Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 4

# Vector Operations

*Submitted by:*
Guy, Lawrence Adrian B.

*Instructor:*
Engr. Dylan Josh D. Lopez

October 26, 2020

# I. Objectives

This laboratory activity aims to refresh the knowledge gained from past classes regarding vector operations and to introduce new operations. The activity also aims to visualize and perform vector operations using NumPy and Matplotlib.

# II. Methods

The activity strives to familiarize students with different vector operations namely: vector addition; vector subtraction; vector multiplication; vector division; modulus of a vector; and dot product or inner product. This is done by using Python modules NumPy and Matplotlib to help students perform and visualize vector operations faster and easier.

The first deliverable of the activity is to create an explicit function to compute for the modulus of a vector using the Euclidian Norm formula and comparing it to the built-in function np.linalg.norm().

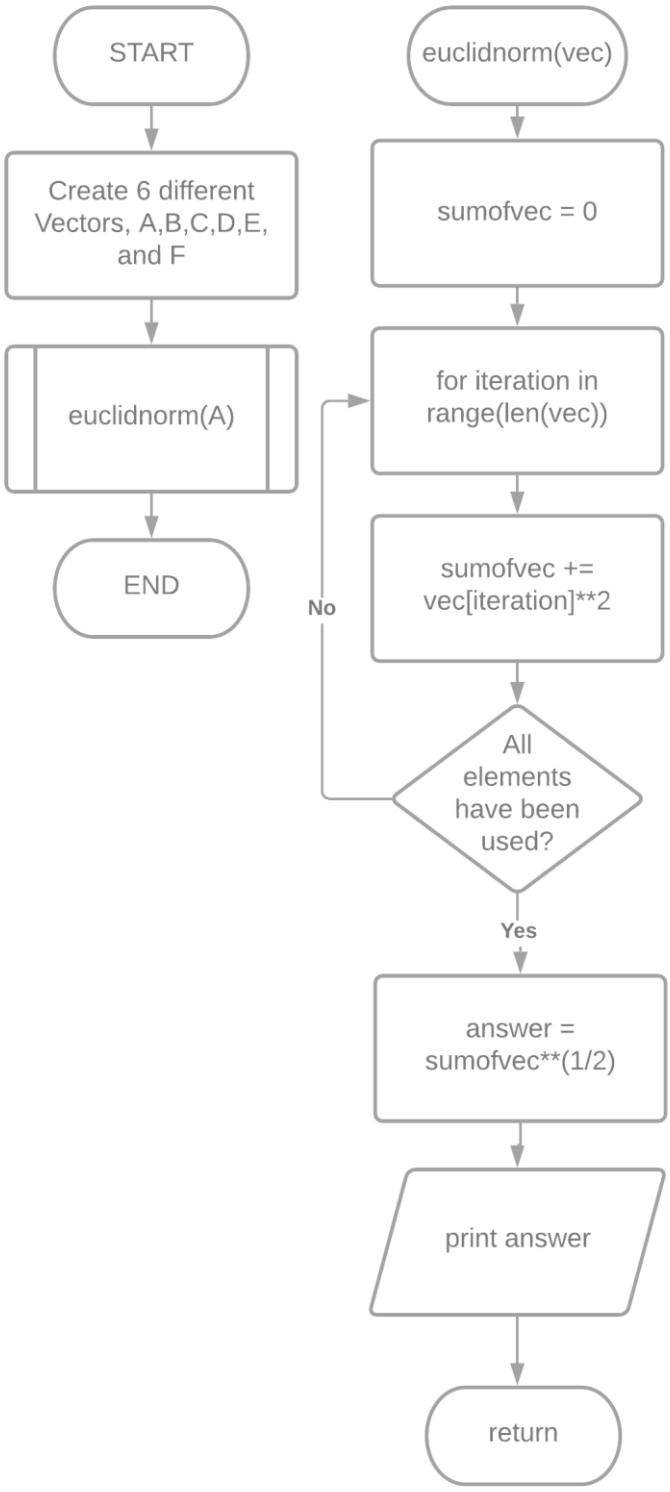



Figure 1 Flowchart for task 1

The first deliverable is achieved by first creating six different vectors. Next, using one of the created vectors as an argument to the created function euclidnorm(). Inside the function, it would then compute for the modulus of the vector. Using a for loop and iterating it based on the length of the argument passed. Inside the loop, it would compute for the sum of each element squared. Then, getting the square root of the sum of each element named sumofvec. Then printing the answer and returns the result when the function is called.

The second deliverable is to create another explicit function that solves for the inner product of two vectors using the given formula and comparing it to the built-in function np.inner().

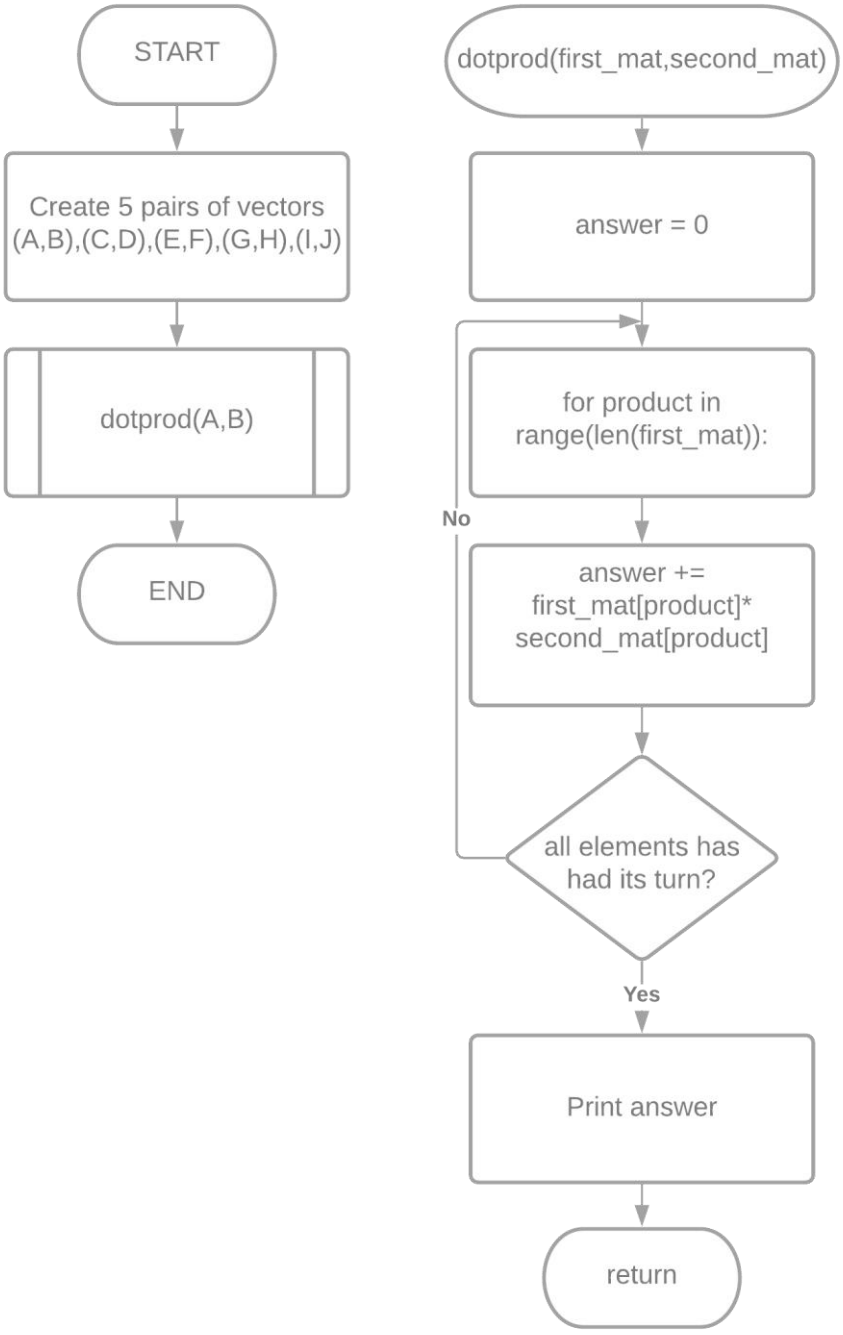



Figure 2 Flowchart for task 2

The second deliverable is achieved by first creating five pairs of vectors, (A, B), (C, D), (E, F), (G, H), (I, J) with elements not lower than five elements. Next, using a pair of vectors as an argument to the dotprod() function to compute for the inner product of the two vectors. Using a for loop and iterating it based on the range of the length of any of the two arguments passed. Inside the loop, to compute for the inner product, multiply each of the nth indices of the first and second vector. Lastly, print the answer and return the result when the function is called.

The third deliverable is to code the vector operation using the given vector values and plotting using a 3D plot.

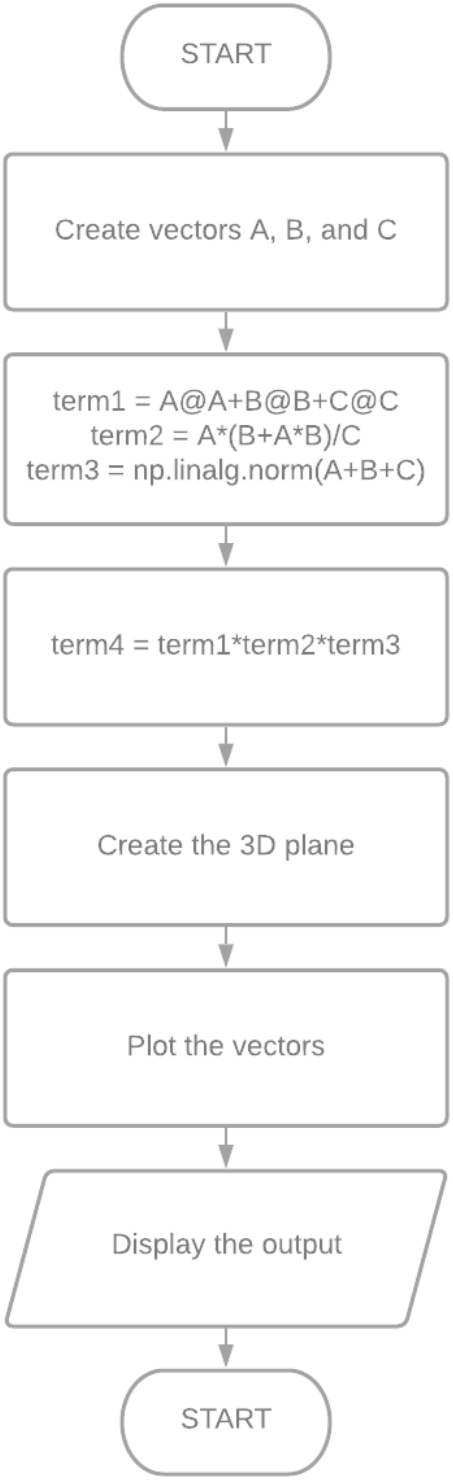



Figure 3 Flowchart for task 3

To achieve the third deliverable, create 3 vectors, A, B, and C. Then separate the equation into four different variables, term1, term2, term3, and term4. The term1 variable is

the sum of the dot product of the vector itself. The term2 is the product of A and the quantity of the sum of A and B multiplied to A all over C. The term3 is the modulus of the sum of A, B, and C. Then to get term4, multiply term1, term2, and term3. Next is to define the 3D plane using plt.figure() and assigning a variable named “fig” and fig.gca() and assigning a variable named “ax” function to create a 3D plane [1][2]. Then using the ax variable to set the limits on x, y, z using ax.set_xlim(), ax.set_ylim(), ax.set_zlim() functions [3][4][5]. Lastly, plot the vectors using ax.quiver() to plot an arrow to the 3D plane and display the output using plt.show() [6][7].

# III. Results

The codes for task 1 is seen in figure 4. The activity wants to compute for the modulus of a vector without using the built-in functions of NumPy.

```
def euclidnorm(vec):
    sumofvec = 0
    for iteration in range(len(vec)):
        sumofvec += vec[iteration]**2
    answer = sumofvec**(1/2)
    return print("The modulus of the vector", letter, "using the created function:",answer)



A = [1,2,3,4]
B = [2,4,6,8]
C = [3,6,8,12]
D = [4,8,12,16]
E = [5,10,15,20]
F = [6,12,18,24]

listofvec = [A,B,C,D,E,F]
listoflet = ["A","B","C","D","E","F"]

for display,letter in zip(listofvec,listoflet):
    euclidnorm(display)
    print("The modulus of the vector", letter, "using the np.linalg.norm() function:", np.linalg.norm(display), "\n" )
```

Figure 4 Codes for task 1

First, create a function named euclidnorm() with a parameter vec. Inside the function, assigning a variable named sumofvec equal to 0. Next, creating a for loop that iterates based on the length of elements of the vec parameter. In the loop, every time the loop iterates an element, it would access the index of the vec parameter and squaring the element on that index, and adding it to the sumofvec variable. After the loop ends, taking the value in the sumofvec variable and getting the square root of it, and assigning it to a new variable named answer. Last, returning the answer variable.

Outside the function, creating six different vectors with elements not lower than four. These vectors are going to be used as an argument for the created function euclidnorm() to

compute for each modulus of the created vector. The output for each vector is seen in Figure 5.

```
The modulus of the vector A using the created function: 5.477225575051661
The modulus of the vector A using the np.linalg.norm() function: 5.477225575051661

The modulus of the vector B using the created function: 10.954451150103322
The modulus of the vector B using the np.linalg.norm() function: 10.954451150103322

The modulus of the vector C using the created function: 15.905973720586866
The modulus of the vector C using the np.linalg.norm() function: 15.905973720586866

The modulus of the vector D using the created function: 21.908902300206645
The modulus of the vector D using the np.linalg.norm() function: 21.908902300206645

The modulus of the vector E using the created function: 27.386127875258307
The modulus of the vector E using the np.linalg.norm() function: 27.386127875258307

The modulus of the vector F using the created function: 32.863353450309965
The modulus of the vector F using the np.linalg.norm() function: 32.863353450309965
```

Figure 5 Output for task 1

Comparing the result of the created function euclidnorm() to the built-in NumPy function np.linalg.norm(). The answer is the same when using the created function euclidnorm() and the built-in NumPy function np.linalg.norm().

The codes for task 2 is seen in figure 6. In task 2, it aims to compute for the inner product of a pair of vectors without using the built-in functions of NumPy.

```
def dotprod(first_vec,second_vec):
    answer = 0
    for product in range(len(first_vec)):
        answer += first_vec[product]*second_vec[product]
    return print("The dot product using the created function:", answer)

A = [1,2,3,4,5]
B = [2,4,6,8,10]

C = [3,6,8,12,15]
D = [4,8,12,16,20]

E = [5,10,15,20,25]
F = [6,12,18,24,30]

G = [7,14,21,28,35]
H = [8,16,24,32,40]

I = [9,18,27,36,45]
J = [10,20,30,40,50]

fveclist = [A,C,E,G,I]
sveclist = [B,D,F,H,J]

for first,second in zip(fmatrixlist,smatrixlist):
    dotprod(first,second)
    print("The product using the np.inner() function:", np.inner(first,second), "\n")
```

Figure 6 Codes for task 2

The function dotprod() with two parameters first_mat and second_mat, computes for the inner product of a pair of vectors. Inside the function, an integer 0 is assigned to the variable answer to be used to store the inner product of the vectors. Next, creating a for loop that iterates depending on the length of the first or second vector since in the equation the length of the two vectors should be equal. Inside the loop, it computes for the inner product of the pair of vectors by accessing the indices of the vectors and multiplying each of the elements then adding the product to the answer variable. Last, return the answer.

Outside the function, creating 5 pairs of vectors with elements not lower than five. Using these pairs of vectors as an argument for the created function dotprod() to find their inner products. The output is seen in figure 7.

```
The dot product using the created function: 110
The product using the np.inner() function: 110

The dot product using the created function: 648
The product using the np.inner() function: 648

The dot product using the created function: 1650
The product using the np.inner() function: 1650

The dot product using the created function: 3080
The product using the np.inner() function: 3080

The dot product using the created function: 4950
The product using the np.inner() function: 4950
```

Figure 7 Output for task 2

Comparing the result of the created function dotprod() to the built-in NumPy function np.inner(). There is no difference when using the created function dotprod() and the built-in function np.inner().

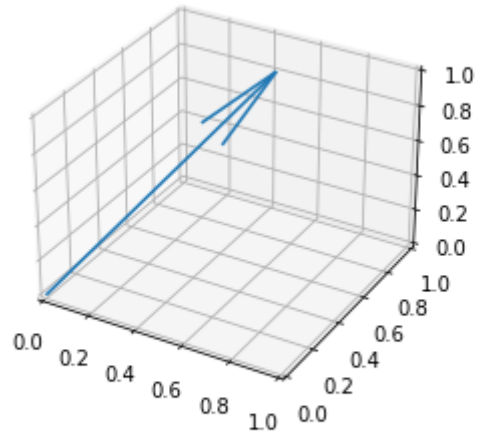
The codes for task 3 is seen in figure 8. The goal of task 3 is to perform the following vector operations using the given vector values.

```
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])

term1 = A@A+B@B+C@C
term2 = A*(B+A*B)/C
term3 = np.linalg.norm(A+B+C)
term4 = term1*term2*term3

print(term4)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.set_xlim(0,1)
ax.set_ylim(0,1)
ax.set_zlim(0,1)

ax.quiver(0,0,0,term4[0],term4[1],term4[2])
plt.show()
```

Figure 8 Codes for task 3

First, creating vectors A, B, and C using np.array(). Then initializing four new variables, term1, term2, term3, and term4. The term1 variable is the sum of the dot product of each vector by itself. The second variable, term2, is the product of A to the quantity of B+A*B then divided by C. The third variable, term3, is the modulus of the sum of A, B, and C. Then finding the product of term1, term2, and term3 to get the answer of the given equation. The output is seen in figure 9.

```
[0.34769805 1.13001866 0.6953961 ]
```




Expected answer:

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

Figure 9 Output for task 3

The programmer got the same answer as the expected answer, [0.34769805 1.13001866 0.6953961 ]. The line in figure 9 is the visualization of the final vector from the vector operations and vector values.

# IV. Conclusion

The laboratory activity discussed different vector operations using NumPy and visualizing the vector in a 3D plane using Matplotlib. The activity compared the explicit functions created by the programmer and the built-in functions of NumPy in performing vector operations. The result was that there was no difference between the two. Using the built-in functions in NumPy would make performing vector operations faster and easier. It is also easier

to visualize the vectors using the functions in Matplotlib especially if the values are not whole numbers as it is harder to plot when doing it manually.

# References


[1] Matplotlib, “matplotlib.pyplot.figure,” 2020. https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.figure.html.

[2] Matplotlib, “matplotlib.pyplot.gca,” 2020. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.gca.html.

[3] Matplotlib, “matplotlib.axes.Axes.set_xlim,” 2020. https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.axes.Axes.set_xlim.html.

[4] Matplotlib, “matplotlib.axes.Axes.set_ylim,” 2020. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.set_ylim.html.

[5] Matplotlib, “mpl_toolkits.mplot3d.axes3d.Axes3D,” 2020. https://matplotlib.org/3.3.2/api/_as_gen/mpl_toolkits.mplot3d.axes3d.Axes3D.html?highlight=set_zlim#mpl_toolkits.mplot3d.axes3d.Axes3D.set_zlim.

[6] Matplotlib, “Quiver,” 2020. https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html.

[7] Matplotlib, “matplotlib.pyplot.show,” 2020. https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.show.html.


# Appendix A

Github Repository Link:

https://github.com/Loreynszxc/Linear-Algebra-Lab-4