MatrixTransformations

August 11, 2018

Hopefully the notebook on changing basis has helped you to develop a conceptual understanding of how matrices can encode linear transformations. The purpose of this notebook is to provide a few more examples of specific matrix transformations in order to further cement your understanding.

To start lets just imagine a matrix that doesn't transform anything, otherwise known as the identity matrix:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

As you can see this matrix simply says that our first basis vector (first column) remains where it is and our second basis vector (seond column) also remains as where it is. If we apply the identity matrix to a specific point in our co-ordinate system then we simply get the same point back:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

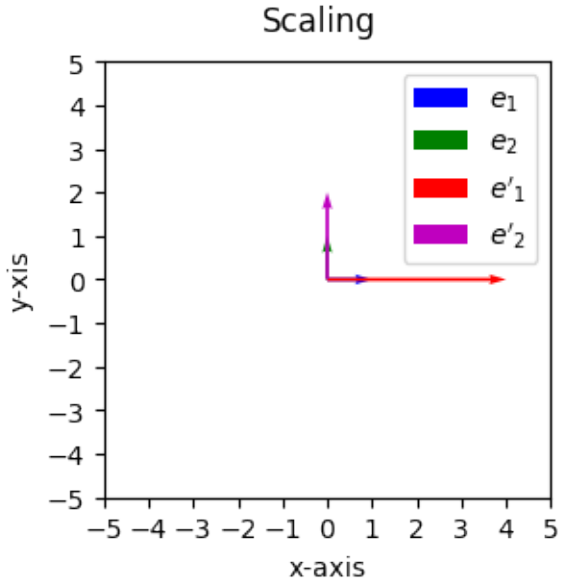
The identity matrix gives us a nice starting to point to build our conceptual understanding of what different matrix transformations might look like. Can you predict what would happen to our vector space if we applied the following transformation matrix A?

$$A = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$$

A tells us that our first basis vector will be scaled by a factor of 4 and our second basis vector will be scaled by a factor of 2. This can be seen below, where e_1 and e_2 are our original unit basis vectors and e_1' and e_2' are our new basis vectors:

```
In [2]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        mpl.rcParams['figure.dpi']= 100

        basis_x = [1,0]
        basis_y = [0,1]
```

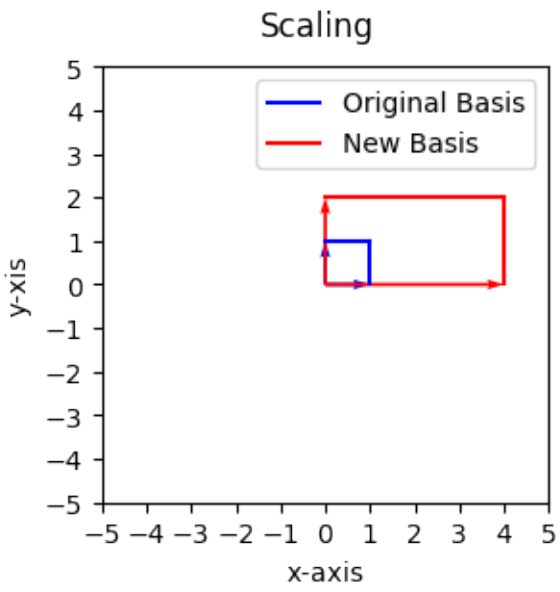
```
A = np.array([[4,0],[0,2]])

fig = plt.figure(figsize=(3,3))
plt.quiver(0, 0, basis_x[0], basis_x[1], color='b', units='xy',
           angles='xy', scale_units='xy', scale=1., label='$e_1$')
plt.quiver(0, 0, basis_y[0], basis_y[1], color='g', units='xy',
           angles='xy', scale_units='xy', scale=1., label='$e_2$')
plt.quiver(0, 0, A[0,0], A[1,0], color='r', units='xy', angles='xy',
           scale_units='xy', scale=1., label='$e\'_1$')
plt.quiver(0, 0, A[0,1], A[1,1], color='m', units='xy', angles='xy',
           scale_units='xy', scale=1., label='$e\'_2$')

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Scaling')
plt.show()
```



This effectively stretches space in the direction of our first basis vector by a factor of 4 and in the direction of our second basis vector by a factor of 2. To see this we can plot a unit square and look at how it changes when we apply the matrix A:

```
In [3]: top_left = np.array([0,1])
        top_right = np.array([1,1])
```

```
bottom_right = np.array([1,0])

new_top_left = np.matmul(A, top_left)
new_top_right = np.matmul(A, top_right)
new_bottom_right = np.matmul(A, bottom_right)

fig = plt.figure(figsize=(3,3))

plt.plot([basis_x[0], basis_x[0]], [basis_y[0], basis_y[1]],
         'b-', label='Original Basis')
plt.plot([basis_x[0], basis_x[1]], [basis_y[1], basis_y[1]],
         'b-')
plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)
plt.quiver(0, 0, basis_y[0], basis_y[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)

plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]],
         'r-', label='New Basis')
plt.plot([new_top_left[0], new_top_right[0]],
         [new_top_left[1], new_top_right[1]], 'r-')
plt.quiver(0, 0, A[0,0], A[1,0], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)
plt.quiver(0, 0, A[0,1], A[1,1], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Scaling')
plt.show()
```



Of course matrix transformations are not just limited to scaling, there are many other classic linear transformations. We will now work through the following linear transformations just to deepen your conceptual understanding of what any given matrix might be doing to space:

- Reflection
- Rotation
- Shear

Importantly, all matrix transformations are linear because they ensure that gridlines remain parallel and evenly spaced. You can see why this is the case because any new basis vector that we define is just a scaled sum of the existing basis vectors. For example if we want to move our basis vector $\begin{bmatrix}1\\0\end{bmatrix}$ to $\begin{bmatrix}3\\4\end{bmatrix}$ we can simply write this as:

$$3 * \begin{bmatrix}1\\0\end{bmatrix} + 4 * \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}3\\4\end{bmatrix}$$

i.e. our new basis vector is just a linear combination of our existing basis vectors. To perform a simple reflection in the y-axis we use the following transformation matrix:

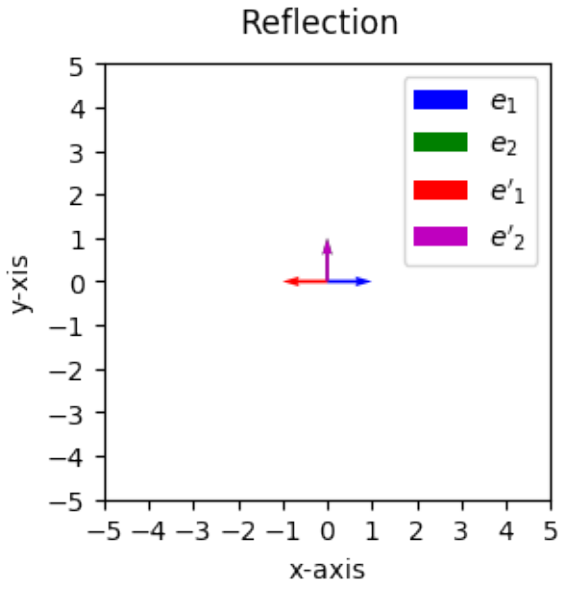
$$B = \begin{bmatrix}-1 & 0\\0 & 1\end{bmatrix}$$

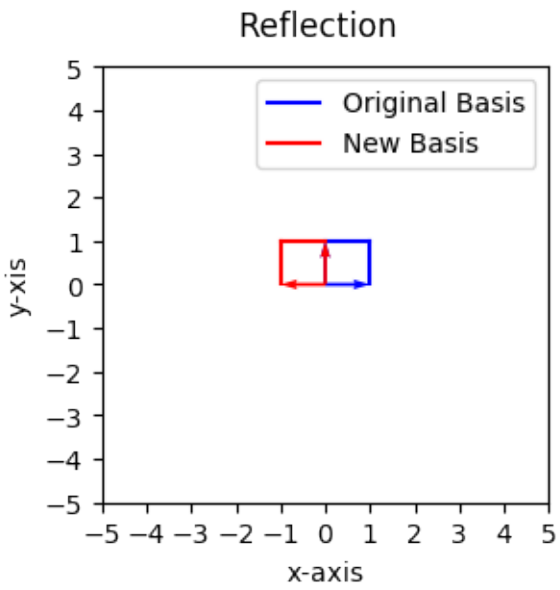
We can see this by simply plotting the new basis vectors and again looking at what happens to our unit square:

```
In [4]: B = np.array([[-1, 0],[0, 1]])
```

```
fig = plt.figure(figsize=(3,3))
plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1., label='$e_1$')
plt.quiver(0, 0, basis_y[0], basis_y[1], color='g',
           units='xy', angles='xy', scale_units='xy',
           scale=1., label='$e_2$')
plt.quiver(0, 0, B[0,0], B[1,0], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.,
           label='$e\'_1$')
plt.quiver(0, 0, B[0,1], B[1,1], color='m', units='xy',
           angles='xy', scale_units='xy', scale=1.,
           label='$e\'_2$')

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Reflection')
plt.show()

new_top_left = np.matmul(B, top_left)
new_top_right = np.matmul(B, top_right)
new_bottom_right = np.matmul(B, bottom_right)

fig = plt.figure(figsize=(3,3))

plt.plot([basis_x[0], basis_x[0]],
         [basis_y[0], basis_y[1]],
         'b-', label='Original Basis')
plt.plot([basis_x[0], basis_x[1]],
         [basis_y[1], basis_y[1]], 'b-')
plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)
plt.quiver(0, 0, basis_y[0], basis_y[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)

plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]],
         'r-', label='New Basis')
plt.plot([new_top_left[0], new_top_right[0]],
         [new_top_left[1], new_top_right[1]],
         'r-')
```

```
plt.quiver(0, 0, B[0,0], B[1,0], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)
plt.quiver(0, 0, B[0,1], B[1,1], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Reflection')
plt.show()
```





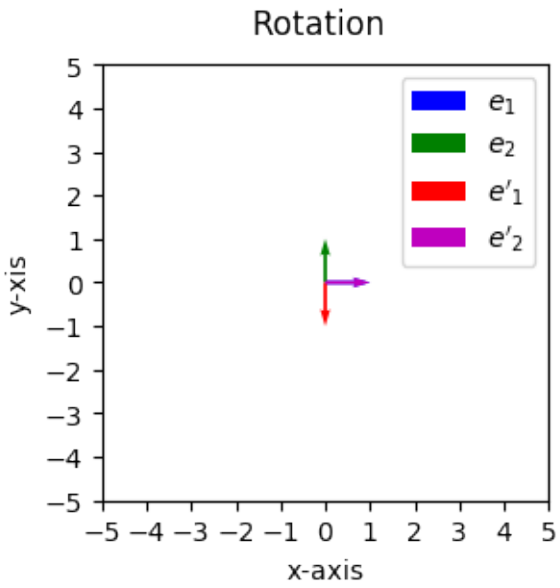
Hopefully these nature of these transformations is starting to become more and more obvious. Can you predict what a matrix might look like that performs a reflection in the line $y = x$?

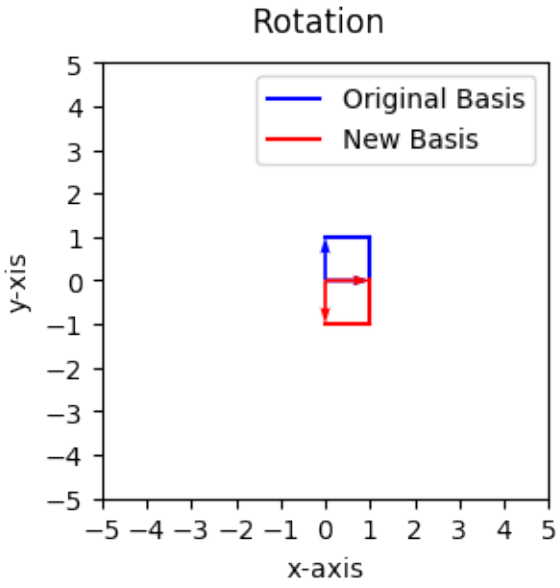
Moving on to the next classic type of transformation, below is a matrix that simply rotates space by 90 degrees clockwise:

$$C = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

```
In [5]: C = np.array([[0, 1],[-1, 0]])

        fig = plt.figure(figsize=(3,3))
        plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
                   units='xy', angles='xy', scale_units='xy',
                   scale=1., label='$e_1$')
        plt.quiver(0, 0, basis_y[0], basis_y[1], color='g',
                   units='xy', angles='xy', scale_units='xy',
                   scale=1., label='$e_2$')
        plt.quiver(0, 0, C[0,0], C[1,0], color='r', units='xy',
                   angles='xy', scale_units='xy', scale=1.,
                   label='$e\'_1$')
        plt.quiver(0, 0, C[0,1], C[1,1], color='m', units='xy',
                   angles='xy', scale_units='xy', scale=1.,
                   label='$e\'_2$')

        plt.legend()
        plt.xlim([-5, 5])
```

```
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Rotation')
plt.show()

new_top_left = np.matmul(C, top_left)
new_top_right = np.matmul(C, top_right)
new_bottom_right = np.matmul(C, bottom_right)

fig = plt.figure(figsize=(3,3))

plt.plot([basis_x[0], basis_x[0]],
         [basis_y[0], basis_y[1]], 'b-',
         label='Original Basis')
plt.plot([basis_x[0], basis_x[1]],
         [basis_y[1], basis_y[1]], 'b-')
plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)
plt.quiver(0, 0, basis_y[0], basis_y[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)

plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]],
         'r-', label='New Basis')
plt.plot([new_top_left[0], new_top_right[0]],
         [new_top_left[1], new_top_right[1]], 'r-')
plt.quiver(0, 0, C[0,0], C[1,0], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)
plt.quiver(0, 0, C[0,1], C[1,1], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Rotation')
plt.show()
```





A more general form for performing rotations is as follows:

$$D = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

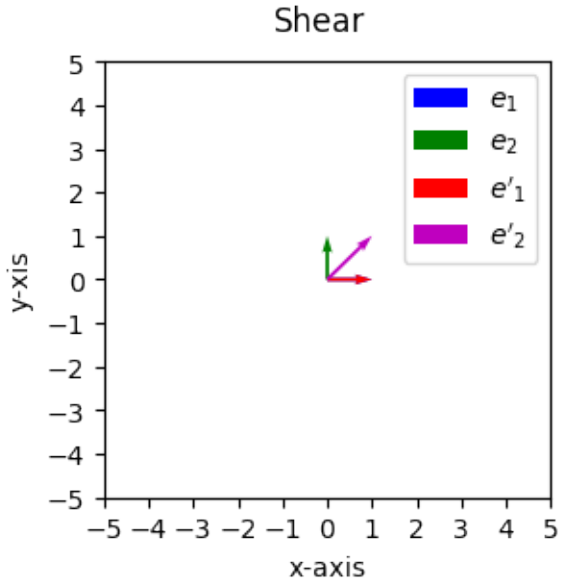
where θ is the angle you wish to rotate space by in a clockwise direction ($-\theta$ for anti-clockwise rotations).

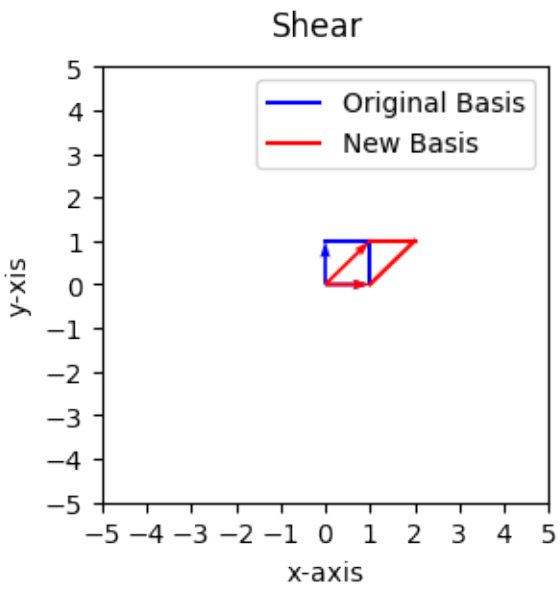
The final linear transformation we will cover in this notebook is shear. To perform a simple horizontal shear we can use the following transformation matrix:

$$E = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

In [6]:
```
E = np.array([[1, 1],[0, 1]])

fig = plt.figure(figsize=(3,3))
plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1., label='$e_1$')
plt.quiver(0, 0, basis_y[0], basis_y[1], color='g',
           units='xy', angles='xy', scale_units='xy',
           scale=1., label='$e_2$')
plt.quiver(0, 0, E[0,0], E[1,0], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.,
           label='$e\'_1$')
plt.quiver(0, 0, E[0,1], E[1,1], color='m', units='xy',
           angles='xy', scale_units='xy', scale=1.,
           label='$e\'_2$')

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Shear')
plt.show()

new_top_left = np.matmul(E, top_left)
new_top_right = np.matmul(E, top_right)
new_bottom_right = np.matmul(E, bottom_right)

fig = plt.figure(figsize=(3,3))

plt.plot([basis_x[0], basis_x[0]],
         [basis_y[0], basis_y[1]], 'b-',
         label='Original Basis')
plt.plot([basis_x[0], basis_x[1]],
         [basis_y[1], basis_y[1]], 'b-')
plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)
plt.quiver(0, 0, basis_y[0], basis_y[1], color='b',
           units='xy', angles='xy', scale_units='xy',
```

```
            scale=1.)

plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]],
         'r-', label='New Basis')
plt.plot([new_top_left[0], new_top_right[0]],
         [new_top_left[1], new_top_right[1]], 'r-')
plt.quiver(0, 0, E[0,0], E[1,0], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)
plt.quiver(0, 0, E[0,1], E[1,1], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Shear')
plt.show()
```



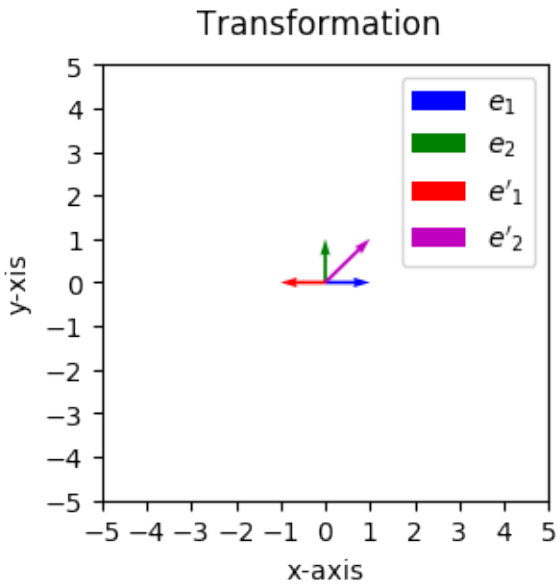


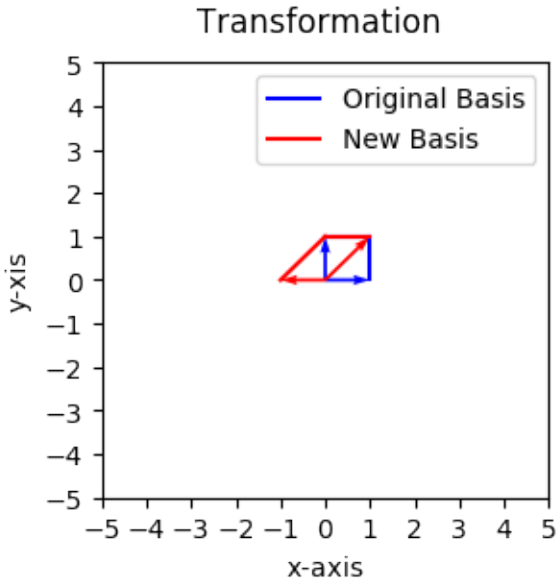
Of course we can combine these different types of linear transformation into a single transformation matrix. As a final example try to predict what the plots will look like for the following transformation matrix:

$$F = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}$$

```
In [7]: F = np.array([[-1, 1],[0, 1]])

        fig = plt.figure(figsize=(3,3))
        plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
                   units='xy', angles='xy', scale_units='xy',
                   scale=1., label='$e_1$')
        plt.quiver(0, 0, basis_y[0], basis_y[1], color='g',
                   units='xy', angles='xy', scale_units='xy',
                   scale=1., label='$e_2$')
        plt.quiver(0, 0, F[0,0], F[1,0], color='r', units='xy',
                   angles='xy', scale_units='xy', scale=1.,
                   label='$e\'_1$')
        plt.quiver(0, 0, F[0,1], F[1,1], color='m', units='xy',
                   angles='xy', scale_units='xy', scale=1.,
                   label='$e\'_2$')

        plt.legend()
        plt.xlim([-5, 5])
        plt.xticks(np.arange(-5, 6))
        plt.ylim([-5, 5])
        plt.yticks(np.arange(-5, 6))
```

```
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Transformation')
plt.show()

new_top_left = np.matmul(F, top_left)
new_top_right = np.matmul(F, top_right)
new_bottom_right = np.matmul(F, bottom_right)

fig = plt.figure(figsize=(3,3))

plt.plot([basis_x[0], basis_x[0]],
         [basis_y[0], basis_y[1]], 'b-',
         label='Original Basis')
plt.plot([basis_x[0], basis_x[1]],
         [basis_y[1], basis_y[1]], 'b-')
plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)
plt.quiver(0, 0, basis_y[0], basis_y[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)

plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]],
         'r-', label='New Basis')
plt.plot([new_top_left[0], new_top_right[0]],
         [new_top_left[1], new_top_right[1]], 'r-')
plt.quiver(0, 0, F[0,0], F[1,0], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)
plt.quiver(0, 0, F[0,1], F[1,1], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Transformation')
plt.show()
```





To help understand what is happening when we apply the matrix F, we can break it down into two successive transformations using matrix multiplication:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} = F$$

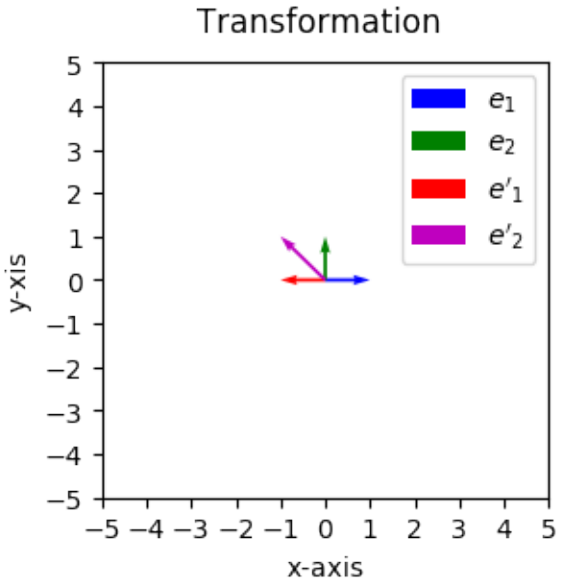
So F is actually performing a reflection in the y-axis followed by a horizontal shear. Note that the order is important here! If we do a reflection in the y-axis followed by a horizontal shear we get a different transformation matrix:

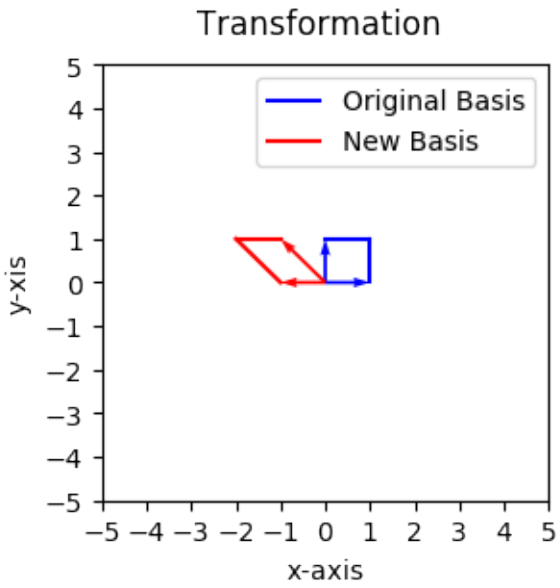
$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 0 & 1 \end{bmatrix} = G$$

This transformation matrix G looks like the following:

```
In [8]: G = np.array([[-1, -1],[0, 1]])

        fig = plt.figure(figsize=(3,3))
        plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
                   units='xy', angles='xy', scale_units='xy',
                   scale=1., label='$e_1$')
        plt.quiver(0, 0, basis_y[0], basis_y[1], color='g',
                   units='xy', angles='xy', scale_units='xy',
                   scale=1., label='$e_2$')
        plt.quiver(0, 0, G[0,0], G[1,0], color='r', units='xy',
                   angles='xy', scale_units='xy', scale=1.,
                   label='$e\'_1$')
        plt.quiver(0, 0, G[0,1], G[1,1], color='m', units='xy',
                   angles='xy', scale_units='xy', scale=1.,
                   label='$e\'_2$')

        plt.legend()
        plt.xlim([-5, 5])
        plt.xticks(np.arange(-5, 6))
        plt.ylim([-5, 5])
        plt.yticks(np.arange(-5, 6))
        plt.xlabel('x-axis')
        plt.ylabel('y-xis')
        plt.suptitle('Transformation')
        plt.show()

        new_top_left = np.matmul(G, top_left)
        new_top_right = np.matmul(G, top_right)
        new_bottom_right = np.matmul(G, bottom_right)

        fig = plt.figure(figsize=(3,3))

        plt.plot([basis_x[0], basis_x[0]],
                 [basis_y[0], basis_y[1]], 'b-',
                 label='Original Basis')
        plt.plot([basis_x[0], basis_x[1]],
                 [basis_y[1], basis_y[1]], 'b-')
        plt.quiver(0, 0, basis_x[0], basis_x[1], color='b',
                   units='xy', angles='xy', scale_units='xy',
```

```
            scale=1.)
plt.quiver(0, 0, basis_y[0], basis_y[1], color='b',
           units='xy', angles='xy', scale_units='xy',
           scale=1.)

plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]],
         'r-', label='New Basis')
plt.plot([new_top_left[0], new_top_right[0]],
         [new_top_left[1], new_top_right[1]], 'r-')
plt.quiver(0, 0, G[0,0], G[1,0], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)
plt.quiver(0, 0, G[0,1], G[1,1], color='r', units='xy',
           angles='xy', scale_units='xy', scale=1.)

plt.legend()
plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.xlabel('x-axis')
plt.ylabel('y-xis')
plt.suptitle('Transformation')
plt.show()
```





I hope these examples have helped to deepen your understanding of matrix transformations. In the next notebook we will look at inverses and determinants and how they relate to the types of transformations we have described here.