Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 3

Linear Combinations and Vector Spaces

Submitted by:

Guy, Lawrence Adrian B.

Instructor:

Engr. Dylan Josh D. Lopez

October 25, 2020

I. Objectives

This laboratory activity aims to familiarize students with representing linear combinations in the 2-dimensional plane. The laboratory activity also aims to help students visualize spans using vector fields in Python. Lastly, the laboratory activity helps students perform vector field operations faster and easier using NumPy and Matplotlib.

II. Methods

The practices of this activity are performing linear combinations and vector spaces using scientific programming. Linear combinations are the sum of each product of scalars and matrices. Using NumPy and Matplotlib, to perform vector field operations and visualize the created vectors.

The deliverables of the activity are trying different linear combinations and using different scalar values, and making unique spans.

In task 1, the general linear equations with their vector forms are:

$$A = c \cdot (2x + y)$$

$$B = c \cdot (2x - y)$$

$$C = c \cdot y$$

$$A = c \cdot \begin{bmatrix} 2 \\ 1 \end{bmatrix}, B = c \cdot \begin{bmatrix} 2 \\ -1 \end{bmatrix}, C = c \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The function used to create the vectors is np.array(). The scalar values are created by using the np.arange() function, this function works by creating a range of numbers from start to end within a given interval [1]. Then plot the vectors using the plt.scatter() function [2].

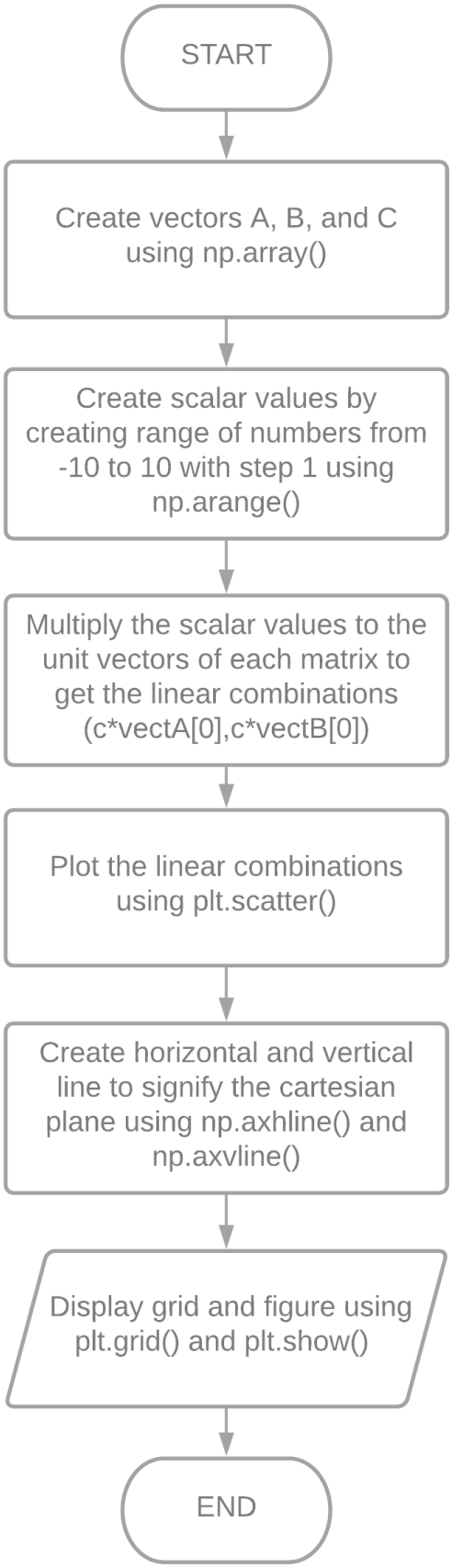

Figure 1 Flowchart for Task 1

In task 2, the general linear equations with their vector forms used are:

$$S_1 = \begin{cases} c_1 \cdot (2x + 3y) \\ c_2 \cdot (-4x + 3y) \end{cases}$$

$$S_1 = \left\{ c_1 \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix}, c_2 \cdot \begin{bmatrix} -4 \\ 3 \end{bmatrix} \right\}$$

$$S_2 = \begin{cases} c_1 \cdot (2y) \\ c_2 \cdot (2x) \end{cases}$$

$$S_2 = \left\{ c_1 \cdot \begin{bmatrix} 0 \\ 2 \end{bmatrix}, c_2 \cdot \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right\}$$

$$S_3 = \begin{cases} c_1 \cdot (2x + 2y) \\ c_2 \cdot (4x + 4y) \end{cases}$$

$$S_3 = \left\{ c_1 \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix}, c_2 \cdot \begin{bmatrix} 4 \\ 4 \end{bmatrix} \right\}$$

The function used to create the vectors is np.array(). The scalar values are created by using the np.arange() function, this function works by creating a range of numbers from start to end within a given interval [1]. Then using np.meshgrid() function that returns coordinate matrices and multiplying it to the unit vectors and getting their sum to get the span of the vectors as seen in the equation above [3]. Then plot the vectors using the plt.scatter() function [2].

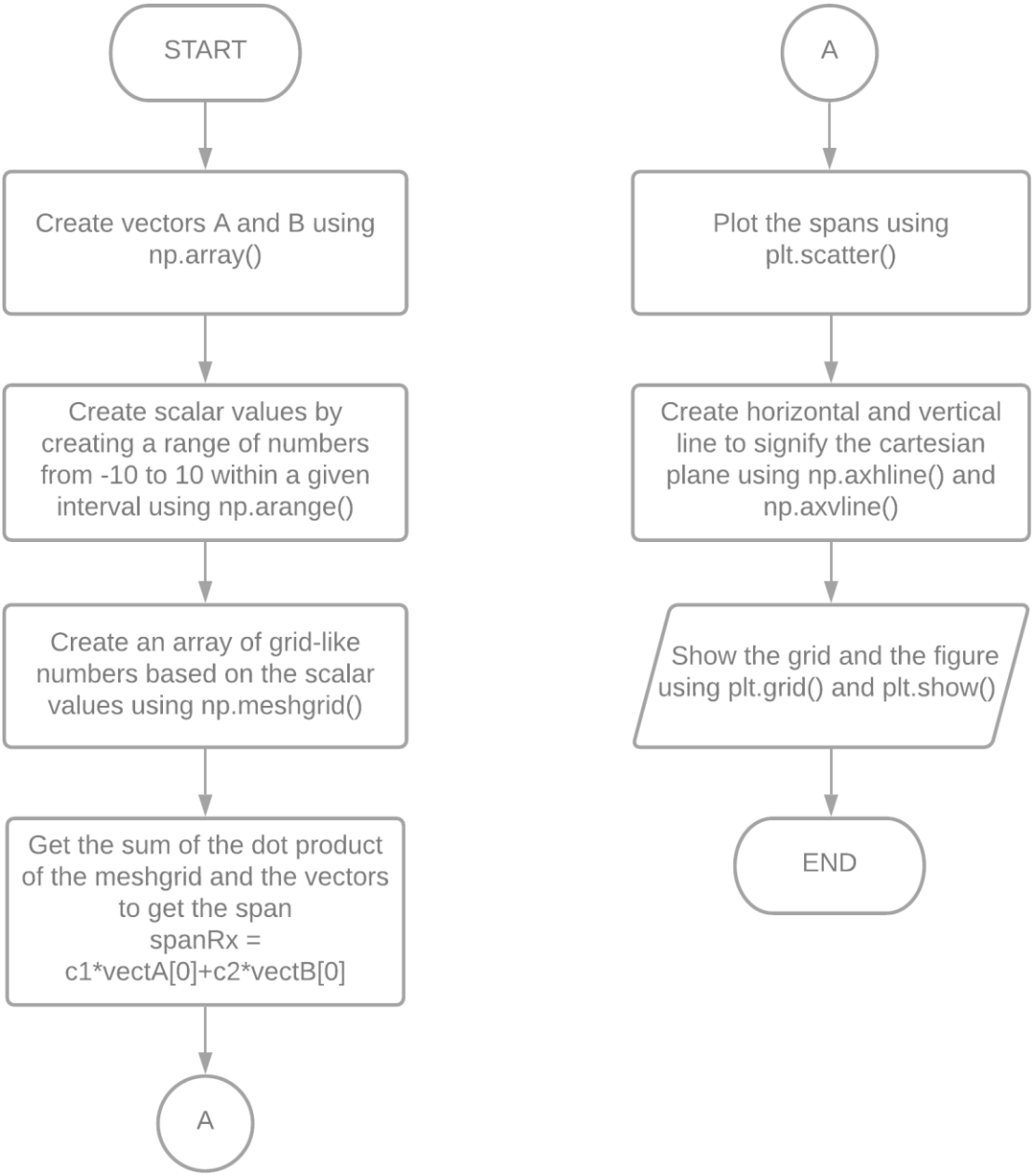

Figure 2 Flowchart for Task 2

In order to signify the cartesian plane, the functions plt.axhline() and plt.axvline() are used to create horizontal and vertical lines [4][5]. Then the function plt.grid() is used to put grid lines on the figure [6]. Lastly, using plt.show() function to display the figure [7].

III. Results

The laboratory activity used NumPy and Matplotlib to perform linear combinations and vector operations and visualize the result.

The output in task 1 as seen in figure 7, is achieved by initializing variables, vectA, vectB, and vectC as vectors using np.array() function as seen in figure 3.

```
#Creating vectors using np.array() function
vectA = np.array([2,1])
vectB = np.array([2,-1])
vectC = np.array([0,1])
```

Figure 3 Creating Vectors using np.array() function

Next, creating different scalar values by initializing a variable named “c” using np.arange() function that has 3 important parameters, first parameter is the starting value, the second parameter is the end value and the third parameter is the step value [1].

```
#creating a range of numbers from -10 to 10 with step 0.25 using np.arange()
c = np.arange(-10,10,0.25)
```

Figure 4 Creating scalar values using np.arange() function

Multiplying the c value to the unit vectors of vectA, vectB, and vectC to get different linear combinations. Then, using plt.scatter() function that has 4 parameters, the first and second parameters are the X and Y location, the third parameter is the marker size, the fourth parameter is the opacity of the marker [2].

```
#Multiplying c to the vectors to create a span
#Plotting the vectors using plt.scatter()
plt.scatter(c*vectA[0],c*vectA[1],s=10,alpha=0.75)
plt.scatter(c*vectB[0],c*vectB[1],s=10,alpha=0.75)
plt.scatter(c*vectC[0],c*vectC[1],s=10,alpha=0.75)
```

Figure 5 Using plt.scatter() to plot the vectors

Then, to create the visualization of a cartesian plane, using plt.axhline() and plt.axvline() to create a horizontal and vertical line to signify X and Y axes [4][5]. Lastly, define the grid using plt.grid() and display the output using plt.show() [6][7].

```
#creating lines on x and y axes
plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')

#shows the grid
plt.grid()

#shows the figure
plt.show()
```

Figure 6 Creating the cartesian plane

The output of the 3 linear combinations can be seen in figure 7. The 3 linear combinations aren't linearly dependent as there are no overlapping vectors. All the linear combinations intersected at the origin because the scalar value is ranging from -10 to 10 and the vectors would be multiplied to 0 at some point.

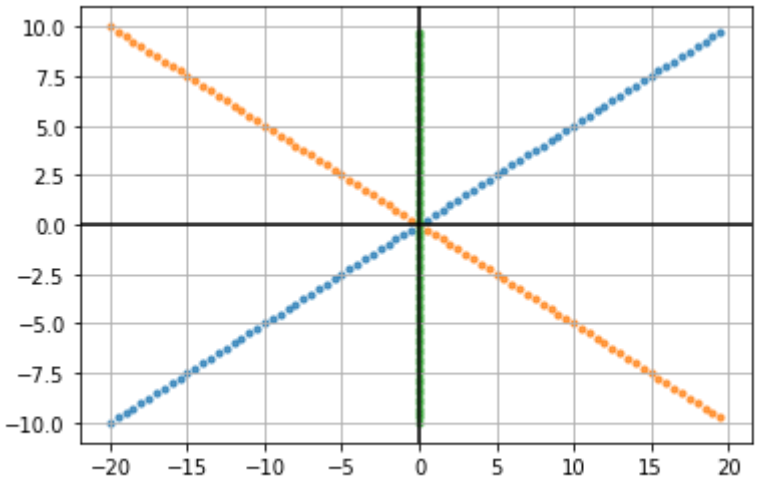

Figure 7 Output for Task 1

Task 2 has the same process as task 1 except this time the scalar values will be determined using np.meshgrid() function. Then using the result using np.meshgrid() function, to multiply it to the unit vectors and getting the sum to get the spanRx and spanRy.

```
#creating a range of numbers from -10 to 10 with step 1 using np.arange()
R = np.arange(-10,10,1)
c1,c2 = np.meshgrid(R,R)
#Getting the sum of the dot product of c1 and A and c2 and B to get the span of the linear combiination
spanRx = c1*vectA[0]+c2*vectB[0]
spanRy = c1*vectA[1]+c2*vectB[1]
```

Figure 8 Creating spans

The output for the first span is seen in figure 8 and the equation used is seen in figure 8. Figure 9 isn't linearly dependent, although some vectors overlap not all of it overlaps.

$$S_1 = \begin{cases} c_1 \cdot (2x + 3y) \\ c_2 \cdot (-4x + 3y) \end{cases}$$

Figure 8 Equation for span 1

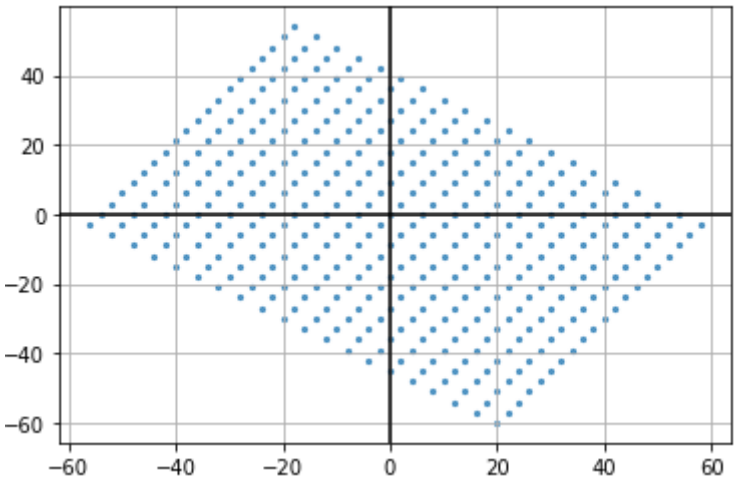

Figure 9 Output for task 2 span 1

The output for the second span is seen in figure 11 and the equation used is seen in figure 10. The same with figure 9, figure 11 isn't linearly dependent. The output became like that because the first element of vectA is 0 and the first element of vectB has a value and the second element of vectA has a value and the second element of vectB is 0. So when plotting for the values for spanRx and spanRy it would look like the same when plotting the values from the np.meshgrid() function.

$$S_2 = \begin{cases} c_1 \cdot (2y) \\ c_2 \cdot (2x) \end{cases}$$

Figure 10 Equation for span 2

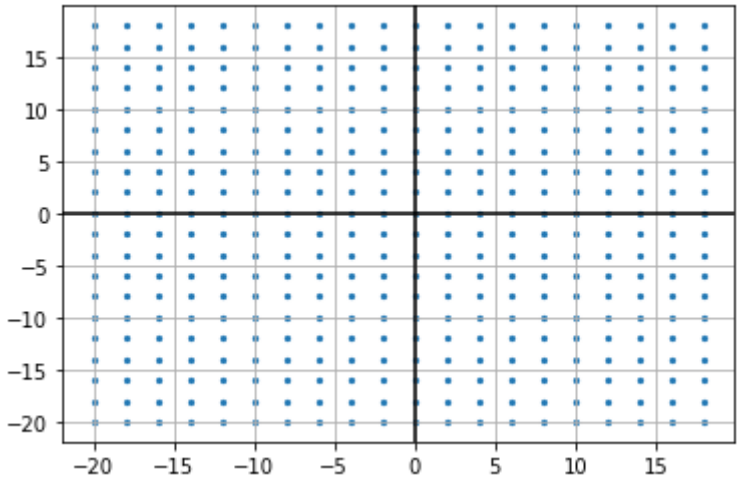

Figure 11 Output for task 2 span 2

The output for the third span is seen in figure 13 and the equation is seen in figure 12. Figure 13 shows a linearly dependent linear combination. The vectors in figure 13 are linearly dependent as it overlaps into a single line. This can be seen in the equation, the first equation is the same as the second equation when scaled to 2.

$$S_3 = \begin{cases} c_1 \cdot (2x + 2y) \\ c_2 \cdot (4x + 4y) \end{cases}$$

Figure 12 Equation for span 3

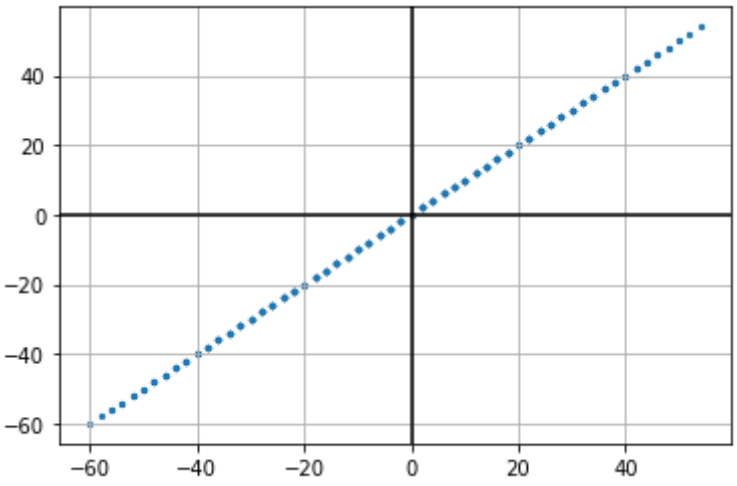

Figure 13 Output for task 2 span 3

Based on the result and observation, the dimension of R is equal to the R value. So when R = 3 and R = 4, their dimensions are 3 and 4. As seen in figures 14 and 15.

```
c1,c2,c3 = np.meshgrid(R,R,R)
print("Dimension of R when R = 3: ", c1.ndim)
```

```
Dimension of R when R = 3:  3
```

Figure 14 Dimension of R = 3

```
c1,c2,c3,c4 = np.meshgrid(R,R,R,R)
print("Dimension of R when R = 4: ", c1.ndim)
```

```
Dimension of R when R = 4:  4
```

Figure 15 Dimension of R = 4

The shape of the vector visualization of R = 3 is three dimensional. Meanwhile, the vector visualization of R = 4 looks also like a three-dimensional even though it is in four dimension as seen in figures 16 and 17. The limitation can be on the function used which is the plt.scatter().

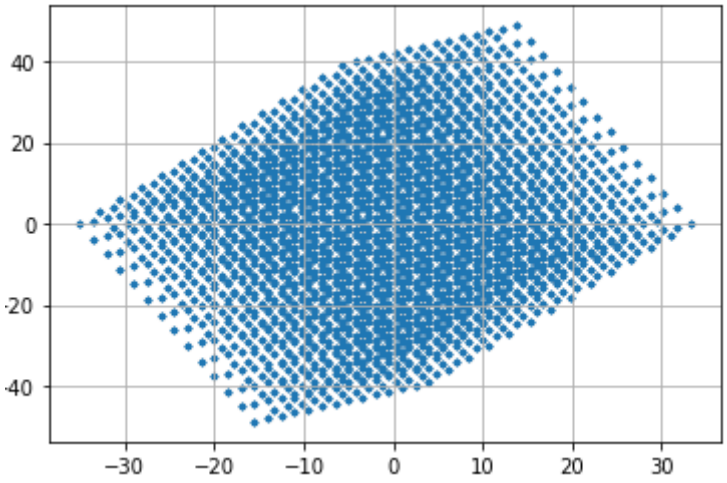

Figure 16 Vector visualization of R = 3

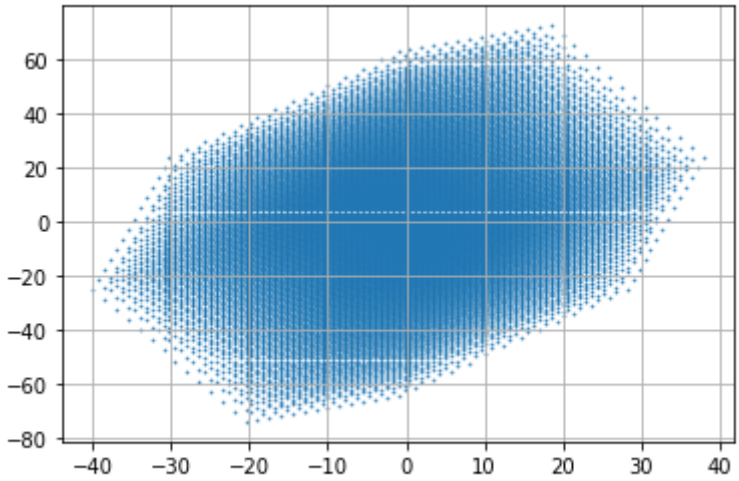

Figure 17 Vector visualization of R = 4

$$S_4 = \begin{cases} c_1 \cdot (2x - 5y) \\ c_2 \cdot (3x + 4y) \\ c_3 \cdot (2x + y) \end{cases} \quad S_5 = \begin{cases} c_1 \cdot (2x - 5y) \\ c_2 \cdot (3x + 4y) \\ c_3 \cdot (2x + y) \\ c_4 \cdot (x + 5y) \end{cases}$$

Figure 18 Equations used for R = 3 and R = 4

Unit vectors in linear combinations are the base values to be scaled by scalar values. Without unit vectors, there would be no linear combinations as they serve as the X and Y values to be plotted.

IV. Conclusion

The laboratory activity discussed linear combinations and spans of linear combinations. The activity also discussed using the functions in NumPy and Matplotlib to perform operations and visualize linear combinations.

This activity is a good introduction for beginners wanting to learn Data Science. It would be a good start for beginners to be familiar with the functions of NumPy and Matplotlib and scientific programming in general.

An example of a real-life application of linear combinations would be in economics. Using linear combinations, one can predict if a business would be getting profit in a given amount of time. It can also be used in budgeting to predict the expenses in a month's rent and meals [8].

Another real-life application of linear combinations would be in engineering. In any time-sensitive engineering problems, multiplying matrices can give approximate results of

complicated calculations. In computer science/engineering, it can be used on image processing and genetic analysis [9].

References

[1] NumPy, “numpy.arange,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.arange.html.

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

[3] NumPy, “numpy.meshgrid,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html.

[4] Matplotlib, “matplotlib.axes.Axes.axhline,” 2020. https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.axes.Axes.axhline.html.

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

[6] Matplotlib, “matplotlib.pyplot.grid,” 2020, [Online]. Available: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.grid.html.

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

[8] J. Smith, “How Are Linear Equations Used in Everyday Life?” https://sciencing.com/use-algebra-real-life-5714133.html.

[9] L. Hardesty, “Explained: Matrices.” https://news.mit.edu/2013/explained-matrices-1206#:~:text=The numbers in a matrix,can also represent mathematical equations.&text=In a range of applications,many more than two variables.

Appendix A

Github Repository Link:
https://github.com/Loreynszxc/Linear-Algebra-Lab-3/tree/main/Lab%203