



---

Linear Algebra

Laboratory Activity No. 3

---

# Linear Combination and Vector Spaces

---

*Submitted by:*

Sustento, Myke Alvin E.

*Instructor:*

Engr. Dylan Josh D. Lopez

October 25, 2000

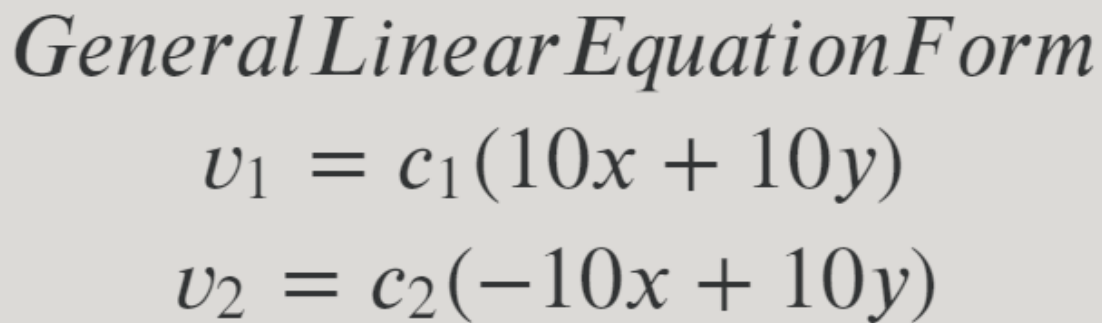
---

## I. Objectives

This laboratory activity aims to implement the principles and techniques of linear combinations in a 2-dimensional plane and span of a linear combination of vectors. This activity also aims to visualize the linear combinations and span of linear combinations using vector field operations used in Python.

## II. Methods

The practices of this laboratory activity consist of using the NumPy library and matplotlib library. The functions that was used using the Numpy library was the `np.array()`, `np.arange()`, and `np.meshgrid()` while the functions that was used using the matplotlib library was the `plt.scatter()`, `plt.axhline()`, `plt.axvline()`, `plt.grid()`, and `plt.show()`. This is to show the deliverables of the laboratory activity which is to visualize the linear combinations and span of linear combinations using vector field operations.



The image shows a light gray rectangular box containing the text "General Linear Equation Form" in a large, italicized serif font. Below this title, two linear equations are displayed in a smaller serif font. The first equation is  $v_1 = c_1(10x + 10y)$  and the second equation is  $v_2 = c_2(-10x + 10y)$ .

$$\textit{General Linear Equation Form}$$
$$v_1 = c_1(10x + 10y)$$
$$v_2 = c_2(-10x + 10y)$$

Figure 1 Linear Equation of the Linear Combination in Task 1

In the linear combination of the activity, there are two vectors given. The first vector has  $c_1(10x + 10y)$  while the second vector has  $c_2(-10x + 10y)$ . The  $c$  values would be shown in the code.

$$\begin{array}{c}
 \textit{Vector Form} \\
 V_1 = c_1 \cdot \begin{bmatrix} 10 \\ 10 \end{bmatrix}, V_2 = c_2 \cdot \begin{bmatrix} -10 \\ 10 \end{bmatrix}
 \end{array}$$

Figure 2 Vector Form of the Linear Combination in Task 1

Figure 2 displays the vector form of the linear equation given in figure 1 which is  $V_1 = c_1 \cdot \begin{bmatrix} 10 \\ 11 \end{bmatrix}$  and  $V_2 = c_2 \cdot \begin{bmatrix} -10 \\ 11 \end{bmatrix}$ .

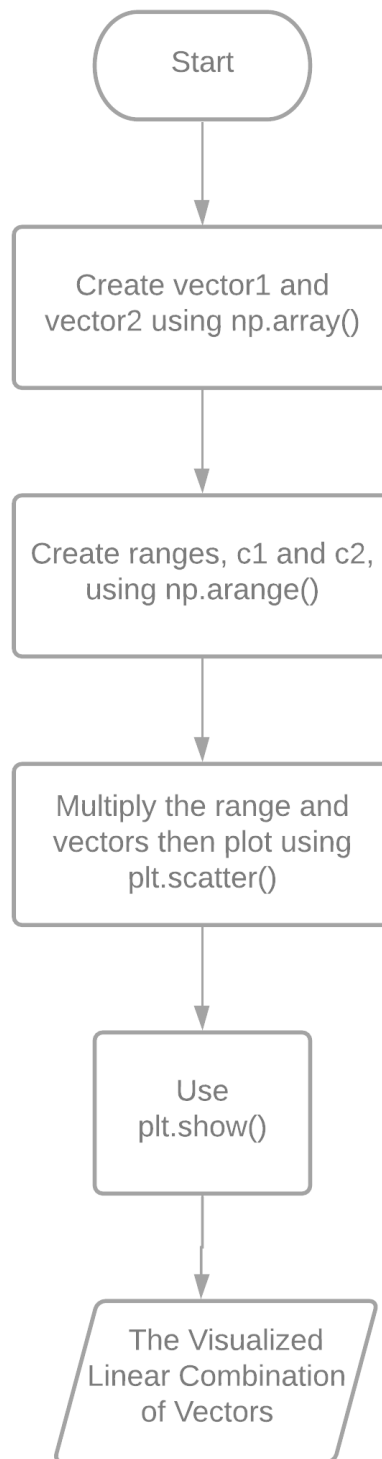


Figure 3 The Flowchart for Declaring and Display Linear Combination in Task 1

Figure 3 shows the flowchart for declaring and display linear combination to show the process that was used in the code in Task 1.

*General Linear Equation Form*

$$A = c_1(-1x + 1y)$$

$$B = c_2(1x + 1y)$$

Figure 4 Linear Equation of the 1st Span of Linear Combination in Task 2

*Vector Form*

$$S = \left\{ c_1 \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix}, c_2 \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

Figure 5 Vector Form of the 1st Span of Linear Combination in Task 2

*General Linear Equation Form*

$$A = c_1(2x + 5y)$$

$$B = c_2(5x + 2y)$$

Figure 6 Linear Equation of the 2nd Span of Linear Combination in Task 2

*Vector Form*

$$S = \left\{ c_1 \cdot \begin{bmatrix} 2 \\ 5 \end{bmatrix}, c_2 \cdot \begin{bmatrix} 5 \\ 2 \end{bmatrix} \right\}$$

Figure 7 Vector Form of the 2nd Span of Linear Combination in Task 2

### *General Linear Equation Form*

$$A = c_1(3x + 6y)$$

$$B = c_2(9x + 12y)$$

Figure 8 Linear Equation of the 3rd Span of Linear Combination in Task 2

### *Vector Form*

$$S = \left\{ c_1 \cdot \begin{bmatrix} 3 \\ 6 \end{bmatrix}, c_2 \cdot \begin{bmatrix} 9 \\ 12 \end{bmatrix} \right\}$$

Figure 9 Vector Form of the 3rd Span of Linear Combination in Task 2

In the span of a linear combination of the activity, there are 3 unique spans with different linear combinations. The 1<sup>st</sup> span has linear equations  $A = c_1(-1x + 1y)$  and  $B = c_2(1x + 1y)$ , the 2<sup>nd</sup> span has linear equations  $A = c_1(2x + 5y)$  and  $B = c_2(5x + 2y)$ , and the 3<sup>rd</sup> span has linear equations  $A = c_1(3x + 6y)$  and  $B = c_2(9x + 12y)$ . The 1<sup>st</sup> span has vector form  $S = \left\{ c_1 \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix}, c_2 \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$ , the 2<sup>nd</sup> span has vector form  $S = \left\{ c_1 \cdot \begin{bmatrix} 2 \\ 5 \end{bmatrix}, c_2 \cdot \begin{bmatrix} 5 \\ 2 \end{bmatrix} \right\}$ , and the 3<sup>rd</sup> span has vector form  $S = \left\{ c_1 \cdot \begin{bmatrix} 3 \\ 6 \end{bmatrix}, c_2 \cdot \begin{bmatrix} 9 \\ 12 \end{bmatrix} \right\}$ ,

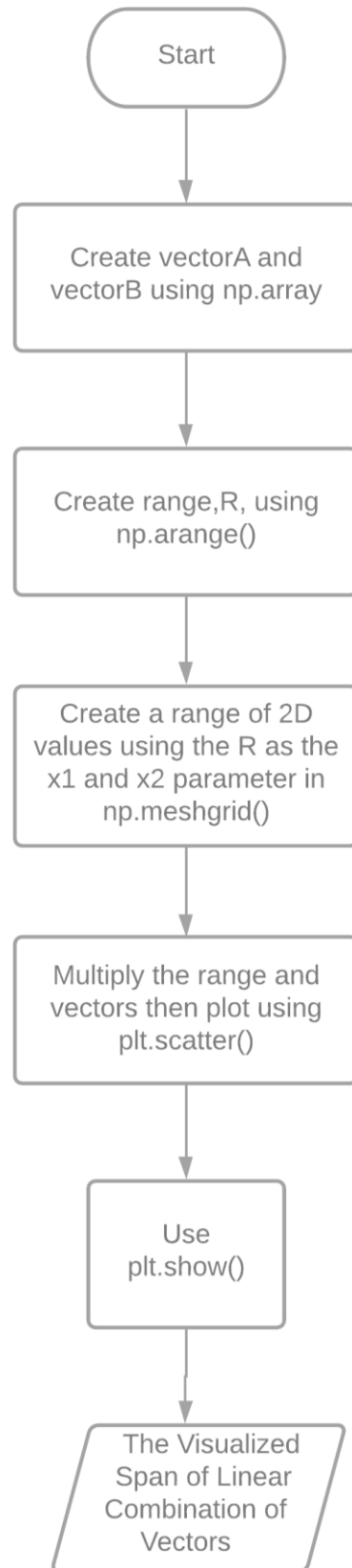


Figure 10 The Flowchart for Declaring and Display Linear Combination in Task 2

Figure 10 shows the flowchart for declaring and displaying a span of linear combination to show the process that was used in Task 2.

### III. Results

```
# unit vectors
vect1 = np.array([10,10])
vect2 = np.array([-10,10])

# range
c1 = np.arange(-10,11,1)
c2 = np.arange(0,11,1)

# multiplying the range to the unit vectors
# linear combination
plt.scatter(c1*vect1[0],c1*vect1[1], color = 'r')
plt.scatter(c2*vect2[0],c2*vect2[1], color = 'g')

# lines across the origin with the color as black
plt.axhline(y=0, color='black')
plt.axvline(x=0, color='black')

#configures grid lines
plt.grid()

#displays all figures
plt.show()
```

Figure 11 The Code in Task 1

Figure 11 is the implementation of the code that was used in task 1 to visualize the linear combination. The function `np.array()` was used to create an array [1] that would represent the values of the x and y in the linear equation form while `np.arange()` was used to return evenly spaced values within a given interval [2] to represent the values of  $c_1$  and  $c_2$ . After creating the array of vectors and the range of c values, `plt.scatter()` was used to create a scatter plot of x vs y [3] to multiply the range to the unit vectors. In order to create a line across the origin, `plt.axhline()` and `plt.axvline()` to create a horizontal line and vertical line across the axis [4] [5].



In order to configure the grid lines, `plt.grid()` was used [6]. Finally, `plt.show()` function was used to display all figures[7].

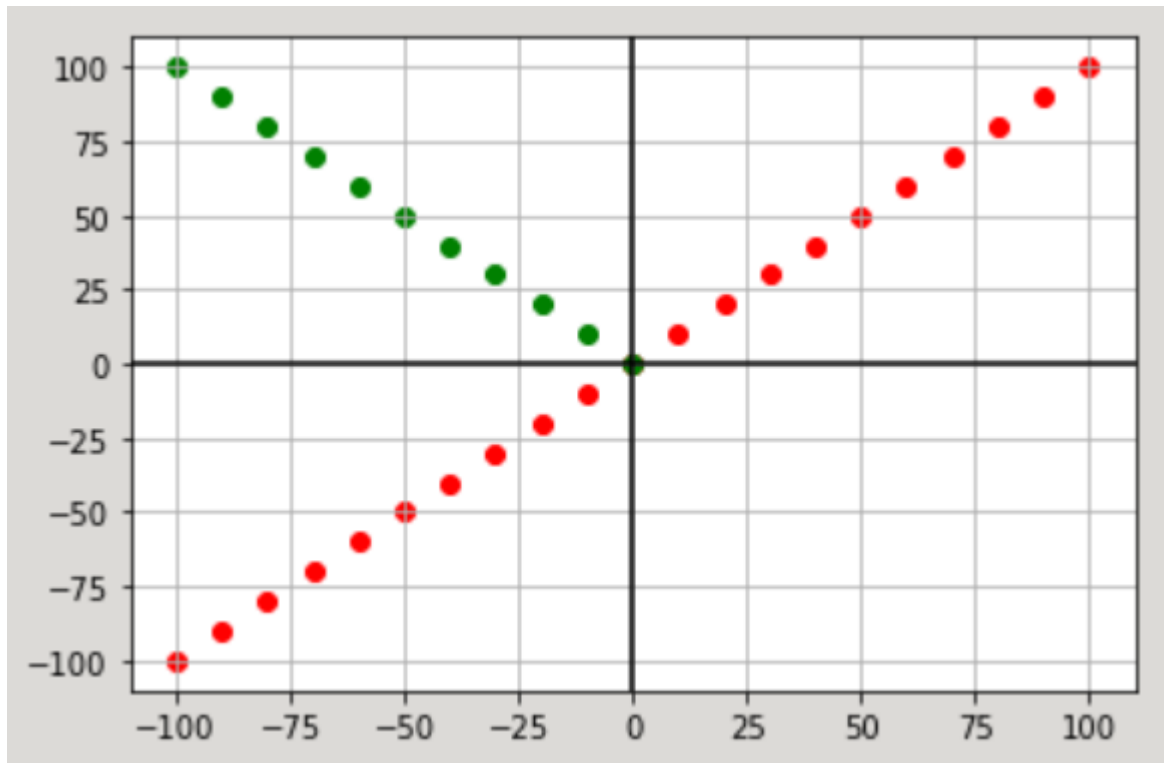


Figure 12 The Output of the Code in Task 1

Figure 12 shows the plot created by the code in figure 11 using the values that were used in the linear equation form or the vector form. The output that was created is a one-dimensional line since the rank of the vectors was 1. The two lines differ in direction and range since both of them have distinct vector values and distinct range.

```

# unit vectors
vectA = np.array([-1,1])
vectB = np.array([1,1])

# range
R = np.arange(-10,11)

# creates a range of 2d values of R and R
# coordinate matrices from coordinate vectors
c1, c2 = np.meshgrid(R,R)

#span
spanRx = c1*vectA[0] + c2*vectB[0]
spanRy = c1*vectA[1] + c2*vectB[1]
plt.scatter(spanRx,spanRy, s=20, alpha=0.75, color = 'r')

# Lines across the origin with the color as black
plt.axhline(y=0, color='black')
plt.axvline(x=0, color='black')

#configures grid lines
plt.grid()

#displays all figures
plt.show()

```

Figure 13 The Code of the 1<sup>st</sup> span in Task 2

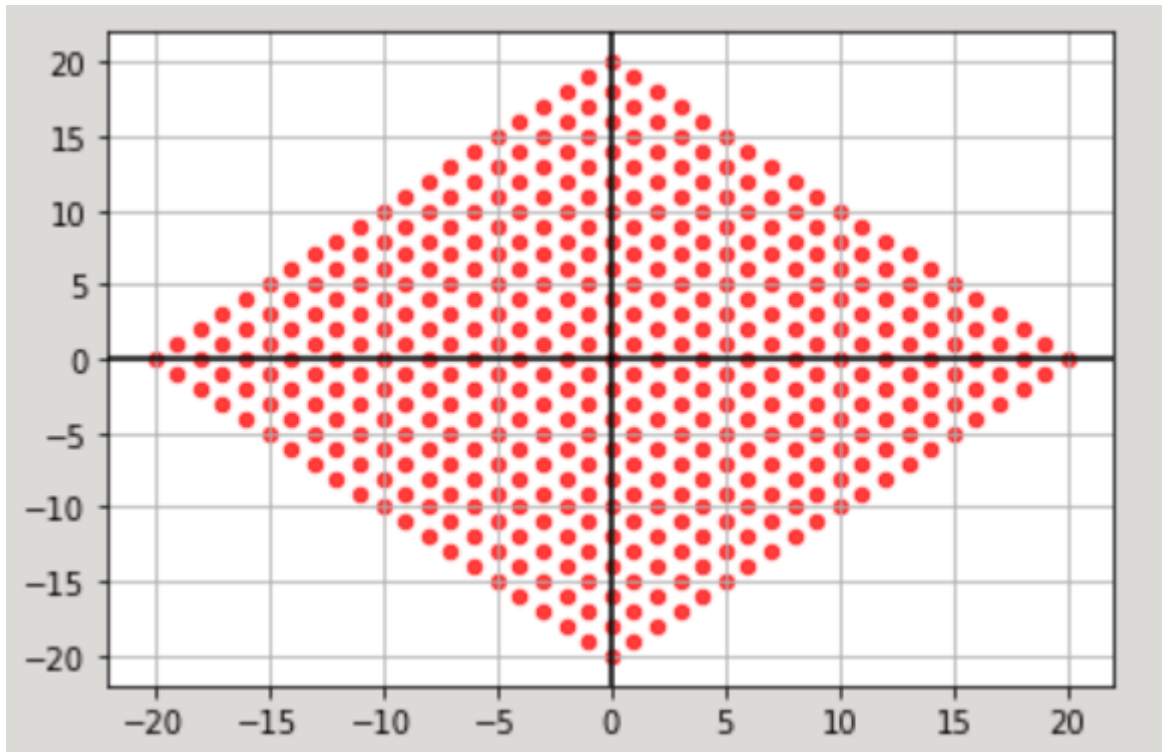


Figure 14 The Output of the Code of the 1<sup>st</sup> span in Task 2

```

# unit vectors
vectA = np.array([2,5])
vectB = np.array([5,2])

# range
R = np.arange(-10,11)

# creates a range of 2d values of R and R
# coordinate matrices from coordinate vectors
c1, c2 = np.meshgrid(R,R)

#span
spanRx = c1*vectA[0] + c2*vectB[0]
spanRy = c1*vectA[1] + c2*vectB[1]
plt.scatter(spanRx,spanRy, s=20, alpha=0.75, color = 'g')

# lines across the origin with the color as black
plt.axhline(y=0, color='black')
plt.axvline(x=0, color='black')

#configures grid lines
plt.grid()

#displays all figures
plt.show()

```

Figure 15 The Code of the 2<sup>nd</sup> span in Task 2

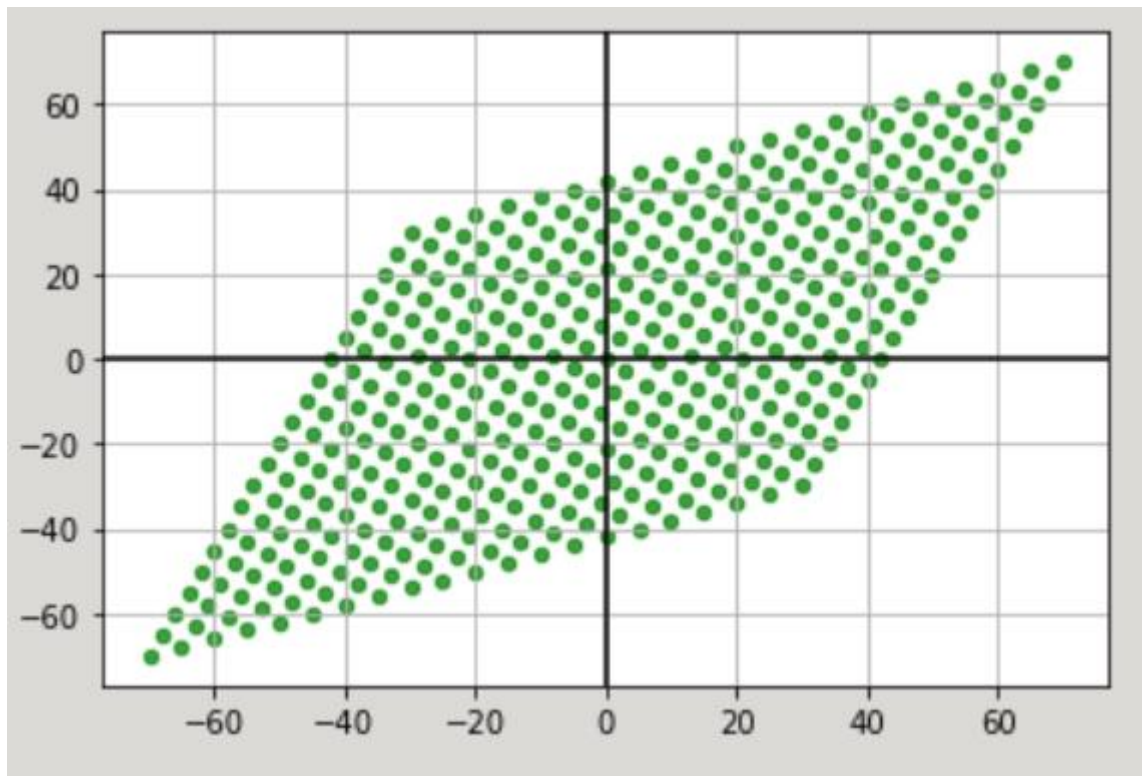


Figure 16 The Output of the Code of the 2<sup>nd</sup> span in Task 2

```

# unit vectors
vectA = np.array([3,6])
vectB = np.array([9,12])

# range
R = np.arange(-10,11,2)

# creates a range of 2d values of R and R
# coordinate matrices from coordinate vectors
c1, c2 = np.meshgrid(R,R)

#span
spanRx = c1*vectA[0] + c2*vectB[0]
spanRy = c1*vectA[1] + c2*vectB[1]
plt.scatter(spanRx,spanRy, s=20, alpha=0.75, color = 'b')

# Lines across the origin with the color as black
plt.axhline(y=0, color='black')
plt.axvline(x=0, color='black')

#configures grid lines
plt.grid()

#displays all figures
plt.show()

```

Figure 17 The Code of the 3<sup>rd</sup> span in Task 2

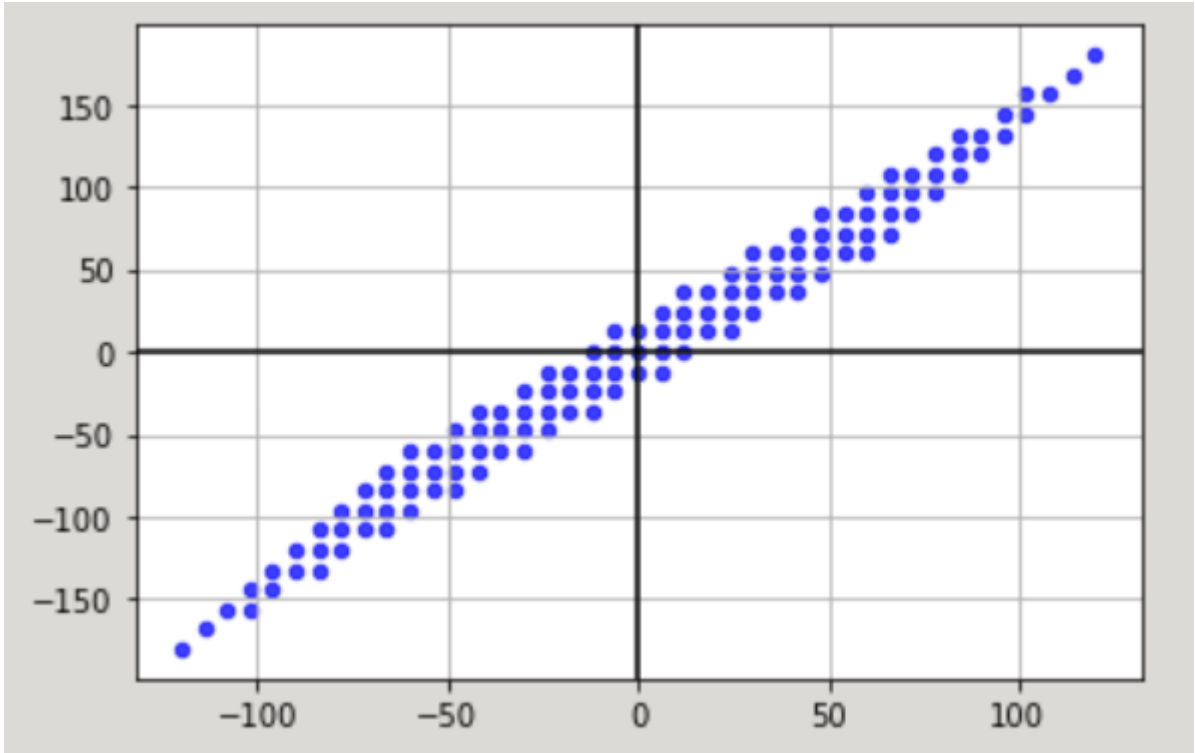


Figure 18 The Output of the Code of the 3<sup>rd</sup> span in Task 2

The figures in figure 13, figure 15, and figure 17 shows the code that was used to output the plots in figure 14, figure 16, and figure 18. They all have the same functionality but with different values in the vectors in the array. Some of the functions used in task 1 was also used in task 2 such as `np.array()`, `np.arange()`, `plt.scatter()`, `plt.axhline()`, `plt.axvline()`, `plt.grid()`, and `plt.show()` and they were used the same as task . The difference that made the outputs in task 2 a 2d plane is due to the `np.meshgrid()` that used the `R` in `np.arange()` as its parameter. The function `np.meshgrid()` is used to return coordinate matrices from coordinate vectors [8].

The dimensions plot of linear combinations changes according to its rank as can be seen in task 1 and task 2. The rank in task 1 was  $\mathbb{R} = 1$  so the plot resulted in a line, a 1-dimensional plot, while the rank in task 2 was  $\mathbb{R} = 2$  so the plot resulted in a plane, a 2-dimensional plot. If the rank was  $\mathbb{R} = 3$ , the plot would result in a 3-dimensional plot and it would be a 4-dimensional plot if the  $\mathbb{R} = 4$  [9]. The rank indicates what is the dimension of the plot.

The role of the unit vectors is to determine the direction of the linear combination and the span. They are the values that would be scaled by the scalar values given by a range. This is the reason why the output in figure 12 was a green line going to quadrant 2 and a red line going from quadrant 3 to quadrant 1 since the unit vector of the green line was  $V1 = c_1 \cdot \begin{bmatrix} 10 \\ 11 \end{bmatrix}$

while the red line had the unit vector  $V2 = c_2 \cdot \begin{bmatrix} -10 \\ 11 \end{bmatrix}$  where the c represents the scalar values that determine the range of the line in the code in figure 11.

## IV. Conclusion

In conclusion, the laboratory activity not only proves that linear combinations can be represented by a 1-dimensional line and 2-dimensional plane, but it also proves that the functions in the libraries, NumPy and matplotlib, can visualize and perform linear combination and spans of a linear combination. It is also known in the laboratory activity that the dimension of the plot changes depending on the rank and that unit vectors can determine the direction of the plot while the scalar values determine its range.

The concept of linear combination can be used in physics to visualize different linear equations or different dimensional planes for an easier conception of the topic. It would also help to determine the rate of changes in business or economic graphs. Linear combinations would be most helpful by visualizing problems through a plot to find correlations.



## References

- [1]"numpy.array — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: <https://numpy.org/doc/stable/reference/generated/numpy.array.html>. [Accessed: 25- Oct- 2020].
- [2]"numpy.arange — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: <https://numpy.org/doc/stable/reference/generated/numpy.arange.html>. [Accessed: 25- Oct- 2020].
- [3]"matplotlib.pyplot.scatter — Matplotlib 3.3.2 documentation", Matplotlib.org, 2020. [Online]. Available: [https://matplotlib.org/3.3.2/api/\\_as\\_gen/matplotlib.pyplot.scatter.html](https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.scatter.html). [Accessed: 25- Oct- 2020].
- [4]"matplotlib.axes.Axes.axhline — Matplotlib 3.3.2 documentation", Matplotlib.org, 2020. [Online]. Available: [https://matplotlib.org/3.3.2/api/\\_as\\_gen/matplotlib.axes.Axes.axhline.html](https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.axes.Axes.axhline.html). [Accessed: 25- Oct- 2020].
- [5]"matplotlib.pyplot.axvline — Matplotlib 3.3.2 documentation", Matplotlib.org, 2020. [Online]. Available: [https://matplotlib.org/3.3.2/api/\\_as\\_gen/matplotlib.pyplot.axvline.html](https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.axvline.html). [Accessed: 25- Oct- 2020].
- [6]"matplotlib.pyplot.grid — Matplotlib 3.1.2 documentation", Matplotlib.org, 2020. [Online]. Available: [https://matplotlib.org/3.1.1/api/\\_as\\_gen/matplotlib.pyplot.grid.html](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.grid.html). [Accessed: 25- Oct- 2020].
- [7]"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](https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.show.html). [Accessed: 25- Oct- 2020].
- [8]"numpy.meshgrid — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: <https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html>. [Accessed: 25- Oct- 2020].
- [9]"Vector and spaces | Linear algebra", 2020. [Online]. Available: <https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/vectors/v/linear-algebra-parametric-representations-of-lines?modal=1>. [Accessed: 25- Oct- 2020].

# Appendix

Github repository - [https://github.com/Sus102/LA\\_Lab/tree/master/LA\\_LAB\\_3](https://github.com/Sus102/LA_Lab/tree/master/LA_LAB_3)