EigenvectorsAndEigenvalues

August 11, 2018

The last few notebooks have given us all the knowledge we need to understand eigenvectors and eigenvalues at a conceptual level. Eigen related topics often seem difficult to people first learning about linear algebra. However a good understanding of how matrices correspond to linear transformations will reveal a very intuitive explanation for eigenvectors and eigenvalues.

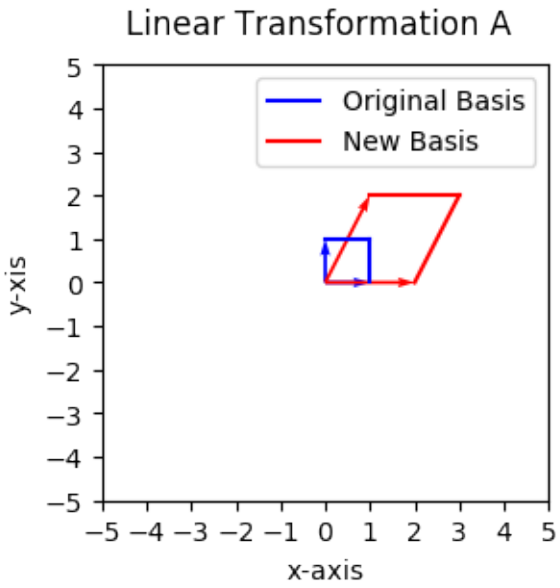
If we were to translate the German word 'eigen' into English we would see that it roughly means 'characteristic'. This translation makes sense because eigenvectors and eigenvalues are a useful way of describing the characteristics of a given linear transformation. Lets look at an example matrix so that we can see how the eigenvectors and eigenvalues can describe the characteristics of a linear transformation. The following matrix A describes a horizontal sheer:

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}$$

The code below demonstrates the linear transformation expressed by A graphically:

In [29]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.dpi']= 100

standard_basis = np.array([[1, 0],[0, 1]])
A = np.array([[2, 1],[0, 2]])

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([standard_basis[0, 0], standard_basis[0, 0]],
         [standard_basis[0, 1], standard_basis[1, 1]], 'b-',
         label='Original Basis')
plt.plot([standard_basis[0, 0], standard_basis[1, 0]],
         [standard_basis[1, 1], standard_basis[1, 1]], 'b-')
plt.quiver(0, 0, standard_basis[0, 0], standard_basis[1, 0],
           color='b', units='xy', angles='xy', scale_units='xy',
           scale=1.)
plt.quiver(0, 0, standard_basis[0, 1], standard_basis[1, 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('Linear Transformation A')
plt.show()
```



Notice how our first basis vector $\begin{bmatrix}1\\0\end{bmatrix}$ continues to point in the same direction but is simply scaled by a factor of 2. In linear algebra terms we would say that this basis vector remains in the same span because it can be expressed as just a scaled version of itself. This is exactly how we define an eigenvector! $\begin{bmatrix}1\\0\end{bmatrix}$ is an eigenvector of A with a corresponding eigenvalue of 2. To be concrete, the mathematical defintion of an eigenvector and eigenvalue is as follows:

$$Ax = \lambda x$$

Where A is a matrix expressing a linear transformation, x is a vector in space and λ is a scalar value. x is said to be an eigenvector of A and λ is said to be its corresponding eigenvalue. We can easily see how the basis vector $\begin{bmatrix}1\\0\end{bmatrix}$ satisfies this equation and is therefore defined as an eigenvector:

$$\begin{aligned} A\begin{bmatrix}1\\0\end{bmatrix} &= \begin{bmatrix}2 & 1\\0 & 2\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} \\ &= \begin{bmatrix}2\\0\end{bmatrix} \\ &= 2 * \begin{bmatrix}1\\0\end{bmatrix} \\ &= \lambda\begin{bmatrix}1\\0\end{bmatrix} \end{aligned}$$

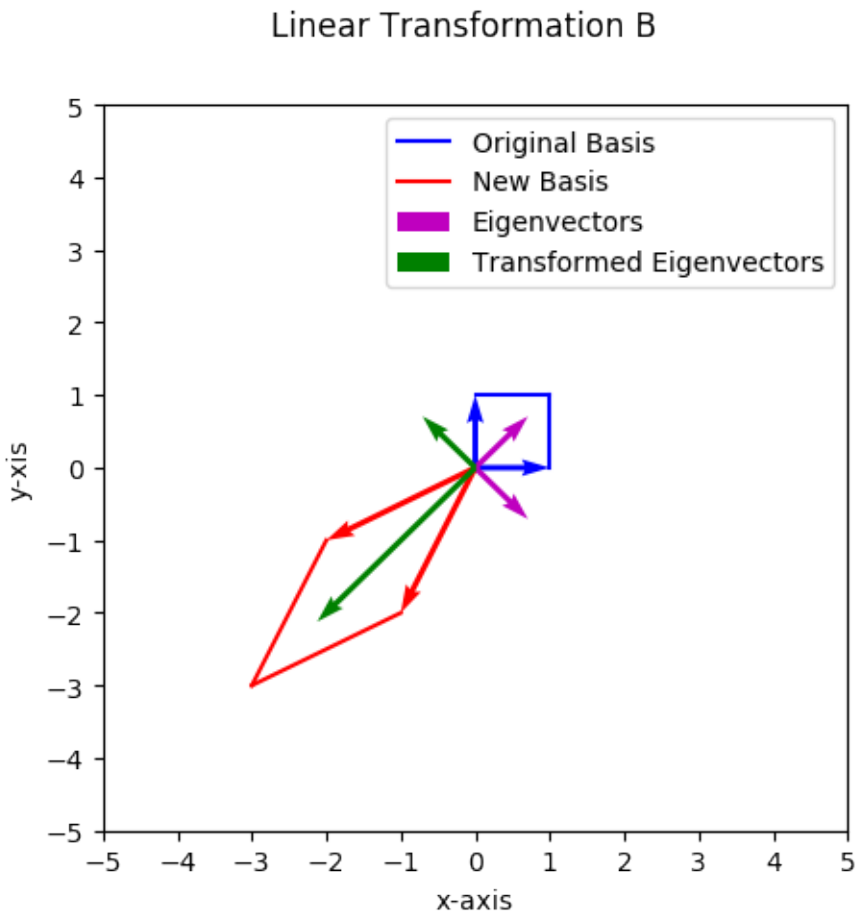
Of course any vector that is a scaled version of $\begin{bmatrix}1\\0\end{bmatrix}$ will also be an eigenvector of A and will have an eigenvalue of 2. Lets look at one more matrix just to solidify our conceptual understanding of eigenvectors and eigenvalues. The matrix B applies a 180 degree rotation followed by a sheer:

$$\begin{bmatrix}-2 & -1\\-1 & -2\end{bmatrix}$$

Can you predict what the eigenvectors and their correpsonding eigenvalues might be? The code below uses numpy's eig() function to find and plot the eigenvectors of B and shows how they change when we apply the matrix B.

```
In [30]: B = np.array([[-2, -1],[-1, -2]])

         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=(5,5))

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

         # Get and plot the eigenvectors
         w, v = np.linalg.eig(B)

         plt.quiver(0, 0, v[0,0], v[1,0], color='m', units='xy',
                    angles='xy', scale_units='xy', scale=1.,
                    label='Eigenvectors')
```

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

plt.quiver(0, 0, np.matmul(B, v[:,0])[0],
           np.matmul(B, v[:,0])[1],
           color='g', units='xy', angles='xy',
           scale_units='xy', scale=1.,
           label='Transformed Eigenvectors')
plt.quiver(0, 0, np.matmul(B, v[:,1])[0],
           np.matmul(B, v[:,1])[1],
           color='g', 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('Linear Transformation B')
plt.show()

print('Eigenvalues: ' + str(w))
```




```
Eigenvalues: [-1. -3.]
```

In this example the eigenvalues for both eigenvectors are negative. This means that while the eigenvectors still lie on the same span after the linear transformation they face in the opposite direction and are scaled by a factor of 1 and 3 repsectively.

Of course we don't just have to restrict ourselves to 2 dimensions. The principles behind eigenvectors and eigenvalues still apply regardless of the number of dimensions. For example take the following linear transformation expressed by the matrix C:

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

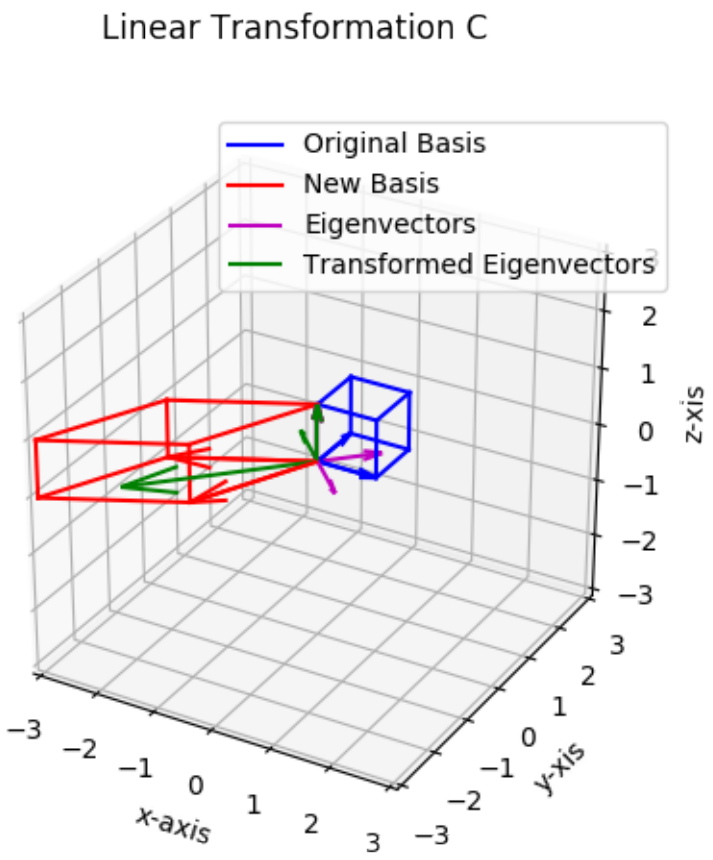
This linear transformation is the same as the previous one but with an additional dimension. Notice how the basis vector for this additional dimension does not change according to C. It

therefore defines the axis of rotation and will be an eigenvector with a corresponding eigenvalue of 1. This is demonstrated in the code below:

```
In [31]: standard_basis = np.array([[1, 0, 0],[0, 1, 0],[0, 0, 1]])
         C = np.array([[-2,-1,0],[-1,-2,0],[0,0,1]])

         point1 = np.array([0, 0, 0])
         point2 = np.array([1, 0, 0])
         point3 = np.array([1, 1, 0])
         point4 = np.array([0, 1, 0])
         point5 = np.array([0, 0, 1])
         point6 = np.array([1, 0, 1])
         point7 = np.array([1, 1, 1])
         point8 = np.array([0, 1, 1])

         from mpl_toolkits.mplot3d import Axes3D
         fig = plt.figure(figsize=(5,5))
         ax = plt.axes(projection='3d')

         ax.plot([point1[0], point2[0]], [point1[1], point2[1]],
                 [point1[2], point2[2]], 'b-',
                 label='Original Basis')
         ax.plot([point2[0], point3[0]], [point2[1], point3[1]],
                 [point2[2], point3[2]], 'b-')
         ax.plot([point3[0], point4[0]], [point3[1], point4[1]],
                 [point3[2], point4[2]], 'b-')
         ax.plot([point4[0], point1[0]], [point4[1], point1[1]],
                 [point4[2], point1[2]], 'b-')
         ax.plot([point1[0], point5[0]], [point1[1], point5[1]],
                 [point1[2], point5[2]], 'b-')
         ax.plot([point2[0], point6[0]], [point2[1], point6[1]],
                 [point2[2], point6[2]], 'b-')
         ax.plot([point3[0], point7[0]], [point3[1], point7[1]],
                 [point3[2], point7[2]], 'b-')
         ax.plot([point4[0], point8[0]], [point4[1], point8[1]],
                 [point4[2], point8[2]], 'b-')
         ax.plot([point5[0], point6[0]], [point5[1], point6[1]],
                 [point5[2], point6[2]], 'b-')
         ax.plot([point6[0], point7[0]], [point6[1], point7[1]],
                 [point6[2], point7[2]], 'b-')
         ax.plot([point7[0], point8[0]], [point7[1], point8[1]],
                 [point7[2], point8[2]], 'b-')
         ax.plot([point8[0], point5[0]], [point8[1], point5[1]],
                 [point8[2], point5[2]], 'b-')

         ax.quiver(0, 0, 0, standard_basis[0,0],
                   standard_basis[1, 0],
                   standard_basis[2, 0], color='b')
```

```
ax.quiver(0, 0, 0, standard_basis[0,1],
          standard_basis[1, 1],
          standard_basis[2, 1], color='b')
ax.quiver(0, 0, 0, standard_basis[0,2],
          standard_basis[1, 2],
          standard_basis[2, 2], color='b')

point1 = np.matmul(C, point1)
point2 = np.matmul(C, point2)
point3 = np.matmul(C, point3)
point4 = np.matmul(C, point4)
point5 = np.matmul(C, point5)
point6 = np.matmul(C, point6)
point7 = np.matmul(C, point7)
point8 = np.matmul(C, point8)

ax.plot([point1[0], point2[0]],
        [point1[1], point2[1]],
        [point1[2], point2[2]],
        'r-', label='New Basis')
ax.plot([point2[0], point3[0]],
        [point2[1], point3[1]],
        [point2[2], point3[2]], 'r-')
ax.plot([point3[0], point4[0]],
        [point3[1], point4[1]],
        [point3[2], point4[2]], 'r-')
ax.plot([point4[0], point1[0]],
        [point4[1], point1[1]],
        [point4[2], point1[2]], 'r-')
ax.plot([point1[0], point5[0]],
        [point1[1], point5[1]],
        [point1[2], point5[2]], 'r-')
ax.plot([point2[0], point6[0]],
        [point2[1], point6[1]],
        [point2[2], point6[2]], 'r-')
ax.plot([point3[0], point7[0]],
        [point3[1], point7[1]],
        [point3[2], point7[2]], 'r-')
ax.plot([point4[0], point8[0]],
        [point4[1], point8[1]],
        [point4[2], point8[2]], 'r-')
ax.plot([point5[0], point6[0]],
        [point5[1], point6[1]],
        [point5[2], point6[2]], 'r-')
ax.plot([point6[0], point7[0]],
        [point6[1], point7[1]],
        [point6[2], point7[2]], 'r-')
ax.plot([point7[0], point8[0]],
```

```
        [point7[1], point8[1]],
        [point7[2], point8[2]], 'r-')
ax.plot([point8[0], point5[0]],
        [point8[1], point5[1]],
        [point8[2], point5[2]], 'r-')

ax.quiver(0, 0, 0, C[0,0], C[1, 0],
          C[2, 0], color='r')
ax.quiver(0, 0, 0, C[0,1], C[1, 1],
          C[2, 1], color='r')
ax.quiver(0, 0, 0, C[0,2], C[1, 2],
          C[2, 2], color='r')

# Get and plot the eigenvectors
w, v = np.linalg.eig(C)

plt.quiver(0, 0, 0, v[0,0], v[1,0], v[2,0], color='m',
           label='Eigenvectors')
plt.quiver(0, 0, 0, v[0,1], v[1,1], v[2,1], color='m')
plt.quiver(0, 0, 0, v[0,2], v[1,2], v[2,2], color='m')

plt.quiver(0, 0, 0, np.matmul(C, v[:,0])[0],
           np.matmul(C, v[:,0])[1], np.matmul(C, v[:,0])[2],
           color='g', label='Transformed Eigenvectors')
plt.quiver(0, 0, 0, np.matmul(C, v[:,1])[0],
           np.matmul(C, v[:,1])[1], np.matmul(C, v[:,1])[2],
           color='g')
plt.quiver(0, 0, 0, np.matmul(C, v[:,2])[0],
           np.matmul(C, v[:,2])[1], np.matmul(C, v[:,2])[2],
           color='g')

plt.legend()
ax.set_xlim([-3, 3])
ax.set_xticks(np.arange(-3, 4))
ax.set_ylim([-3, 3])
ax.set_yticks(np.arange(-3, 4))
ax.set_zlim([-3, 3])
ax.set_zticks(np.arange(-3, 4))
ax.set_xlabel('x-axis')
ax.set_ylabel('y-xis')
ax.set_zlabel('z-xis')
plt.suptitle('Linear Transformation C')
plt.show()

print('Eigenvalues: ' + str(w))
```




```
Eigenvalues: [-1. -3.  1.]
```

Hopefully you can now see how the linear transformation A can be described as a set of eigenvectors and corresponding eigenvalues. The eigenvectors and eigenvalues are characteristic of the linear transformation because they are the only vectors that do not change span and are simply scaled by a single scalar value. This is a powerful result in linear algebra.

The eigendecomposition of a matrix takes advantage of the fact that a linear transformation can be described by its eigenvectors and eigenvalues. Remember from the change of basis notebook that a matrix provides us with the instructions we need to express a point in another co-ordinate system in our co-ordinate system. Similarly the inverse of a matrix provides us with the instructions we need to express a point in our co-ordinate system in another co-ordinate system. Using these facts we can change our basis to be the eigenvectors of our linear transformation, apply the neccessary scaling factors (the eigenvalues) and then change back to our original basis. This is known as the eigendecomposition of a matrix:

$$
\begin{aligned}
Ax &= \lambda x \\
AQ &= Q\Lambda \\
A &= Q\Lambda Q^{-1}
\end{aligned}
$$

Q is a square matrix where the ith column is the ith eigenvector q_i and Λ is a diagonal matrix where Λ_{ii} is the ith eigenvalue λ_i for q_i. Hopefully you can intuitively see how Q^{-1} changes basis so that the eigenvectors form the new basis vectors. Λ then just scales these basis vectors by their corresponding eigenvalues and Q changes basis back to our original coordinate system. This is all equivalent to just applying the matrix A! The eigendecomposition doesn't exist for all matrices and is only possible when the matrix in question is diagonizable:

$$Q^{-1}AQ = \Lambda$$

In other words, an nXn matrix A is only diagonizable if A has n linearly indepedent eigenvectors. Diagonalization of a matrix using eigenvectors is in itself a useful operation. For example, imagine we want to apply the same linear transformation D n times e.g. we could be simulating what happens to a point over n time steps. This can be denoted as follows:

$$\begin{aligned} u_1 &= Du_0 \\ u_2 &= Du_1 \\ &= DDu_0 \\ &= D^2u_0 \\ \therefore\ u_n &= D^nu_0 \end{aligned}$$

The problem with this is that if n becomes large then this requires an awful lot of matrix multiplications! Diagonalization can help us circumvent this problem. When a diagonal matrix is raised to a power you simply just need to raise the diagonal entries to that power (check this for yourself). Note how the matrix Λ containing the eigenvalues is a diagonal matrix. This means that when the eigenvectors of D form our new basis vectors the linear transformation D will be a diagonal matrix and so we can easily apply it many times. Then we just need to change back to our original basis once we are done. Mathematically we can just write this as:

$$D^n = Q\Lambda^nQ^{-1}$$

Where Q is a matrix containing the eigenvectors of D and Λ is a diagonal matrix containing the corresponding eigenvalues. When the eigendecomposition of a matrix exists this is a much more efficient process! Lets just work through one example to help demonstrate that the theory actually works. Lets say we have a matrix E and we want to apply it 3 times to the point z:

$$E = \begin{bmatrix} 6 & -1 \\ 2 & 3 \end{bmatrix}$$
$$z = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

If we just apply E 3 times to z then we get:

$$
\begin{aligned}
y &= E^3 z \\
&= EEEz \\
&= \begin{bmatrix} 6 & -1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 6 & -1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 6 & -1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\
&= \begin{bmatrix} 186 & -61 \\ 122 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\
&= \begin{bmatrix} 247 \\ 119 \end{bmatrix}
\end{aligned}
$$

To perform the diagonalization we need the matrix of eigenvectors and diagonal matrix of eigenvalues of E, which are as follows:

$$
\begin{aligned}
Q &= \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \\
Q^{-1} &= \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \\
\Lambda &= \begin{bmatrix} 5 & 0 \\ 0 & 4 \end{bmatrix}
\end{aligned}
$$

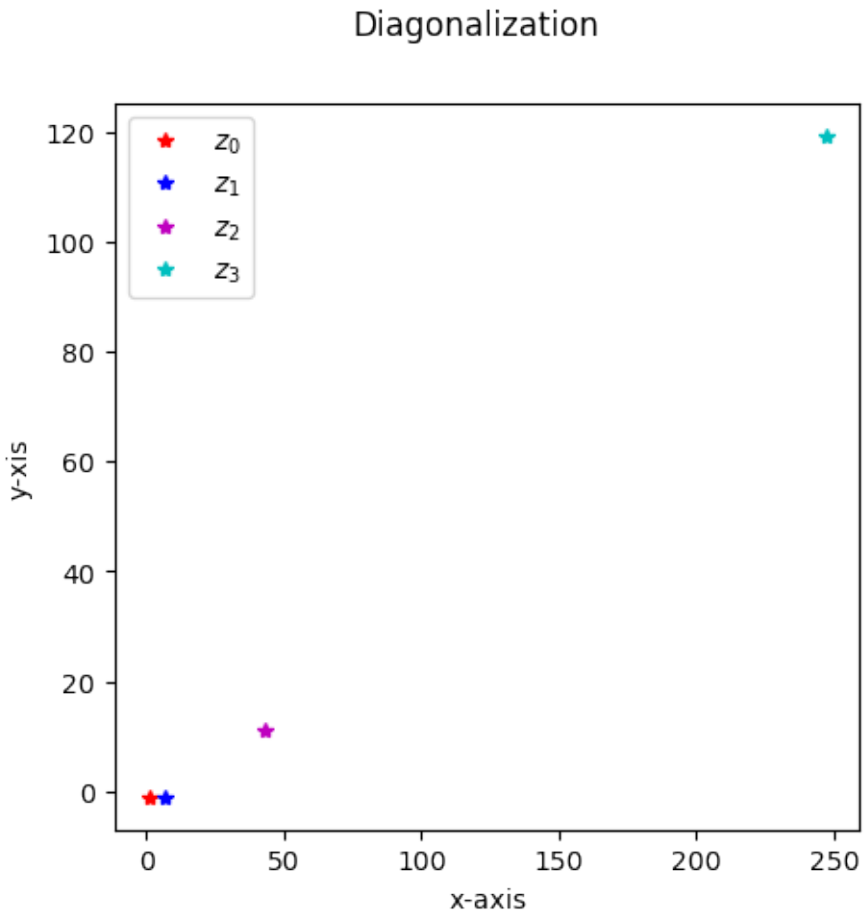
Applying diagonalization we get:

$$
\begin{aligned}
y &= E^3 z \\
&= (Q\Lambda Q^{-1})^3 z \\
&= (Q\Lambda Q^{-1})(Q\Lambda Q^{-1})(Q\Lambda Q^{-1})z \\
&= Q\Lambda^3 Q^{-1} z \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 125 & 0 \\ 0 & 64 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\
&= \begin{bmatrix} 186 & -61 \\ 122 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\
&= \begin{bmatrix} 247 \\ 119 \end{bmatrix}
\end{aligned}
$$

This matches the answer from just applying E tp z 3 times and hopefully proves to you that they are equivalent. Below is code demonstrating this example:

```
In [32]: E = np.array([[6, -1],[2, 3]])
         z = np.array([[1],[-1]])

         lam, Q = np.linalg.eig(E)
         Q_inv = np.linalg.inv(Q)
```

```
n = 3
colours = ['b*', 'm*', 'c*']

fig = plt.figure(figsize=(5,5))
plt.plot(z[0,0], z[1,0], 'r*', label='$z_0$')

for i in range(n):
    Lam = np.eye(2) * np.power(lam, i + 1)
    p = np.matmul(Q, np.matmul(Lam, np.matmul(Q_inv, z)))
    plt.plot(p[0,0], p[1,0], colours[i],
             label='$z_' + str(i + 1) + '$')

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



So far we have relied on intuition or numpy's eig() function to help find the eigenvectors and eigenvalues of a linear transformation but what if we want to calculate them ourselves? To do this we need to do some rearranging of the mathematical definition for an egienvector:

$$\begin{aligned} Ax &= \lambda x \\ Ax - \lambda x &= 0 \\ Ax - \lambda I x &= 0 \\ (A - \lambda I)x &= 0 \end{aligned}$$

The identity matrix is introduced because the subtraction of a scalar from a matrix is undefined. Note that for the expression $(A - \lambda I)x$ to equal 0, either $x = 0$ or $(A - \lambda I) = 0$. We can ignore the case where $x = 0$ because we are only interested in non-zero eigenvectors. Now for $(A - \lambda I)x = 0$ and for x to be a non-zero vector, it follows that $(A - \lambda I)$ must be non-invertible. If $(A - \lambda I)$ is invertible then the following occurs:

$$\begin{aligned} (A - \lambda I)x &= 0 \\ (A - \lambda I)^{-1}(A - \lambda I)x &= (A - \lambda I)^{-1}0 \\ x &= 0 \end{aligned}$$

We know from previous notebooks that for a matrix expression to be non-invertible it must have a zero determinant. We can therefore set the determinant of $(A - \lambda I)$ to zero and solve for λ:

$$det\left|A - \lambda I\right| = 0$$

Solving for this will give us our eigenvalues and we can then use these values in our original expression to find their corresponding eigenvectors. Lets take our matrix A from the first example in this notebook and solve for λ in order to find the eigenvalues:

$$\begin{aligned} 0 &= det\left|A - \lambda I\right| \\ &= det\left|\begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right| \\ &= det\begin{vmatrix} 2-\lambda & 1 \\ 0 & 2-\lambda \end{vmatrix} \\ &= ((2-\lambda)(2-\lambda)) - (1 * 0) \\ &= (2-\lambda)(2-\lambda) \\ \therefore\ \lambda &= 2 \end{aligned}$$

Using this value for λ back in our original expression we get:

$$
\begin{aligned}
0 &= (A - \lambda I)x \\
&= (\begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix})x \\
&= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x \\
&= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\
&= \begin{bmatrix} x_2 \\ 0 \end{bmatrix}
\end{aligned}
$$

In order for $\begin{bmatrix} x_2 \\ 0 \end{bmatrix} = 0$ the x_1 term can be any value but the x_2 term must be equal to 0. This means that the vector $\begin{bmatrix} t \\ 0 \end{bmatrix}$ for any real value of t is an eigenvector of A and has a corresponding eigen value of 2. As you can see we have arrived at the same solution as before, when we were looking at A graphically. If we had more than one solution for λ then we could repeat this procedure for each value to get its corresponding eigenvector.

Hopefully this notebook has given you a basic intuition of what eigenvectors and eigenvalues are and shown you that they are actually quite a simple concept. To recap, an eigenvector is simply a vector that does not change span during a linear transformation and its eigenvalue is the amount by which it is scaled.

The fact that eigenvectors and eigenvalues can be used to describe the characteristics of a linear transformation means that they appear often in mathematics and machine learning. The eigendecomposition is just one type of decomposition and as you progress on your journey through linear algebra and machine learning you will no doubt encounter several other types of matrix decomposition. Finally, it is rare that you will ever need to calculate the eigenvectors and eigenvalues of a matrix by hand but I hope that it was still useful to work through an example to understand how they are derived.