Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 4

Vector Operations

Submitted by:
Columba, Lorenzo Miguel L.

Instructor:
Engr. Dylan Josh D. Lopez

Nov 5, 2020

I. Objectives

The purpose of this laboratory activity is to apply the principles and techniques of vector operation in "Python", such as vector addition, vector subtraction, vector multiplication, vector division, the modulus of a vector, and the vector operation as a docking to the vector dot product and visualizing vector in python programming language.

II. Method

These practices include the use of various vector operations in Python, such as vector addition, vector subtraction, vector multiplication, vector division, vector coefficients, and vector dot product. These vector manipulation operations are used to capture the artifacts of activities that perform and visualize vector manipulations using Python.

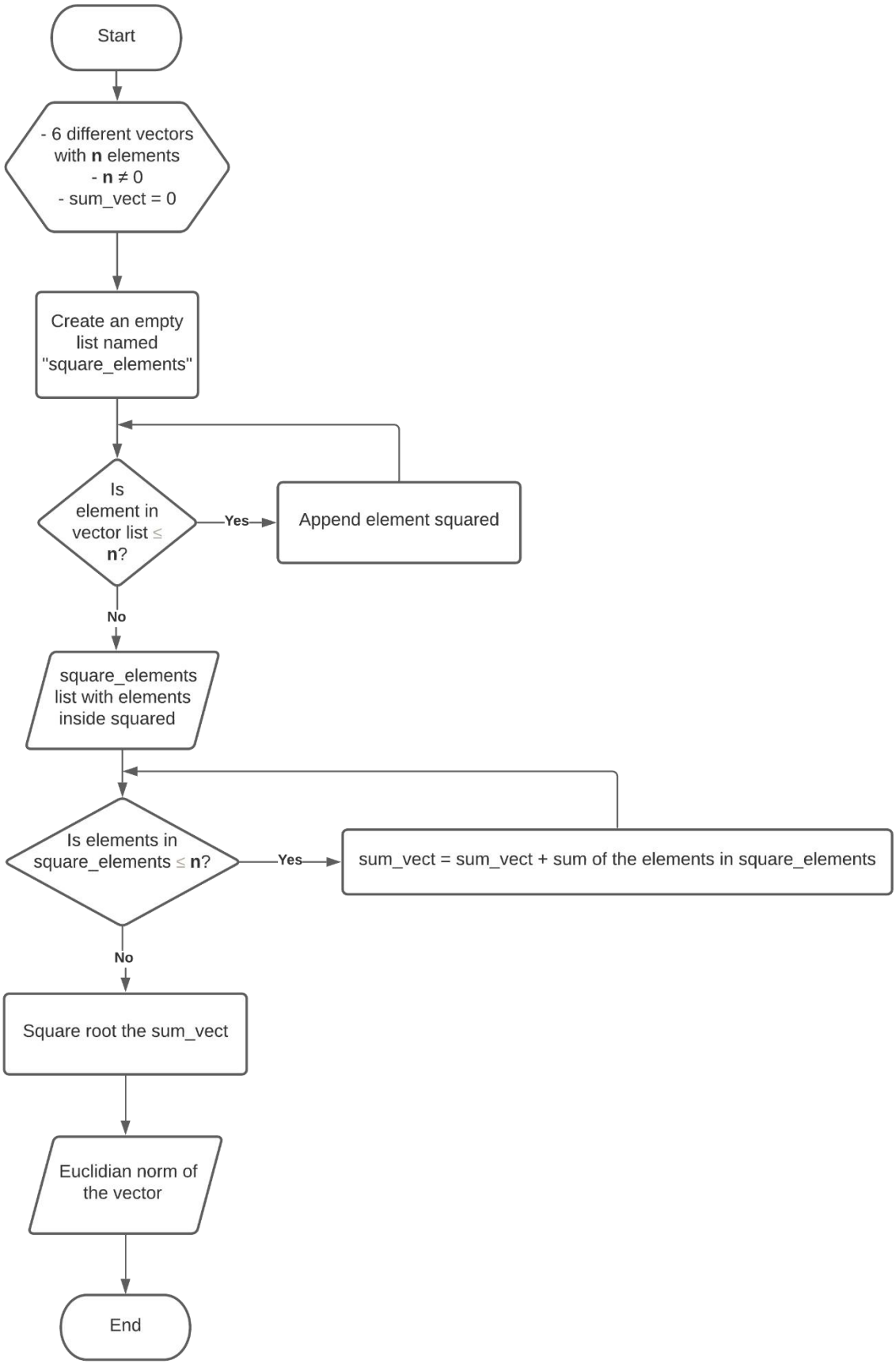

Figure 1 Flowchart for Euclidian Norm Algorithm in Task 1

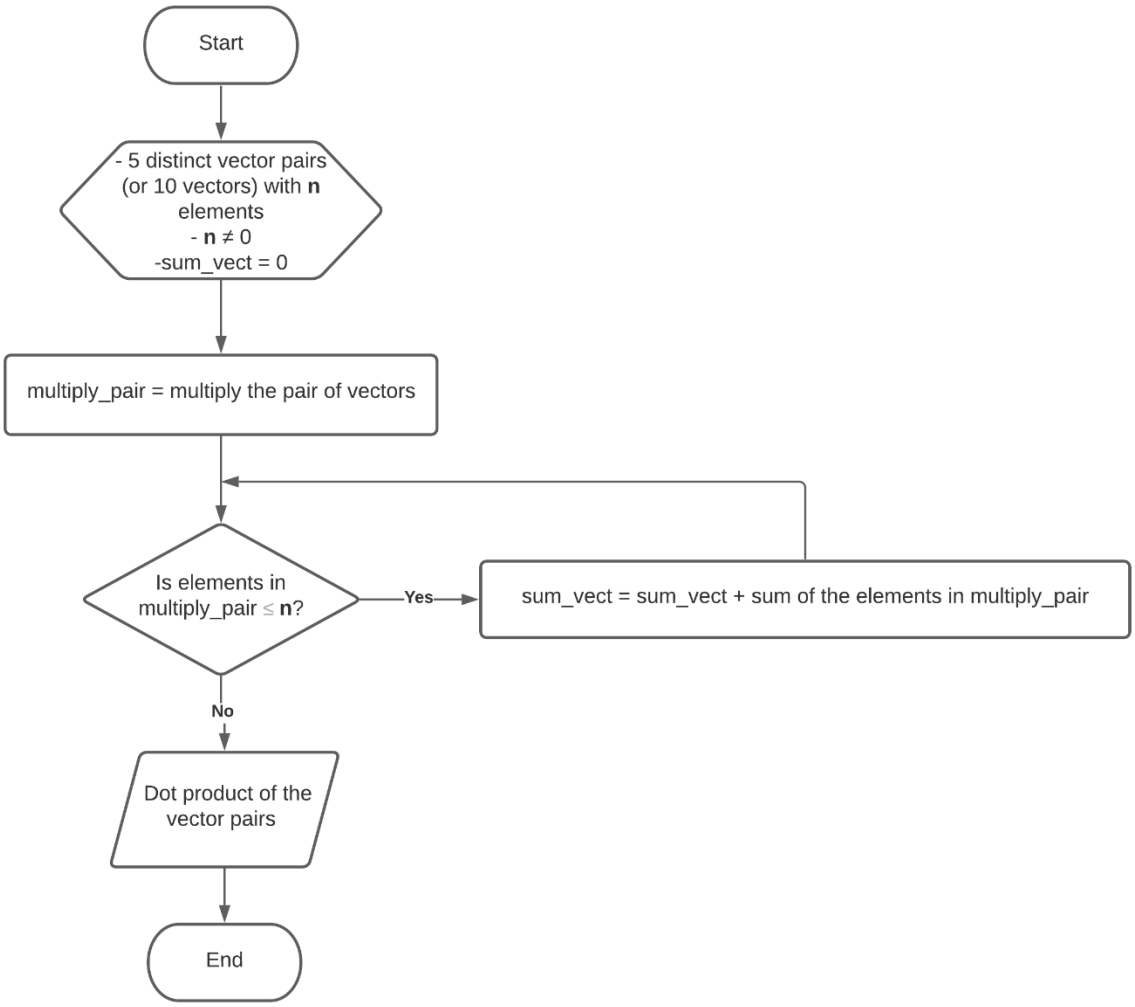

Figure 2 Flowchart for Dot Product in Task 2

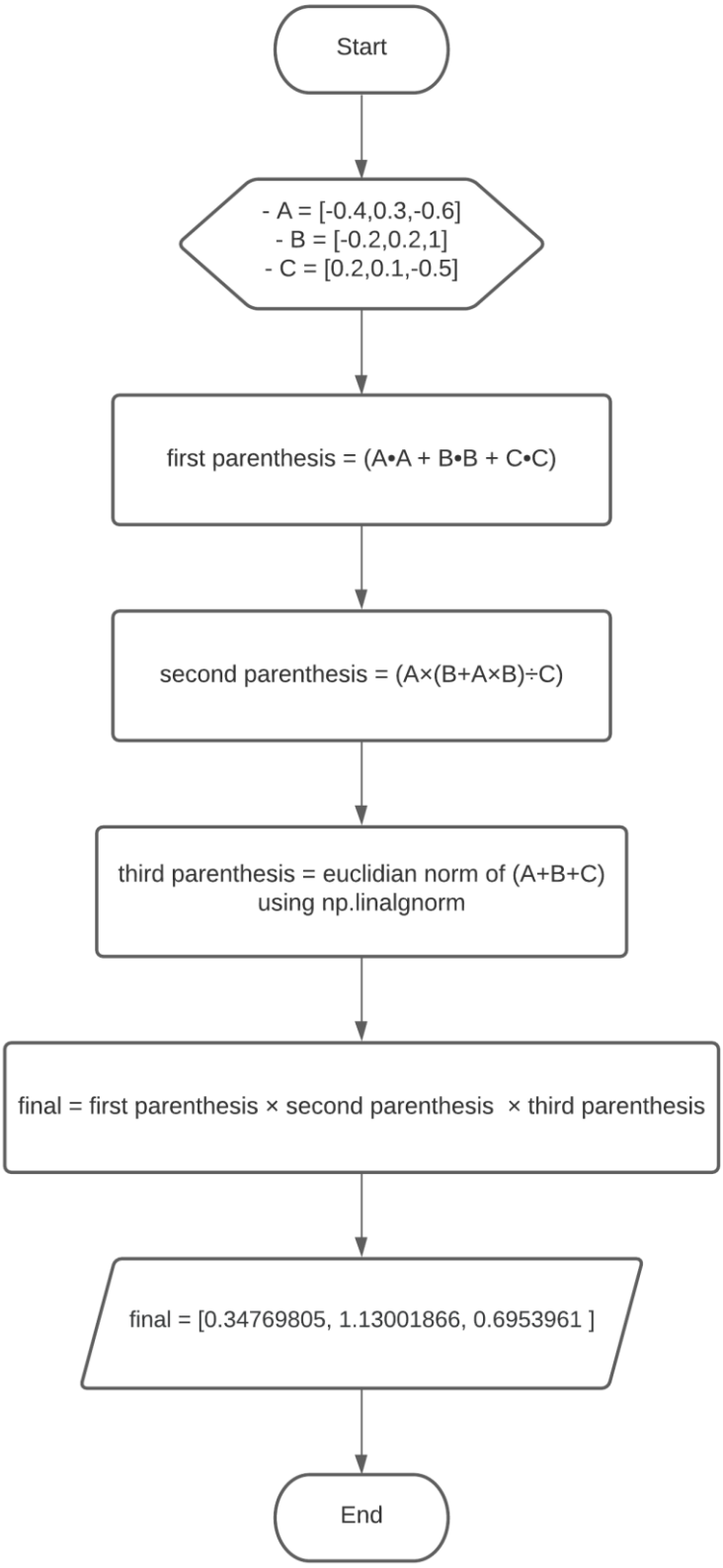

Figure 3 Flowchart for the Given Vector Operation in Task 3

III. Results

```
import math
vect1 = np.array([11,2,3,4])
vect2 = np.array([6,51,4,3])
vect3 = np.array([10,21,4,16])
vect4 = np.array([10,18,6,4])
vect5 = np.array([4,3,2,1])
vect6 = np.array([1,2,3,4])
```

Figure 4 6 Different Vectors with 4 Elements in Task 1

In Figure 4 shows the 6 different vectors with 4 elements that were required for task 1. Since the declared instruction not to use the predefined functions of NumPy, the module of these vectors is resolved by the personal module function of the programmer using the Euclidean standard formula. The math module was added to allow the use of the sqrt () function which finds the square root of the value in its parameter [2].

```
def my_euc_norm(list):

    square_elements = []
    sum_vect = 0
    for element in list:
        square_elements.append(element ** 2)


    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(" Vector 1:",  my_euc_norm(vect1))
print(" Vector 2:",  my_euc_norm(vect2))
print(" Vector 3:",  my_euc_norm(vect3))
print(" Vector 4:",  my_euc_norm(vect4))
print(" Vector 5:",  my_euc_norm(vect5))
print(" Vector 6:",  my_euc_norm(vect6))
```

Figure 5 The Modulus Function in Task 1

```
Euclidian Norm using my Function
 Vector 1: 12.247448713915890
 Vector 2: 51.594573358057720
 Vector 3: 28.513154858766505
 Vector 4: 21.817424229271428
 Vector 5: 5.4772255750516610
 Vector 6: 5.4772255750516610
```

Figure 6 Results of the Modulus Function in Task 1

In 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.

```
def numpy_euclidian_norm(vect):
    numpy_euc_norm = np.linalg.norm(vect)
    return numpy_euc_norm

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

```
Euclidian Norm using Numpy Library
 Vector 1: 12.24744871391589
 Vector 2: 51.59457335805772
 Vector 3: 28.513154858766505
 Vector 4: 21.817424229271428
 Vector 5: 5.477225575051661
 Vector 6: 5.477225575051661
```

Figure 7 Proving the Modulus Function with NumPy

Proving the validity of the modulus function created by the programmer. To get the modulus of 6 vectors I used NumPy's pre-defined function np.linalg.norm () [1]. The results are the same and you can see that the programmer's modulus function is accurate.

```
vects1 = np.array([1,2,3,4,5])
vects2 = np.array([6,7,8,9,10])

vects3 = np.array([4,8,8,5,8])
vects4 = np.array([5,9,4,6,2])

vects5 = np.array([8,4,5,1,6])
vects6 = np.array([8,7,4,6,9])

vects7 = np.array([5,9,4,6,2])
vects8 = np.array([4,8,8,5,8])

vects9 = np.array([4,8,8,5,8])
vects10 = np.array([1,2,3,4,5])
```

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

In Figure 8 shows five different pair vectors with the five elements queried in the instructions in task 2. The procedure provided in the lab activity is to solve the dot product of vector pairs without using NumPy's preset functions.

```
def my_dot_product(vec1, vec2):

    multiply_pair = vec1*vec2

    sum_v = 0
    for elements in range(len(multiply_pair)):
        sum_v = sum_v + multiply_pair[elements];
    return(sum_v)

print("Dot Product using my Function")
print("Vector 1 and  2:",  my_dot_product(vects1,vects2))
print("Vector  3 and  4:",  my_dot_product(vects3,vects4))
print("Vector  5 and  6:",  my_dot_product(vects5,vects6))
print("Vector  7 and  8:",  my_dot_product(vects7,vects8))
print("Vector  9 and  10:",  my_dot_product(vects9,vects10))
```

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

```
Dot Product using my Function
Vector 1 and  2: 130
Vector  3 and  4: 170
Vector  5 and  6: 172
Vector  7 and  8: 170
Vector  9 and  10: 104
```

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

In Figure 9 shows the programmer's individual dot product function in Task 2. The algorithm works by multiplying the vector pairs provided in Figure 8 and then summing the vector pairs. This leads to the dot product of the vector pairs provided in Figure 10.

```
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

Proving if the dot product perform by the programmer is accurate, the Numpy's planned function np.inner() function was wont to get the inner product of 2 arrays or vectors [3]. The lead to figure 11 and figure 10 is that the same proving that the algorithmic program accurately gets the dot product of the vector pairs.

```
[29] 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@A + B@B + C@C)
     second = (A*(B+A*B)/C)
     third = (np.linalg.norm(A+B+C))

     final = first * second * third

     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

In Figure 12 shows the output in the vector operation given in task 3. The output of the program and the expected output are exactly the same because the vector operation algorithm was implemented correctly.

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

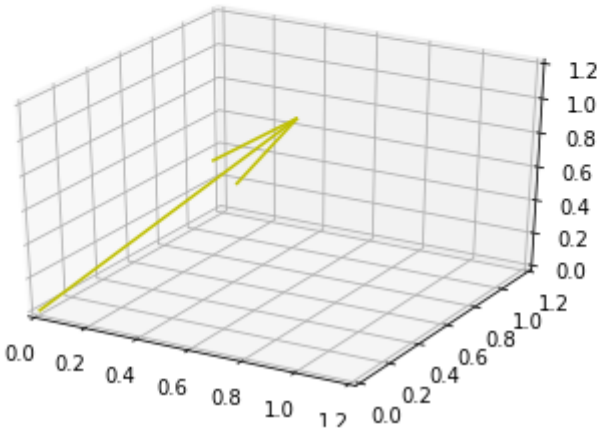

Figure 13 The Code and 3D Plot of the Resulting Vector in Task 3

The resulting vector is shown in Figure 12, so you can use the program shown in Figure 13 to provide a 3D plot. 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

In this laboratory activity provided knowledge of vector operations: vector addition, vector subtraction, vector multiplication, vector division, vector coefficients and vector dot products. Laboratory activities have also shown that they can be used to perform vector operations without the help of programming languages, Python, or unmeasurable libraries. Nevertheless, Numphy has been suggested to be an easier and more efficient way to solve its insurance business. Finally, these vector actions can be visualized using the preset functions provided 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/LorenzoMiguelColumba/LinearAlgebra_Lab4.git