Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 4

Vector Operations

Submitted by:
Sustento, Myke Alvin E.

Instructor:
Engr. Dylan Josh D. Lopez

October 26, 2020

I. Objectives

This laboratory activity aims to implement the principles and techniques of vector operations in python such as vector addition, vector subtraction, vector multiplication, vector division, modulus of a vector, and vector dot product to perform and visualize vector operations using Python as the programming language.

II. Method

The practices consists of using the different vector operations such as vector addition, vector subtraction, vector multiplication, vector division, modulus of a vector, and vector dot product in Python. These vector operations are used in order to achieve the deliverables of the activity which is to perform and visualize vector operations using Python.

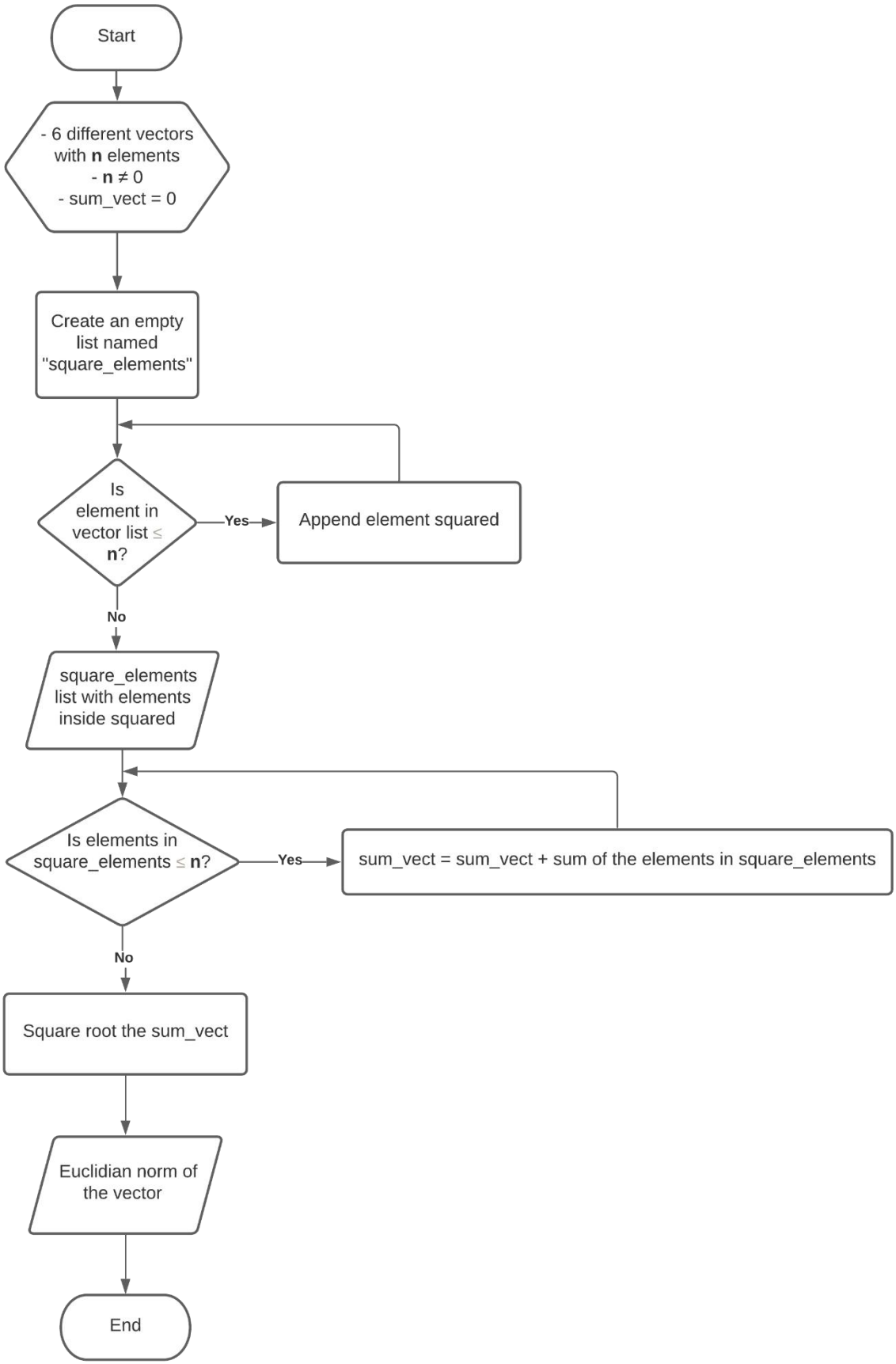

Figure 1 Flowchart for Euclidian Norm Algorithm in Task 1

Figure 1 shows the flowchart in how the algorithm created by the code work to solve the modulus of a vector. The formula that was used to achieve the modulus was the Euclidian norm formula which is $\left||X|\right| = \sqrt{\sum_{n=1}^{N} x_n^2}$. Basically, the Euclidian norm formula is the squareroot of the sum of the squared elements of the vector. The algorithm in the flowchart started by squaring the elements of the vector in the list which would loop depending on how many elements were given. After the squared elements loop, it would proceed to another loop which would get the sum of the squared elements. In the formula, this would be the $\sum$ and all that is needed to do to get the modulus is to get the square root of the sum or $\sqrt{\sum}$ which is the last process in the flow chart. Lastly, this would output the Euclidian norm of the vector and then end the flowchart.

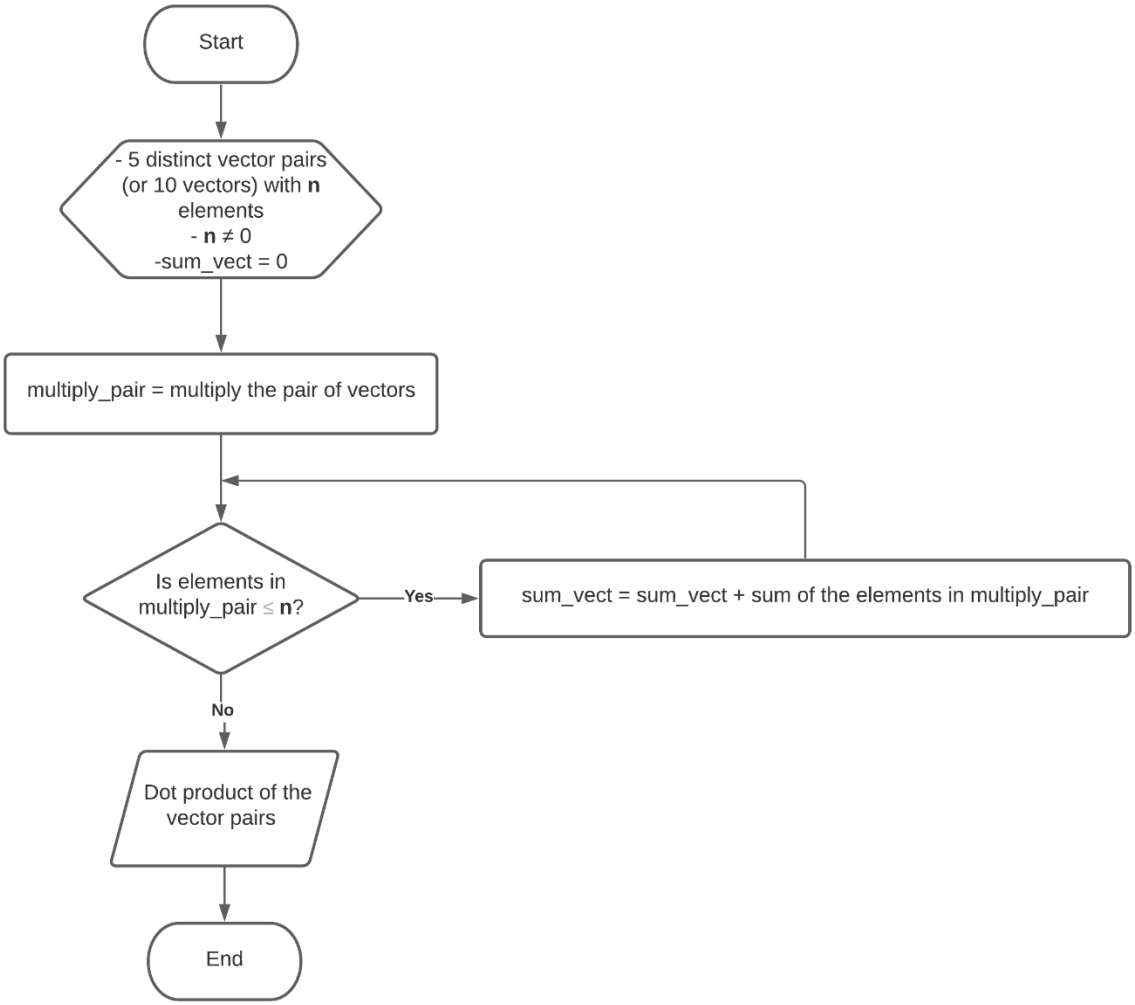

Figure 2 Flowchart for Dot Product in Task 2

Figure 2 shows the flowchart for the algorithm of the dot product in task 2. In order to get the dot product, take the element-wise product of the vectors and then take the sum of the product. This can be seen in the flowchart where the the 1st process multiplies the pair of vectors and the loop would take the sum of the elements which would produce the dot product of the vector pairs output and end the flowchart.

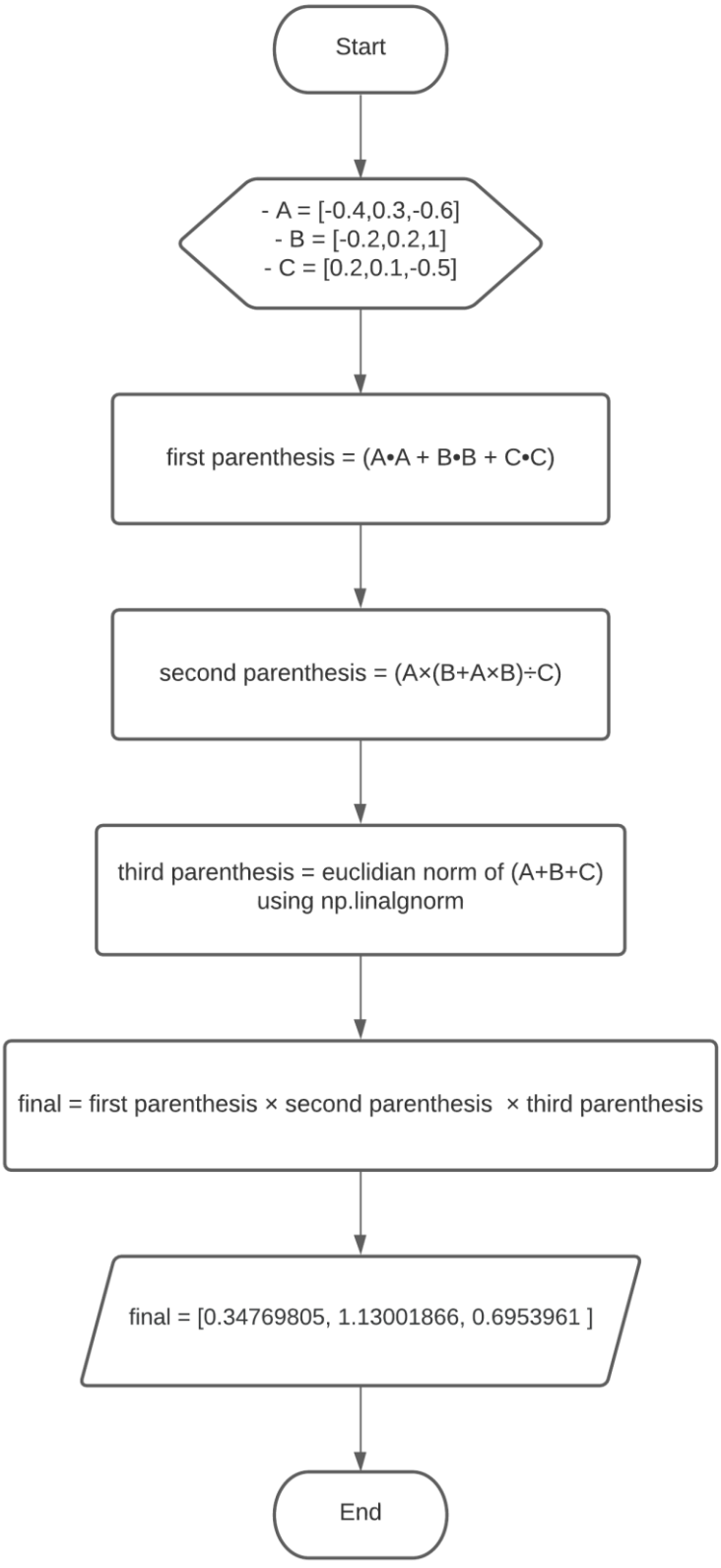

Figure 3 Flowchart for the Given Vector Operation in Task 3

Figure 3 shows the flowchart for the vector operation $\left((A^2 + B^2 + C^2) \times \left(A \times \frac{(B + A \times B)}{C}\right)\right) \times ||A + B + C||$. The parenthesis was split into 3 for easier representation

and reading of the operation. The first parenthesis is equal to $A \cdot A + B \cdot B + C \cdot C$. The $\cdot$ represents a dot product since the square of a vector is essentially the dot product of the vector with itself. The second parenthesis is the $\left(A \times \frac{(B + A \times B)}{C}\right)$ which is basically just a PEMDAS rule with vectors. The code would automatically compute this if the PEMDAS rule was properly followed. The third parenthesis is the Euclidian norm of the sum of A, B, and C. In order to achieve this, the np.linalg.norm [1] was used.

III. Results

```
import math
vect1 = np.array([1,2,3,4])
vect2 = np.array([6,5,4,3])
vect3 = np.array([0,2,4,6])
vect4 = np.array([10,8,6,4])
vect5 = np.array([1,3,5,7])
vect6 = np.array([11,9,7,5])
```

Figure 4 6 Different Vectors with 4 Elements in Task 1

Figure 4 is the 6 different vectors with 4 elements that was required for task 1. Since the instruction stated to not use any NumPy's preset functions, the modulus of this vectors will be solved by the personal modulus function of the programmer using the Euclidian norm formula. The math module was added to enable the utilization of the sqrt() function which finds the squareroot of value in its parameter [2].

```
def my_euc_norm(list):
    # squares the elements
    square_elements = []
    sum_vect = 0
    for element in list:
        square_elements.append(element ** 2)

# # list comprehension version of the for loop
#     square_elements = [element ** 2 for element in list]
#     print(square_elements)

    # gets the sum of the squared elements
    for elements in range(len(square_elements)):
        sum_vect = sum_vect + square_elements[elements];
    euc_norm = math.sqrt(sum_vect)
    return euc_norm

print("Euclidian Norm using my Function")
print("the euclidian norm of vector 1:",  my_euc_norm(vect1))
print("the euclidian norm of vector 2:",  my_euc_norm(vect2))
print("the euclidian norm of vector 3:",  my_euc_norm(vect3))
print("the euclidian norm of vector 4:",  my_euc_norm(vect4))
print("the euclidian norm of vector 5:",  my_euc_norm(vect5))
print("the euclidian norm of vector 6:",  my_euc_norm(vect6))
```

Figure 5 The Modulus Function in Task 1

```
Euclidian Norm using my Function
the euclidian norm of vector 1: 5.477225575051661
the euclidian norm of vector 2: 9.273618495495704
the euclidian norm of vector 3: 7.483314773547883
the euclidian norm of vector 4: 14.696938456699069
the euclidian norm of vector 5: 9.16515138991168
the euclidian norm of vector 6: 16.61324772583615
```

Figure 6 Results of the Modulus Function in Task 1

Figure 5 is the modulus function created by the programmer. The function works by following the Euclidian norm formula. Firstly, the program would square the elements inside the list which can be seen in the first loop. Secondly, the elements in the squared elements list would be added together to produce the sum. Lastly, The sum would then be squarerooted

using the sqrt() function to produce the modulus of the vectors using the Euclidian norm formula which can be seen in figure 6.

```
# proving my function with numpy
def numpy_euclidian_norm(vect):
    numpy_euc_norm = np.linalg.norm(vect)
    return numpy_euc_norm

print("Euclidian Norm using Numpy Library")
print("the euclidian norm of vector 1:",  numpy_euclidian_norm(vect1))
print("the euclidian norm of vector 2:",  numpy_euclidian_norm(vect2))
print("the euclidian norm of vector 3:",  numpy_euclidian_norm(vect3))
print("the euclidian norm of vector 4:",  numpy_euclidian_norm(vect4))
print("the euclidian norm of vector 5:",  numpy_euclidian_norm(vect5))
print("the euclidian norm of vector 6:",  numpy_euclidian_norm(vect6))
```

```
Euclidian Norm using Numpy Library
the euclidian norm of vector 1: 5.477225575051661
the euclidian norm of vector 2: 9.273618495495704
the euclidian norm of vector 3: 7.483314773547883
the euclidian norm of vector 4: 14.696938456699069
the euclidian norm of vector 5: 9.16515138991168
the euclidian norm of vector 6: 16.61324772583615
```

Figure 7 Proving the Modulus Function with NumPy

To prove the validity of the modulus function created by the programmer. NumPy's preset function np.linalg.norm() [1] was used to get the modulus of the 6 vectors. It can be seen that the results is the same which proves the modulus function of the programmer accurate.

```
#1st pair
vect1 = np.array([2,5,4,3,4])
vect2 = np.array([8,4,6,7,5])

#2nd pair
vect3 = np.array([8,7,4,6,9])
vect4 = np.array([8,4,5,1,6])

#3rd pair
vect5 = np.array([8,4,5,2,3])
vect6 = np.array([2,6,4,5,6])

#4th pair
vect7 = np.array([5,9,4,6,2])
vect8 = np.array([4,8,8,5,8])

#5th pair
vect9 = np.array([5,7,8,5,4])
vect10 = np.array([6,5,4,9,8])
```

Figure 8 5 Distinct Pairs Vectors with 5 Elements in Task 2

Figure 8 shows the 5 distinct pairs vectors with 5 elements that was asked in the instruction of task 2. The instruction provided in the lab activity is to solve for the dot product of the vector pairs without using the NumPy's preset functions.

```
def my_dot_product(vec1, vec2):
    # multiplies the vector pairs
    multiply_pair = vec1*vec2
    # gets the sum of the multiply_pair
    sum_vect = 0
    for elements in range(len(multiply_pair)):
        sum_vect = sum_vect + multiply_pair[elements];
    return(sum_vect)

print("Dot Product using my Function")
print("The dot product of vector 1 and vector 2:",  my_dot_product(vect1,vect2))
print("The dot product of vector 3 and vector 4:",  my_dot_product(vect3,vect4))
print("The dot product of vector 5 and vector 6:",  my_dot_product(vect5,vect6))
print("The dot product of vector 7 and vector 8:",  my_dot_product(vect7,vect8))
print("The dot product of vector 9 and vector 10:",  my_dot_product(vect9,vect10))
```

Figure 9 The Programmer's Dot Product Function in Task 2

```
Dot Product using my Function
The dot product of vector 1 and vector 2: 101
The dot product of vector 3 and vector 4: 172
The dot product of vector 5 and vector 6: 88
The dot product of vector 7 and vector 8: 170
The dot product of vector 9 and vector 10: 174
```

Figure 10 The Result of the Dot Product Function in Task 2

Figure 9 shows the programmer's personal dot product function in task 2. The algorithm works by multiplying the vector pairs provided in figure 8 and then getting the sum of the multiplied vector pairs. This will result in the dot product of the vector pairs provided in figure 10.

```
# proving my function with numpy
def numpy_dot_product(vec1,vec2):
    numpy_dot_prod = np.inner(vec1, vec2)
    return numpy_dot_prod

print("Dot Product using Numpy Library")
print("The dot product of vector 1 and vector 2:", numpy_dot_product(vect1,vect2))
print("The dot product of vector 3 and vector 4:", numpy_dot_product(vect3,vect4))
print("The dot product of vector 5 and vector 6:", numpy_dot_product(vect5,vect6))
print("The dot product of vector 7 and vector 8:", numpy_dot_product(vect7,vect8))
print("The dot product of vector 9 and vector 10:", numpy_dot_product(vect9,vect10))
```

```
Dot Product using Numpy Library
The dot product of vector 1 and vector 2: 101
The dot product of vector 3 and vector 4: 172
The dot product of vector 5 and vector 6: 88
The dot product of vector 7 and vector 8: 170
The dot product of vector 9 and vector 10: 174
```

Figure 11 Proving the Dot Product Function with NumPy

To prove if the dot product function by the programmer is accurate, the Numpy's preset function np.inner() function was used to get the inner product of two arrays or vectors [3]. The result in figure 11 and figure 10 is the same proving that the algorithm accurately gets the dot product of the vector pairs.

```
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_p = (A@A + B@B + C@C)
second_p = (A*(B+A*B)/C)
third_p = (np.linalg.norm(A+B+C))

final = first_p * second_p * third_p

print(final)
```

```
[0.34769805 1.13001866 0.6953961 ]
```

Expected answer:

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

Figure 12 The Code and Output of the Vector Operation in Task 3

Figure 12 is the program and output in the vector operation given in task 3. The output of the program and the expected output is exactly the same since the algorithm for the vector operation was properly executed.

```
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.quiver(0, 0, 0, final[0], final[1], final[2], colors='r')
ax.set_xlim([0, 1.2])
ax.set_ylim([0, 1.2])
ax.set_zlim([0, 1.2])
plt.show()
```

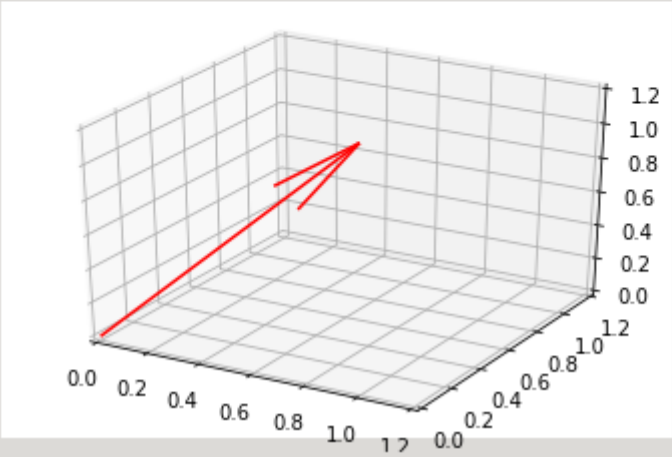

Figure 13 The Code and 3D Plot of the Resulting Vector in Task 3

Since the resulting vector was found in figure 12, a 3d plot could be provided by using the program provided in figure 13. The program starts by creating a figure using the plt.figure() [4]. To get a 3d projection of the figure, fig.gca() was used with the parameter, projection='3d' [5]. The function quiver() was then used to plot a 3d field of arrows with the X, Y, and Z parametesr as 0 and the U, V, and W parameters as final[0], final[1], and final[2] which is the resulting vectors. In order to show the whole line, the x, y ,and z axes was limited using the functions set_ xlim(), set_ ylim(), and set_ zlim() [6][7][9]. Lastly, plt.show() was used to display the figure [9]. As can be seen in the 3D plot, the quiver pointed in the correct values of the x-axis which is 0.34769805, y-axis which is 1.13001866, and z-axis which is 0.6953961.

IV. Conclusion

This laboratory activity imparted the knowledge of the vector operations: vector addition, vector subtraction, vector multiplication, vector division, modulus of a vector, and vector dot product. The laboratory acitivty also demonstrated that the progamming language, Python, can be used to execute those vector operations even without the help of the NumPy library. However, it was insinuated that NumPy is the simpler and more efficient route to solve vector operations. Lastly, these vector operations can be visualized using the preset functions given by the matplotlib library.

References

[1]"numpy.linalg.norm — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html. [Accessed: 25- Oct- 2020].

[2]"Python math.sqrt() Method", W3schools.com, 2020. [Online]. Available: https://www.w3schools.com/python/ref_math_sqrt.asp. [Accessed: 26- Oct- 2020].

[3]"numpy.inner — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.inner.html. [Accessed: 26- Oct- 2020].

[4]"matplotlib.pyplot.figure — Matplotlib 3.3.2 documentation", Matplotlib.org, 2020. [Online]. Available: https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.figure.html. [Accessed: 26- Oct- 2020].

[5]"matplotlib.pyplot.gca — Matplotlib 3.1.2 documentation", Matplotlib.org, 2020. [Online]. Available: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.gca.html. [Accessed: 26- Oct- 2020].

[6]"matplotlib.axes.Axes.set_xlim — Matplotlib 3.3.1 documentation", Matplotlib.org, 2020. [Online]. Available: https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.axes.Axes.set_xlim.html. [Accessed: 26- Oct- 2020].

[7]"matplotlib.axes.Axes.set_ylim — Matplotlib 3.3.2 documentation", Matplotlib.org, 2020. [Online]. Available: https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.axes.Axes.set_ylim.html. [Accessed: 26- Oct- 2020].

[8]"mplot3d API — Matplotlib 2.0.2 documentation", Matplotlib.org, 2020. [Online]. Available: https://matplotlib.org/mpl_toolkits/mplot3d/api.html. [Accessed: 26- Oct- 2020].

[9]"matplotlib.pyplot.show — Matplotlib 3.3.2 documentation", Matplotlib.org, 2020. [Online]. Available: https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.show.html. [Accessed: 26- Oct- 2020].

Appendix

Github link - https://github.com/Sus102/LA_Lab/tree/master/LA_LAB_4