Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No. 10

Linear Transformation

Submitted by:
Sustento, Myke Alvin E.

Instructor:
Engr. Dylan Josh D. Lopez

December 31, 2020

I. Objectives

This laboratory activity aims to implement the principles and techniques of linear transformation such as repositioning/translation, shears, scaling, and rotation. This laboratory activity aims to familiarize the reader with the role of matrix operations, visualize matrix operations, and justify the precedence of matrix operations through the use of the programming language, Python.

II. Methods

The deliverables of this laboratory activity are to implement the principles and techniques of linear transformation such as repositioning/translation, shears, scaling, and rotation. Since this laboratory activity uses Python and utilizes the modules of Numpy and Matplotlib, the methodologies was done using the functions created by the modules such as np.arange() to return an evenly spaced values for the spans [1], np.meshgrid() to return coordinate matrices from coordinate vectors [2], plt.scatter() to create a scatter plot [3], plt.grid() to configure the grid lines [4], np.eye() to return a 2D array with ones on the diagonal and zeros elsewhere [5] or return an identity matrix, np.deg2rad() to convert angles from degrees to radians [6], np.array() to create an array [7], np.cos() to cosine element-wise [8], and np.sin() to trigonometric sine element-wise [9].

III. Results

```
# plots a scatter
def plot_scatter(x,t_mat=np.eye(2)):
    x_prime = x @ t_mat
    R = np.arange(-10,11,2) # range
    c1, c2 = np.meshgrid(R,R) ## creates a range of 2d values of R and R
                              # coordinate matrices from coordinate vectors
    spanRx = c1*x_prime[0][0] + c2*x_prime[1][0]
    spanRy = c1*x_prime[0][1] + c2*x_prime[1][1]
    plt.scatter(spanRx,spanRy) #creates a scatterplot using the spanRx and SpanRy to define the arrow directions
    plt.grid() # creates a grid in the figure
    plt.show() # shows the figure
```

Figure 1 plot_scatter() Method

Figure 1 shows the plot_scatter() method which can be used to show the figure of the scatter plot. It is almost the same as the activity in laboratory 3 which is about linear combinations and vector spaces. The method makes use of the x_prime, the range, and the spans.

```python
# rotation
def rot_matrix(theta):
    theta = np.deg2rad(theta)
    rot_mat = np.array([
        [np.cos(theta), -np.sin(theta)],
        [np.sin(theta), np.cos(theta)]
    ])
    return rot_mat
```

Figure 2 rot_matrix() Method

Figure 2 shows the rot_matrix() method which rotates a given matrix. The theta will be provided once the method is called.

```python
A = np.eye(2)
plot_scatter(A)

rotate = rot_matrix(45)
plot_scatter(A@rotate)
```

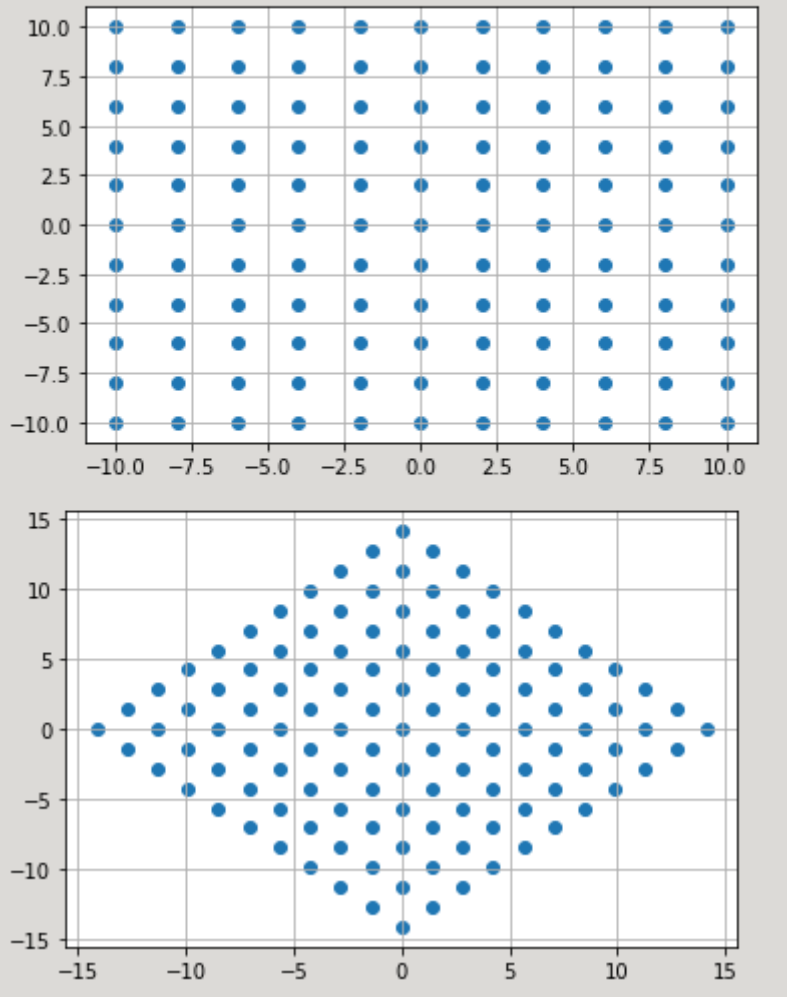

Figure 3 Result of the rot_matrix() Method

Figure 3 shows the calling of the rot_matrix() method. The cell starts by creating an identity matrix using the np.eye() function and is then represented as the A matrix. In order to see the plot scatter of the A matrix, the plot_scatter() method was used. To rotate the given matrix, the rot_matrix() method was used with a theta of 45 degrees and this would consequently turn the 1st scatterplot 45 degrees which can be seen in the 2nd scatterplot.

```
# plots a quiv
def plot_quiv(x,t_mat=np.eye(2)):
    x_prime = x @ t_mat
    R = np.arange(-10,11,2) # range
    c1, c2 = np.meshgrid(R,R) ## creates a range of 2d values of R and R
                              # coordinate matrices from coordinate vectors
    spanRx = c1*x_prime[0][0] + c2*x_prime[1][0]
    spanRy = c1*x_prime[0][1] + c2*x_prime[1][1]
    plt.quiver(spanRx,spanRy) #creates a quiver using the spanRx and SpanRy to define the arrow directions
    plt.grid() # creates a grid in the figure
    plt.show() # shows the figure
```

Figure 4 The plot_quiv() Method

Figure 4 shows the plot_quiv method which can be used to show the repositioning/translation, shear, and scale.

```
# repositioning / translation
A = np.eye(2)
t_mat = np.array([
    [-1,0],
    [0,-1]
])
plot_quiv(A)
plot_quiv(A, t_mat)
```

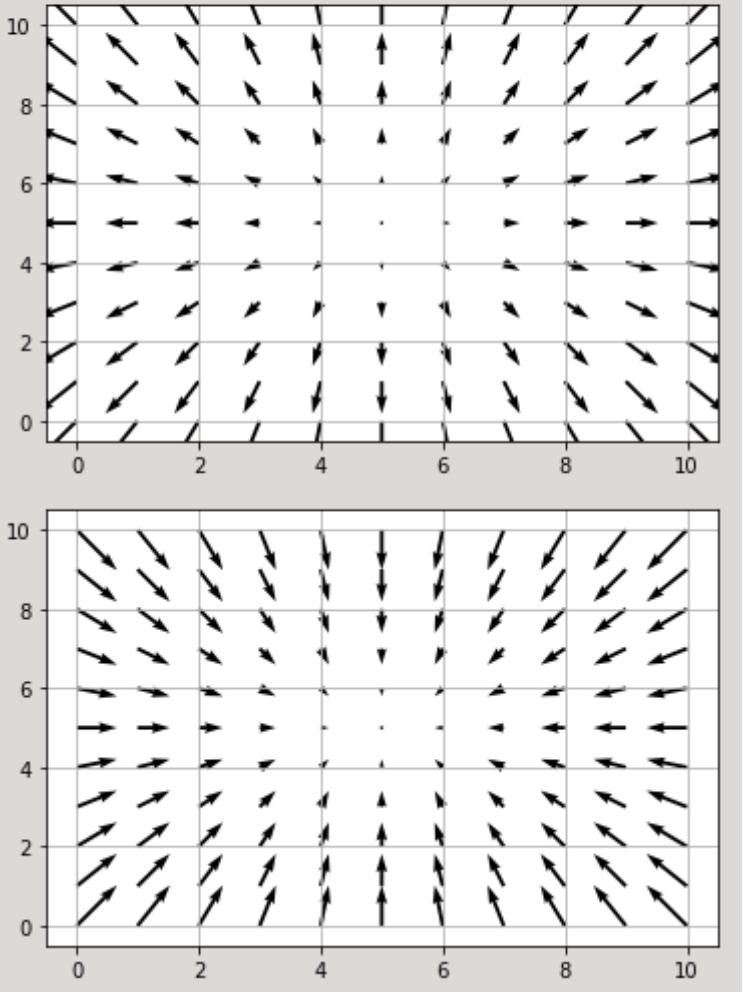

Figure 5 Repositioning / Translation

Figure 5 shows the code and figure of a repositioning linear transformation. It can be seen that the 2nd plot mirrors the 1st plot because the t_mat values are t_mat = np.array([[-1,0],[0,-1]]) which is an identity matrix.

```
# shear
A = np.eye(2)
shear = np.array([
    [1,0],
    [1,1]
])
plot_quiv(A)
plot_quiv(A, shear)
```

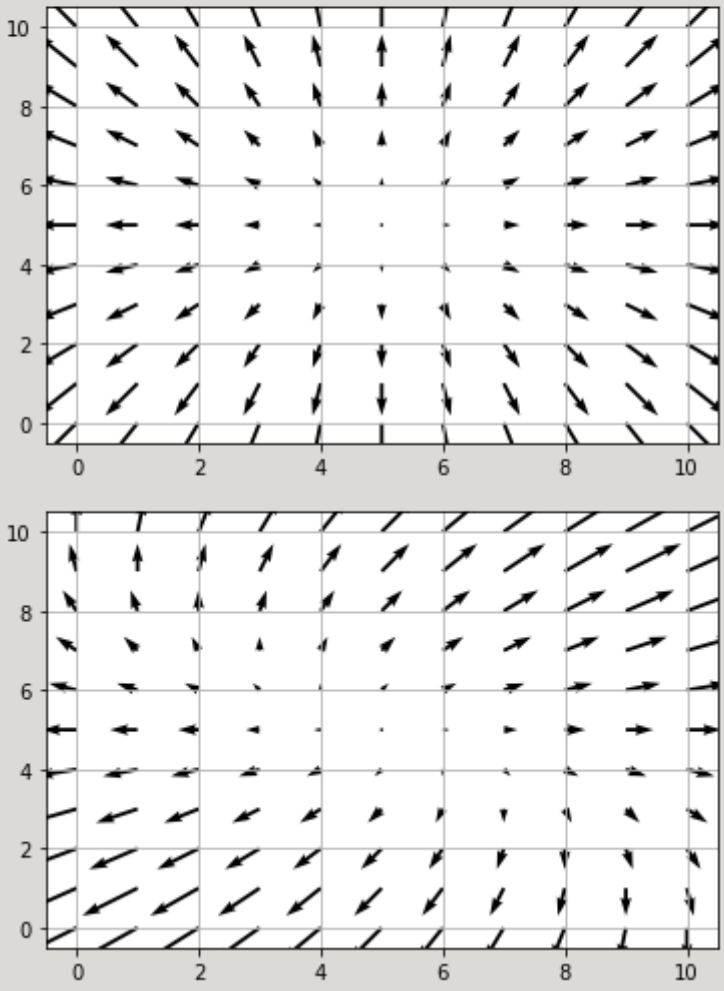

Figure 6 Shear

Figure 6 shows the code and figures of an example of a shear linear transformation. The 1st plot's tips fall on the 2nd plot which is what a shear matrix looks like.

```
# scale
A = np.eye(2)
scale = np.array([
    [3,0],
    [0,3]
])

plot_quiv(A)
plot_quiv(scale, A)
```

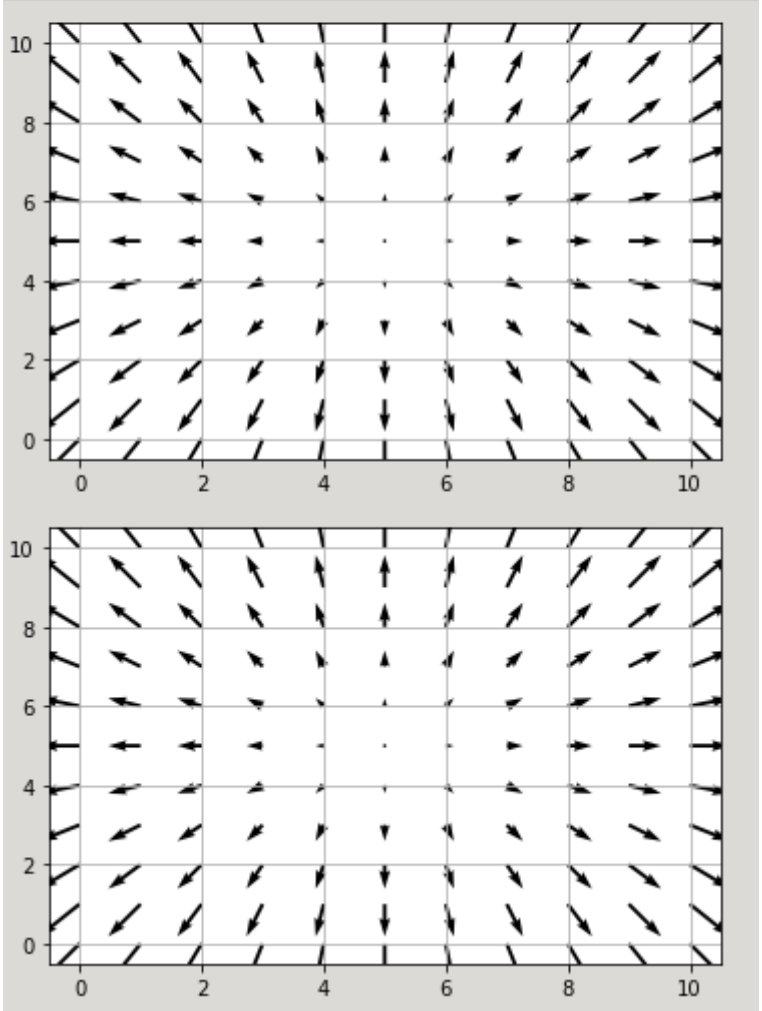

Figure 7 Scale

Figure 7 shows the code and figures of a scale linear transformation. It can be seen that the 1st plot and 2nd plot is the same. However, there were some changes in the code that should have scaled the length of the quivers. The reason why the changes can't be seen in the figure is due to the limitation of perspective in a 2D plane. If the figure was shown with a 3D figure, the changes in the length should be seen.

IV. Conclusion

In conclusion, the transformation of multi-dimensional vectors or simply matrices is possible using matrix operations. Examples of linear transformations are scaling, shearing, rotation, and repositioning which can be done through Python by utilizing the modules of NumPy and matplotlib. As shown in the results, the two prime requirements for linear geometric translations are followed. The two prime requirements are the vectors remain linear upon applying the linear function and the origin of the vector does not change.

There are a lot of complications in the process of transmissions such as various unknown signals, poor signals' stability, and high error rates. Linear transformation can be used to isolate current frequency using a device with a circuit structure that has a strong anti-interference ability [10].

References

[1]"numpy.arange — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.arange.html. [Accessed: 31- Dec- 2020].

[2]"numpy.meshgrid — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html. [Accessed: 31- Dec- 2020].

[3]"matplotlib.pyplot.scatter — Matplotlib 3.3.3 documentation", Matplotlib.org, 2020. [Online]. Available: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.scatter.html. [Accessed: 31- Dec- 2020].

[4]"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. [Accessed: 31- Dec- 2020].

[5]"numpy.eye — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.eye.html. [Accessed: 31- Dec- 2020].

[6]"numpy.deg2rad — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.deg2rad.html. [Accessed: 31- Dec- 2020].

[7]"numpy.array — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.array.html. [Accessed: 31- Dec- 2020].

[8]"numpy.cos — NumPy v1.19 Manual", Numpy.org, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.cos.html. [Accessed: 31- Dec- 2020].

[9]"numpy.sin — NumPy v1.19 Manual", *Numpy.org*, 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.sin.html. [Accessed: 31- Dec- 2020].

[10]Y. Wu, Y. Zhang, G. Xu, X. Wang and W. Gou, " Design and Implementation of Linear Transformation Device for Current Frequency Isolation", Atlantis Press, 2020. [Online]. Available: https://www.atlantis-press.com/proceedings/eame-15/22340. [Accessed: 31- Dec- 2020].

Appendix

Github Link - https://github.com/Sus102/LA_Lab/tree/master/LA_LAB_10