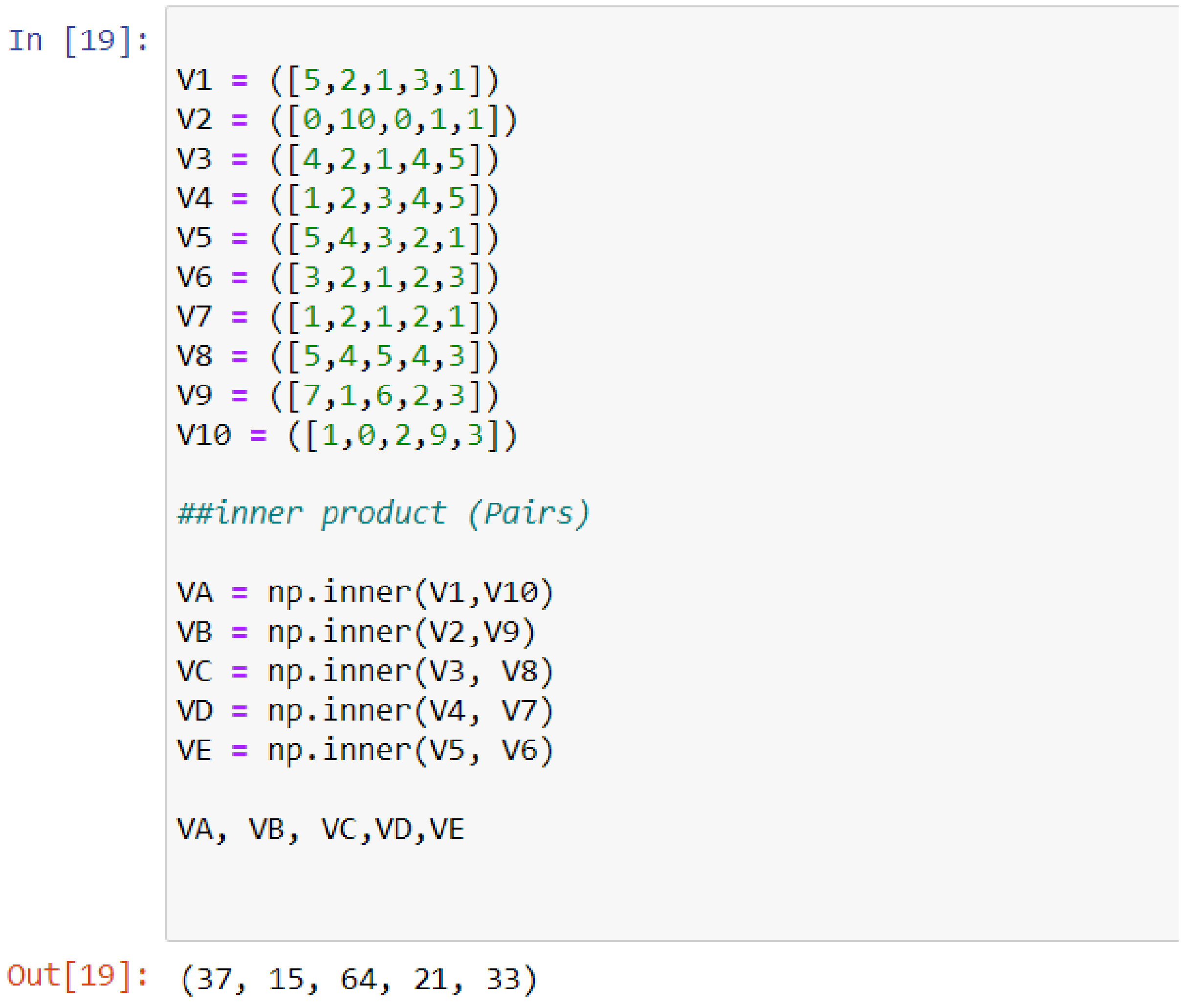


Figure 3: Flowchart of task 1 (programmers algo)

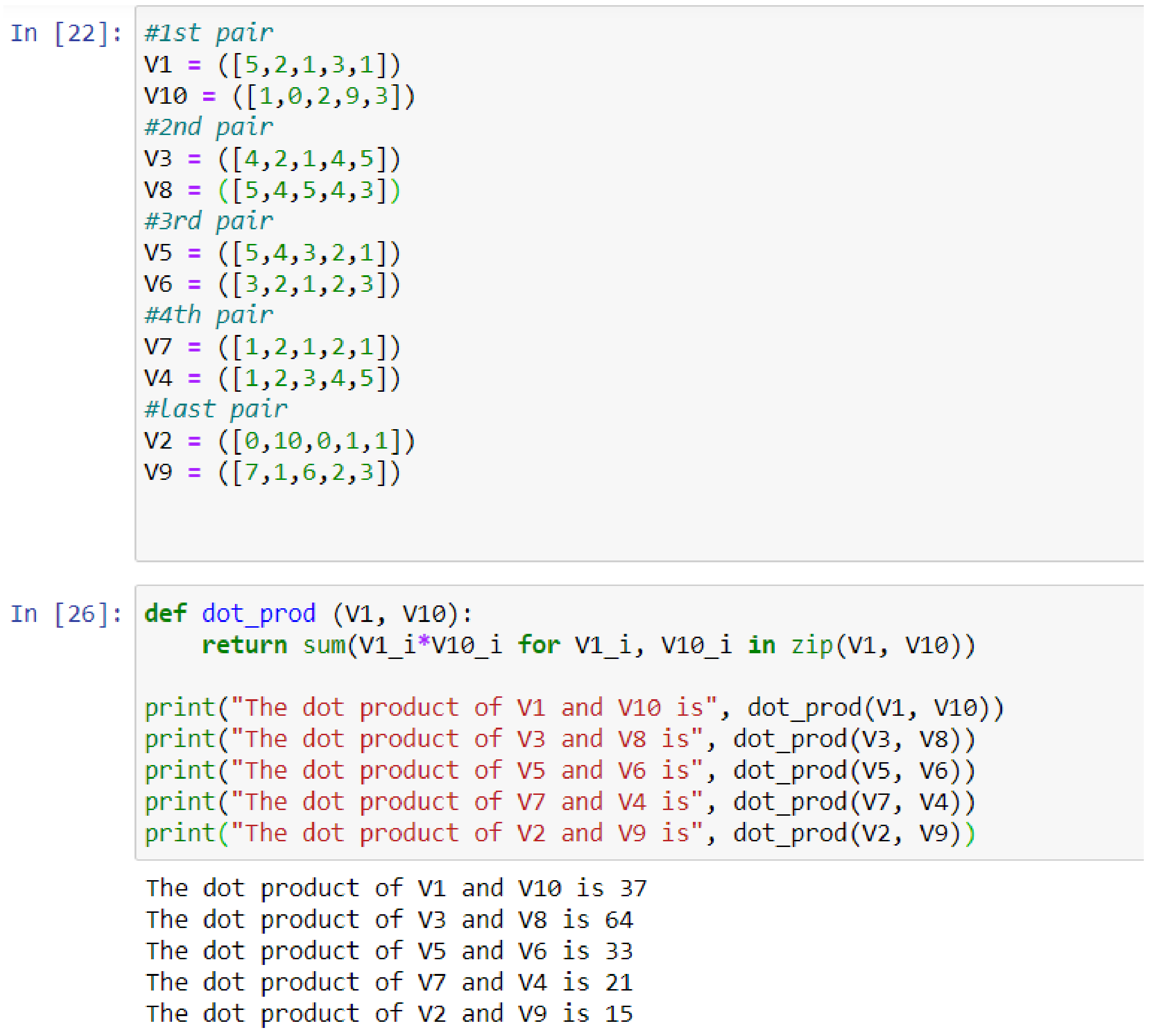
As seen from the figure above, it shows the process of the code the programmers used, from what first happened and how it ended and found the result.



```
In [19]:
V1 = ([5,2,1,3,1])
V2 = ([0,10,0,1,1])
V3 = ([4,2,1,4,5])
V4 = ([1,2,3,4,5])
V5 = ([5,4,3,2,1])
V6 = ([3,2,1,2,3])
V7 = ([1,2,1,2,1])
V8 = ([5,4,5,4,3])
V9 = ([7,1,6,2,3])
V10 = ([1,0,2,9,3])

##inner product (Pairs)

VA = np.inner(V1,V10)
VB = np.inner(V2,V9)
VC = np.inner(V3, V8)
VD = np.inner(V4, V7)
VE = np.inner(V5, V6)

VA, VB, VC,VD,VE

Out[19]: (37, 15, 64, 21, 33)
```

Figure 4: Codes and results of Task 2

Task 2 focused on inner products and dot products, the programmers are tasked to solve the Euclidian norm given in the situation whereas, they also need to solve without using the np.inner or @ operand. As for the figure 4, similarly to the examples given in the laboratory activity, the programmer based this code on it and managed to successfully run it with 10 vectors.



```
In [22]: #1st pair
         V1 = ([5,2,1,3,1])
         V10 = ([1,0,2,9,3])
         #2nd pair
         V3 = ([4,2,1,4,5])
         V8 = ([5,4,5,4,3])
         #3rd pair
         V5 = ([5,4,3,2,1])
         V6 = ([3,2,1,2,3])
         #4th pair
         V7 = ([1,2,1,2,1])
         V4 = ([1,2,3,4,5])
         #last pair
         V2 = ([0,10,0,1,1])
         V9 = ([7,1,6,2,3])
```

```
In [26]: def dot_prod (V1, V10):
             return sum(V1_i*V10_i for V1_i, V10_i in zip(V1, V10))

         print("The dot product of V1 and V10 is", dot_prod(V1, V10))
         print("The dot product of V3 and V8 is", dot_prod(V3, V8))
         print("The dot product of V5 and V6 is", dot_prod(V5, V6))
         print("The dot product of V7 and V4 is", dot_prod(V7, V4))
         print("The dot product of V2 and V9 is", dot_prod(V2, V9))
```

```
The dot product of V1 and V10 is 37
The dot product of V3 and V8 is 64
The dot product of V5 and V6 is 33
The dot product of V7 and V4 is 21
The dot product of V2 and V9 is 15
```

Figure 5: Codes used by the programmer

As for figure 5, these are the codes used by the programmer without using the np.inner() and the @operand. The code focuses on function which is named as dot_prod, with the first pair of vectors, and as for the code, paraphrased and was inspired in the website where they used this kind of code but only with 2 vectors, so the programmer adjusted it for the program to work. [2]

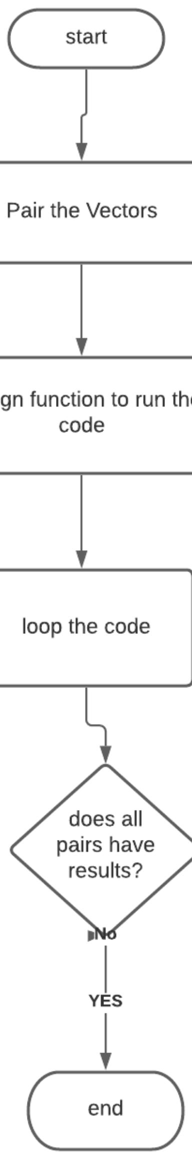
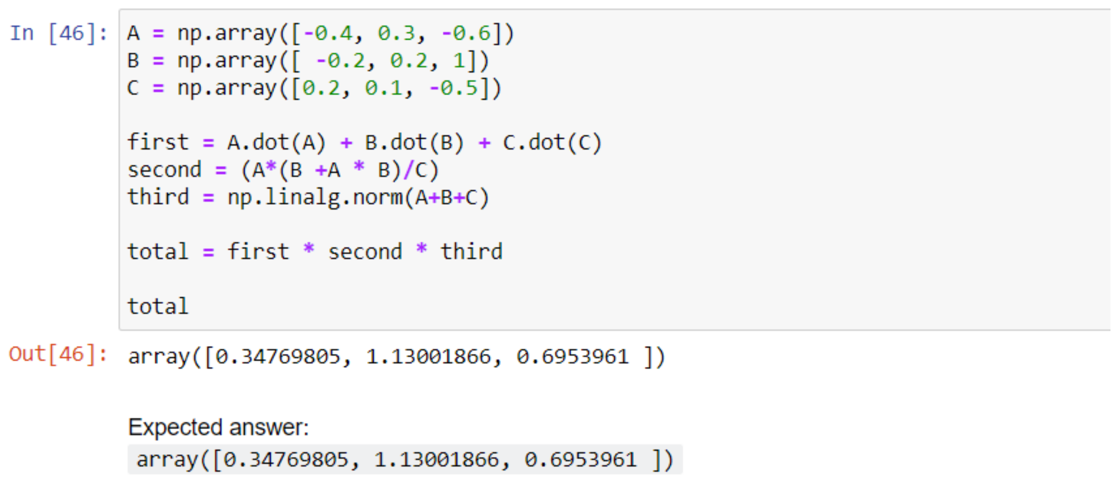


Figure 6: Flowchart for Task 2

As seen above, the flowchart shows the process of coding the program, which first pairs the vectors and then assigning a function for the vectors and formulating the code, then asks the question of all of the vectors are done, if not, loop the code and go to the next one, then if yes, it is done.



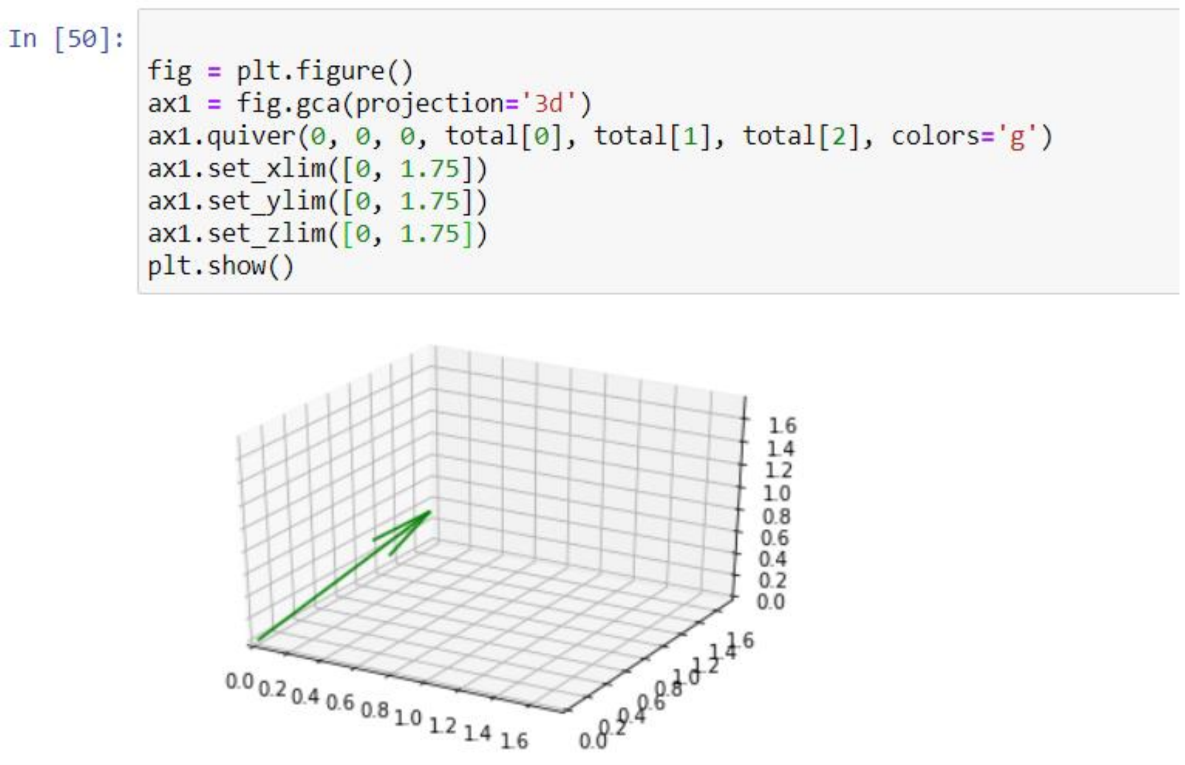
```
In [46]: A = np.array([-0.4, 0.3, -0.6])
         B = np.array([ -0.2, 0.2, 1])
         C = np.array([0.2, 0.1, -0.5])

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

         total = first * second * third

         total
Out[46]: array([0.34769805, 1.13001866, 0.6953961 ])
```

Expected answer:

```
array([0.34769805, 1.13001866, 0.6953961 ])
```

Figure 7: Codes and results for Task 3

This task is tricky at first but upon remembering past lessons and other fundamental knowledge that has been taught to the students, it is starting to feel familiar. The students are able to fo this already and with that knowledge the programmer was able to finish the code and apply the new lesson on this task. Succesfully the answers are correct because it is similar to the expected answer given by the professor.



```
In [50]: 
         fig = plt.figure()
         ax1 = fig.gca(projection='3d')
         ax1.quiver(0, 0, 0, total[0], total[1], total[2], colors='g')
         ax1.set_xlim([0, 1.75])
         ax1.set_ylim([0, 1.75])
         ax1.set_zlim([0, 1.75])
         plt.show()
```

Figure 8: plot for task 3

As for the last task of the lasb activity 4, which is plotting the answers in a 3-Dimensional plane, just like the previous lessons and laboratory, the programmer made use of that knowledge in order to successfully graph the answers.

IV. Conclusion

Overall, additional knowledge and learning have been added to the programmers hard drive, which will enable him to use in future lessons and maybe work. Vector operations and solving them using the different ways of coding is probably the lesson most learned by the programmer. Applying this laboratory can be seen in real life, given the right situations, such as having a company or a business, would help any programmer, graph and predict their loss and income with this, so learning and understanding this laboratory activity will be a great advantage for the programmer.

V. References

[1] "What is the np.linalg.norm() method in NumPy?," edpresso, [Online]. Available: https://www.educative.io/edpresso/what-is-the-nplinalgnorm-method-in-numpy. [Accessed 06 11 2020].

[2] M. BARTOLO, "DOT (SCALAR) PRODUCT," 06 03 2019. [Online]. Available: https://www.maxbartolo.com/ml-index-item/dot-scalar-product/#:~:text=In%20Python%2C%20one%20way%20to,comprehension%20performing%20element%2Dwise%20multiplication.&text=Alternatively%2C%20we%20can%20use%20the,dot()%20function.&text=Keeping%20to%20the%20con. [Accessed 06 11 2020].

Github Repo:

https://github.com/ReyesCarl/LinAlg_LAB_ACT4