InversesAndDeterminants

August 11, 2018

The purpose of this notebook is to give you a conceptual understanding of inverses and the determinants. Both of these concepts are related to each other and build upon the last few notebooks looking at changes in basis and common matrix transformations.

Inverse:

To start with lets consider what we mean by the inverse of a matrix. Mathematically the iverse of a matrix is defined as:

$$AA^{-1} = I$$

i.e. a matrix applied to its inverse simply results in the identity matrix. Hopefully the previous notebooks might already give you some intuition about what this means... there is no change in basis or tranformation at all!

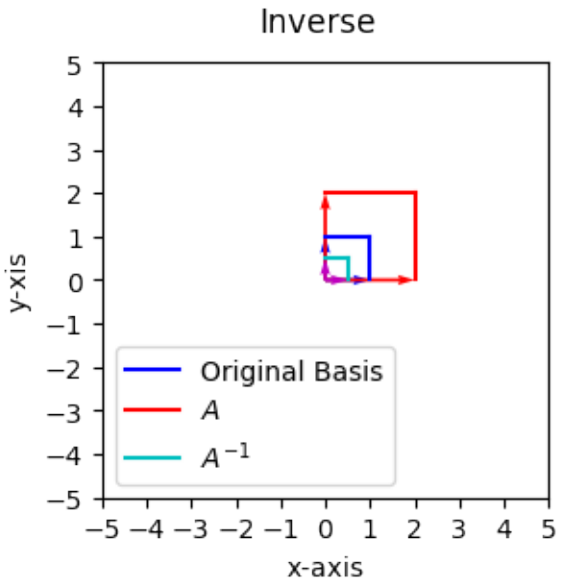
Conceptually the inverse of a matrix is extremely simple, it is just the opposite transformation to the one represented in the matrix. For example if A represents a new set of basis vectors that scales our standard basis vectors by a factor of 2, then A^{-1} simply scales our standard basis vectors by a factor of $\frac{1}{2}$. Similarly if B represents a new set of basis vectors that rotates our standard basis vectors by 90 degrees clockwise, then B^{-1} rotates our standard basis vectors by 90 degrees anti-clockwise. The code below demonstrates these two examples by demonstrating what happens to the basis vectors and a unit square for each transformation:

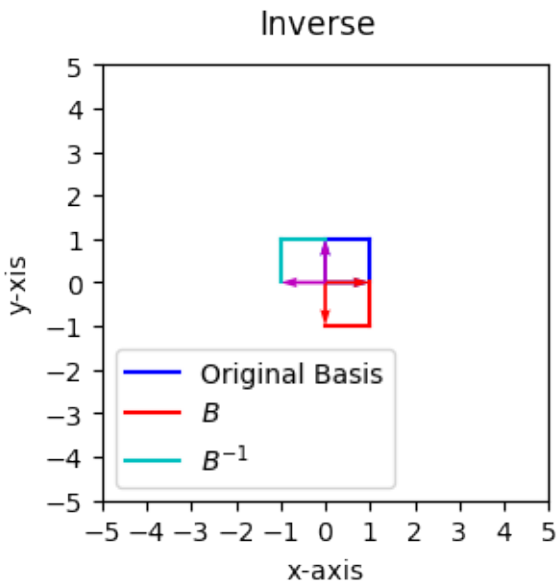
```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        mpl.rcParams['figure.dpi']= 100

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

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

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

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

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.)

new_top_left = np.matmul(A, top_left)
new_top_right = np.matmul(A, top_right)
new_bottom_right = np.matmul(A, bottom_right)
plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1],new_top_right[1]], 'r-',
         label='$A$')
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.)

new_top_left = np.matmul(np.linalg.inv(A), top_left)
new_top_right = np.matmul(np.linalg.inv(A), top_right)
new_bottom_right = np.matmul(np.linalg.inv(A), bottom_right)
plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]], 'c-',
         label='$A^{-1}$')
plt.plot([new_top_left[0], new_top_right[0]],
         [new_top_left[1], new_top_right[1]], 'c-')
plt.quiver(0, 0, np.linalg.inv(A)[0,0],
           np.linalg.inv(A)[1,0], color='m', units='xy',
           angles='xy', scale_units='xy', scale=1.)
plt.quiver(0, 0, np.linalg.inv(A)[0,1],
           np.linalg.inv(A)[1,1], color='m', units='xy',
           angles='xy', scale_units='xy', scale=1.)

plt.legend(loc='lower left')
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('Inverse')
plt.show()

B = np.array([[0, 1],[-1, 0]])

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.)

new_top_left = np.matmul(B, top_left)
new_top_right = np.matmul(B, top_right)
new_bottom_right = np.matmul(B, bottom_right)
plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]],
         'r-', label='$B$')
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.)

new_top_left = np.matmul(np.linalg.inv(B), top_left)
new_top_right = np.matmul(np.linalg.inv(B), top_right)
new_bottom_right = np.matmul(np.linalg.inv(B), bottom_right)
plt.plot([new_bottom_right[0], new_top_right[0]],
         [new_bottom_right[1], new_top_right[1]], 'c-',
         label='$B^{-1}$')
plt.plot([new_top_left[0], new_top_right[0]],
         [new_top_left[1], new_top_right[1]], 'c-')
plt.quiver(0, 0, np.linalg.inv(B)[0,0],
```

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

plt.legend(loc='lower left')
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('Inverse')
plt.show()
```





Hopefully now you can also see that if we apply a matrix and its inverse then we get no transformation at all and therefore we are just left with the identity matrix I.

Aside from being the reverse transformation, the inverse matrix has another interesting conceptual interpretation based on changing basis. Remember that the columns of a matrix give us the location of another set of basis vectors in our coordinate system. In effect the matrix give us the instructions of how to go from some other co-ordiante system that uses different basis vectors to our own co-ordinate system. Using this logic, we can see that the inverse of a matrix gives us the instructions to go from our co-ordiante system to another one.

For example imagine we have a friend ben who uses a different co-ordinate system to us. There is a point $\begin{bmatrix}1\\2\end{bmatrix}$ in Ben's co-ordinate system and we want to know what that point is in our co-ordinate system. If we we knew that Ben's basis vectors were at $\begin{bmatrix}2\\-1\end{bmatrix}$ and $\begin{bmatrix}-1\\1\end{bmatrix}$ in our co-ordinate system then we could simply construct a matrix $C = \begin{bmatrix}2 & -1\\-1 & 1\end{bmatrix}$ and apply it to the point $\begin{bmatrix}1\\2\end{bmatrix}$:

$$C\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}x\\y\end{bmatrix}$$
$$\begin{bmatrix}2 & -1\\-1 & 1\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix}$$

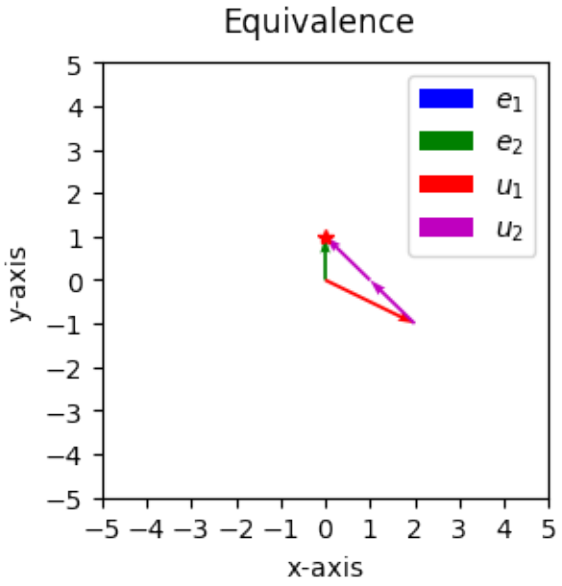
So the point $\begin{bmatrix}1\\2\end{bmatrix}$ in Ben's co-ordinate system is $\begin{bmatrix}0\\1\end{bmatrix}$ in our co-ordinate system. In comparison, the inverse gives us the instructions to go from our co-ordinate system to Ben's. Equivalently, the columns of the inverse matrix correspond to our basis vectors in Ben's co-ordinate system. We can see this just by applying the inverse of C to the above equation:

$$
\begin{aligned}
C\begin{bmatrix}1\\2\end{bmatrix} &= \begin{bmatrix}0\\1\end{bmatrix} \\
C^{-1}C\begin{bmatrix}1\\2\end{bmatrix} &= C^{-1}\begin{bmatrix}0\\1\end{bmatrix} \\
I\begin{bmatrix}1\\2\end{bmatrix} &= C^{-1}\begin{bmatrix}0\\1\end{bmatrix} \\
\begin{bmatrix}1\\2\end{bmatrix} &= C^{-1}\begin{bmatrix}0\\1\end{bmatrix} \\
\begin{bmatrix}1\\2\end{bmatrix} &= \begin{bmatrix}1 & 1\\1 & 2\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}
\end{aligned}
$$

The code below shows how the points $\begin{bmatrix}1\\2\end{bmatrix}$ and $\begin{bmatrix}0\\1\end{bmatrix}$ are equivalent given their respective basis vectors (e_1 and e_2 for our basis vectors and u_1 and u_2 for Ben's basis vectors).

```
In [2]: fig = plt.figure(figsize=(3,3))
        our_point = np.array([0, 1])
        our_basis = np.array([[1, 0],[0, 1]])

        bens_point = np.array([1, 2])
        bens_basis = np.array([[2, -1],[-1, 1]])

        plt.plot(our_point[0], our_point[1], 'r*')

        for x in range(our_point[0]):
            plt.quiver(x, 0, our_basis[0, 0], our_basis[1, 0],
                       color='b', units='xy', angles='xy',
                       scale_units='xy', scale=1.)

        for y in range(our_point[1]):
            plt.quiver(our_point[0], y, our_basis[0, 1],
                       our_basis[1, 1], color='g', units='xy',
                       angles='xy', scale_units='xy', scale=1.)


        for x in range(bens_point[0]):
            plt.quiver(x * bens_basis[0, 0], x * bens_basis[1, 0],
                       bens_basis[0, 0], bens_basis[1, 0],
                       color='r', units='xy', angles='xy',
                       scale_units='xy', scale=1.)
```

```
for y in range(bens_point[1]):
    plt.quiver((bens_point[0] * bens_basis[0, 0]) +
               (y * bens_basis[0, 1]),
               (bens_point[0] * bens_basis[1, 0]) +
               (y * bens_basis[1, 1]),
              bens_basis[0, 1], bens_basis[1, 1],
              color='m', units='xy', angles='xy',
              scale_units='xy', scale=1.)

from matplotlib.patches import Patch
legend_elements = [Patch(facecolor='b', label='$e_1$'),
                   Patch(facecolor='g', label='$e_2$'),
                   Patch(facecolor='r', label='$u_1$'),
                   Patch(facecolor='m', label='$u_2$')]

plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.legend(handles=legend_elements, loc='upper right')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.suptitle('Equivalence')
plt.show()
```



Can you think of an example when it might be useful to express a point in another co-ordinate system using the inverse of a matrix? Remember back to the first notebook on linear algebra

(change of basis) when we used simultaneous equations to obtain the price of individual apples and oranges given two shopping prices. We found that the price for an individual apple and orange was £1 and £0.5 repsectively:

$$M = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} * \begin{bmatrix} 1 \\ .5 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

We could have simlarly found this answer by using the inverse:

$$M^{-1} = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.5 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.5 \end{bmatrix} * \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ .5 \end{bmatrix}$$

What is great about using the inverse is that we can now obtain the individual apple and orange prices for any given shopping prices as long as the numbers of apples and oranges don't change! For example imagine we bought 2 apples and 2 oranges on one day and 2 apples and 4 oranges on another day, but this time it cost us £4 and £5 respectively. Using the inverse we can easily find the price of an individual apple and orange again:

$$\begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.5 \end{bmatrix} * \begin{bmatrix} 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix}$$

So with these new shopping prices the individual price of an apple was £1.5 and the individual price of an orange was £0.5.

Lets cover one final example of why the inverse of a matrix can be so useful in linear algebra. Imagine we have a vector that we want to rotate in our co-ordinate system by 90 degrees clockwise. However we only know the location of this vector using Ben's co-ordinate system ($\begin{bmatrix} 1 \\ 2 \end{bmatrix}$) and Ben would like the rotated vector to be expressed in terms of his co-ordinate system. Lets assume we know the location of Ben's basis vectors in our co-ordinate system ($C = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$).

The problem we have is that we don't know the transformation matrix D in Ben's co-ordinate system that corresponds to a 90 degree clockwise rotation in our co-ordinate system. This is because his basis vectors are not orthogonal to each other or of unit length. However we do know the transformation matrix E for our nice orthogonal, unit length basis vectors:

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

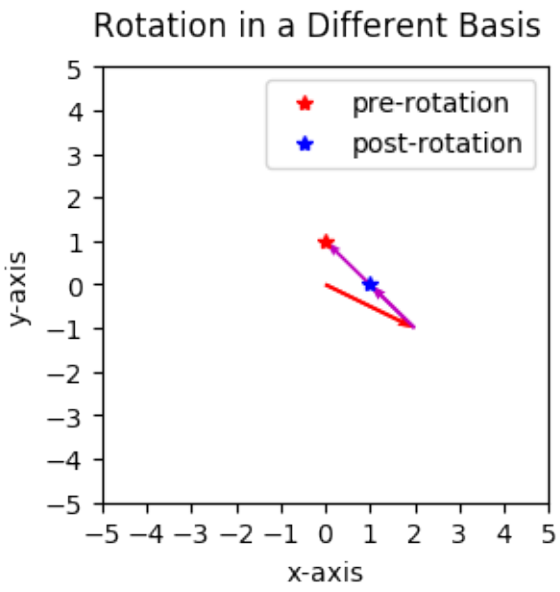
We can use the inverse of C to help us find out what the new rotated point wil be in Ben's co-ordiante system. To do this remember that C^{-1} contains our basis vectors in Ben's co-ordinate system and give us the instructions to go from our co-ordinate system to Ben's. So what we can do is express Ben's point in our co-ordinate system using C, do the rotation in our co-ordinate system using E and then express the new point back in Ben's co-ordinate system using C^{-1}. We therefore get:

$$\begin{aligned}
\begin{bmatrix} x \\ y \end{bmatrix} &= D \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\
&= C^{-1}EC \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} -3 & 2 \\ -5 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\end{aligned}$$

So the point $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ in Ben's co-ordiante system rotated by 90 degrees clockwise in our co-ordinate system is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ in Ben's co-ordiante system. The code below demonstrates graphically that our dervied matrix $D = \begin{bmatrix} -3 & 2 \\ -5 & 3 \end{bmatrix}$ does indeed rotate Ben's point $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ by 90 degrees clockwise in our co-ordinate system.

```
In [3]: fig = plt.figure(figsize=(3,3))
        plt.plot(our_point[0], our_point[1], 'r*',
                 label='pre-rotation')

        for x in range(bens_point[0]):
            plt.quiver(x * bens_basis[0, 0],
                       x * bens_basis[1, 0],
                       bens_basis[0, 0], bens_basis[1, 0],
                       color='r', units='xy', angles='xy',
                       scale_units='xy', scale=1.)

        for y in range(bens_point[1]):
            plt.quiver((bens_point[0] * bens_basis[0, 0]) +
                       (y * bens_basis[0, 1]),
                       (bens_point[0] * bens_basis[1, 0]) +
                       (y * bens_basis[1, 1]),
                       bens_basis[0, 1], bens_basis[1, 1],
                       color='m', units='xy', angles='xy',
```

```
                scale_units='xy', scale=1.)

D = np.array([[-3, 2],[-5, 3]])
bens_new_point = np.matmul(D, bens_point)

plt.plot(np.matmul(bens_basis, bens_new_point)[0],
         np.matmul(bens_basis, bens_new_point)[1],
         'b*', label='post-rotation')

for x in range(bens_new_point[0]):
    plt.quiver(x * bens_basis[0, 0],
               x * bens_basis[1, 0],
               bens_basis[0, 0], bens_basis[1, 0],
               color='r', units='xy', angles='xy',
               scale_units='xy', scale=1.)

for y in range(bens_new_point[1]):
    plt.quiver((bens_new_point[0] * bens_basis[0, 0]) +
               (y * bens_basis[0, 1]),
               (bens_new_point[0] * bens_basis[1, 0]) +
               (y * bens_basis[1, 1]),
               bens_basis[0, 1], bens_basis[1, 1],
               color='m', units='xy', angles='xy',
               scale_units='xy', scale=1.)

plt.xlim([-5, 5])
plt.xticks(np.arange(-5, 6))
plt.ylim([-5, 5])
plt.yticks(np.arange(-5, 6))
plt.legend(loc='upper right')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.suptitle('Rotation in a Different Basis')
plt.show()
```



You will see expressions like $C^{-1}EC$ alot in linear algebra and hopefully now you can see why they are so useful. You are essentially applying a transformation in a different basis and then transforming back to the original basis. Note that if we wanted to do the opposite and transform a point in our co-ordinate system via Ben's co-ordinate system then we could just use the expression CFC^{-1}.

So far I have just given you the inverse of a matrix but you may well be asking how do we calculate the inverse of a matrix? It is rare that we need to calculate it by hand as we commonly rely on our computer to calculate it e.g. using np.linalg.inv() in python. With that being said, for completeness sake, we shall quickly go through one method of calculating the inverse using gaussian elimination. Recall the definition of the inverse of a matrix:

$$GG^{-1} = I$$

Lets use the matrix C again to illustrate how to use gaussian elimination to find the inverse. Our problem can be written as follows:

$$\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} C^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} c_{11}^{-1} & c_{12}^{-1} \\ c_{21}^{-1} & c_{22}^{-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Where c_{11}^{-1} indexes the first row and first column of C^{-1}. Now by applying successive row operations to the above expression we can get C to become the identity matrix and so the left side

just becomes C^{-1}. Importantly then, whatever we are left with on the right hand side must be our solution for C^{-1}.

$$\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} c_{11}^{-1} & c_{12}^{-1} \\ c_{21}^{-1} & c_{22}^{-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \text{add row 2 to row 1}$$

$$\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} c_{11}^{-1} & c_{12}^{-1} \\ c_{21}^{-1} & c_{22}^{-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad \text{add row 1 to row 2}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{11}^{-1} & c_{12}^{-1} \\ c_{21}^{-1} & c_{22}^{-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

This solution for C^{-1} matches the one we used earlier in the notebook and so provides a nice sanity check for the solution I gave you previously. We can also double check our answer by ensuring that $CC^{-}1 = I$:

$$CC^{-}1 = I$$

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

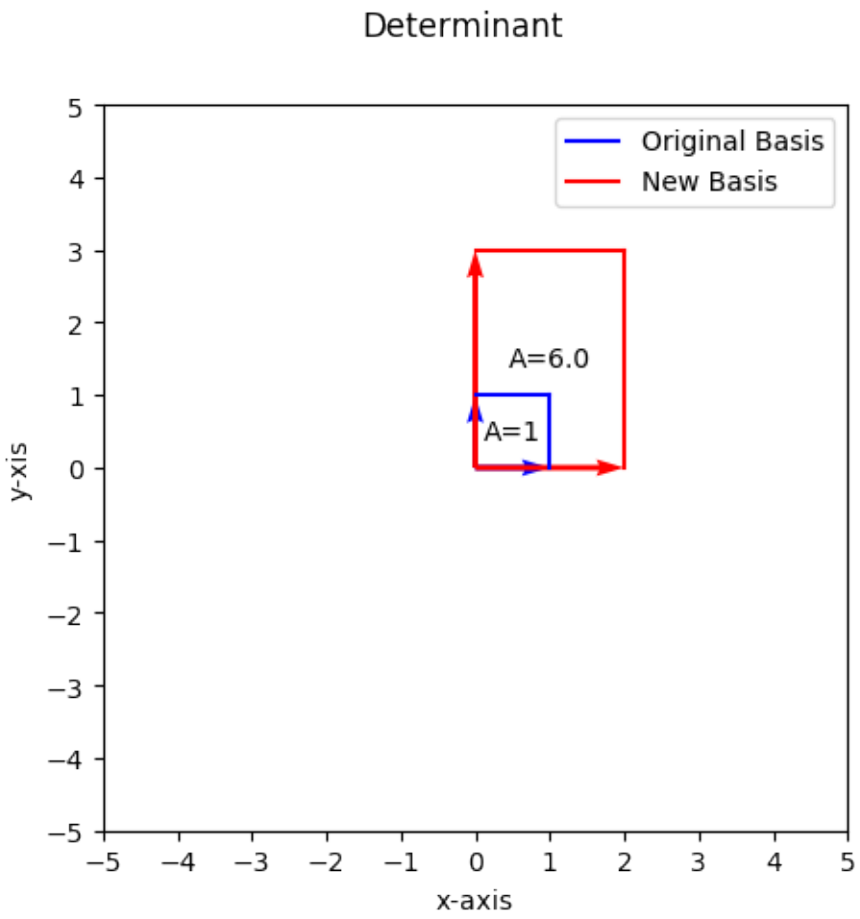
Hopefully now you have a better understanding of why the inverse is useful from a conceptual point of view and also an idea of how to calculate the inverse by hand. Next we will discuss the determinant of a matrix, which has some interesting links to the inverse and is another important step towards a thorough understanding of linear algebra.

Determinant:

As with the inverse of a matrix, the determinant of a matrix has a relatively simple geometric interpretation. The determinant of a matrix H is denoted as $|H|$ and is a single value that tells you how much space is scaled by after applying the linear transformation H. For example imagine that $H = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$, this transformation scales our first basis vector by a factor of 2 and our second basis vector by a factor of 3. Our original unit square had an area of 1 but after the transformation it has an area of 2 * 3 = 6. Intuitively, H has scaled space by a factor of 6 and so $|H|$ is equal to 6. The code below demonstrates this graphically:

```
In [4]: H = np.array([[2, 0],[0, 3]])

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

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

```
fig = plt.figure(figsize=(5,5))

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.text(basis_x[0] / 2, basis_y[1] / 2, 'A=1',
         horizontalalignment='center',
         verticalalignment='center')

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

plt.text(H[0, 0] / 2, H[1, 1] / 2,
         'A=' + str(np.linalg.det(H)),
         horizontalalignment='center',
         verticalalignment='center')

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('Determinant')
plt.show()
```



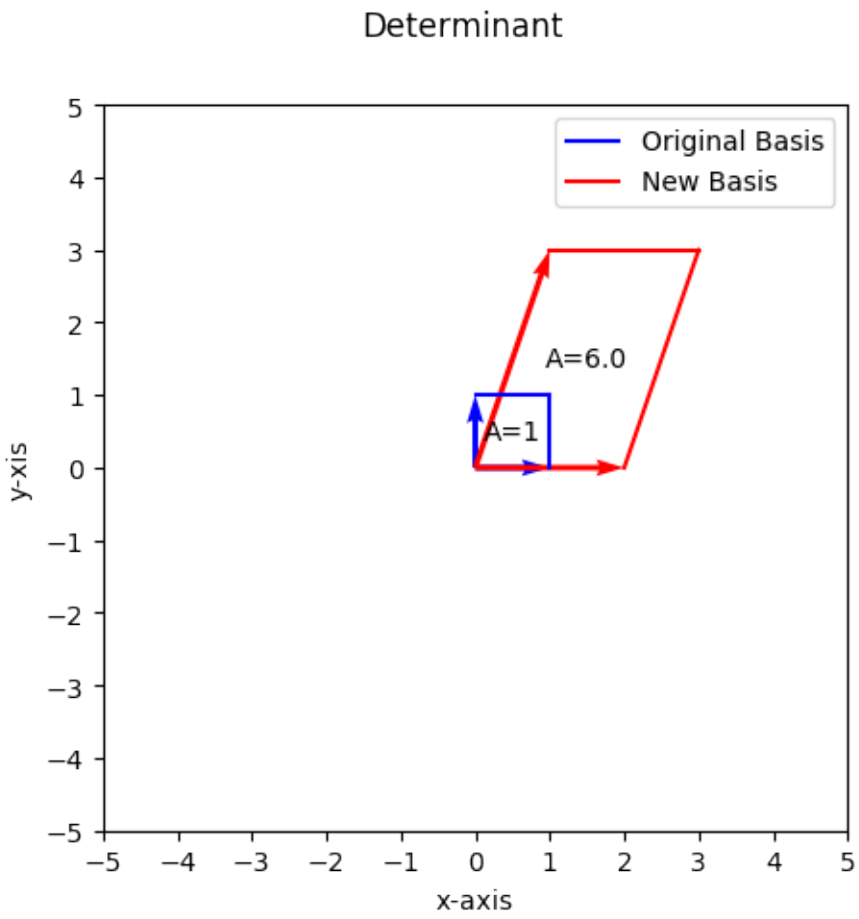
To emphasise the point let us consider another transformation J:

$$J = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$$

This time the matrix performs a shear and a scaling of our original basis vectors. However, as the code shows below, this changes our unit square into a parallelogram and the area of this parallelogram is still 2 * 3 = 6. This means that $|J| = |H| = 6$.

```
In [5]: J = np.array([[2, 1],[0, 3]])

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

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

```
fig = plt.figure(figsize=(5,5))

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.text(basis_x[0] / 2, basis_y[1] / 2, 'A=1',
         horizontalalignment='center',
         verticalalignment='center')

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

plt.text((J[0, 0] + J[0, 1]) / 2, J[1, 1] / 2,
         'A=' + str(np.linalg.det(J)),
         horizontalalignment='center',
         verticalalignment='center')

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('Determinant')
plt.show()
```



A common way to learn how to calculate the determinant of a 2 X 2 matrix is the following formula:

$$det \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

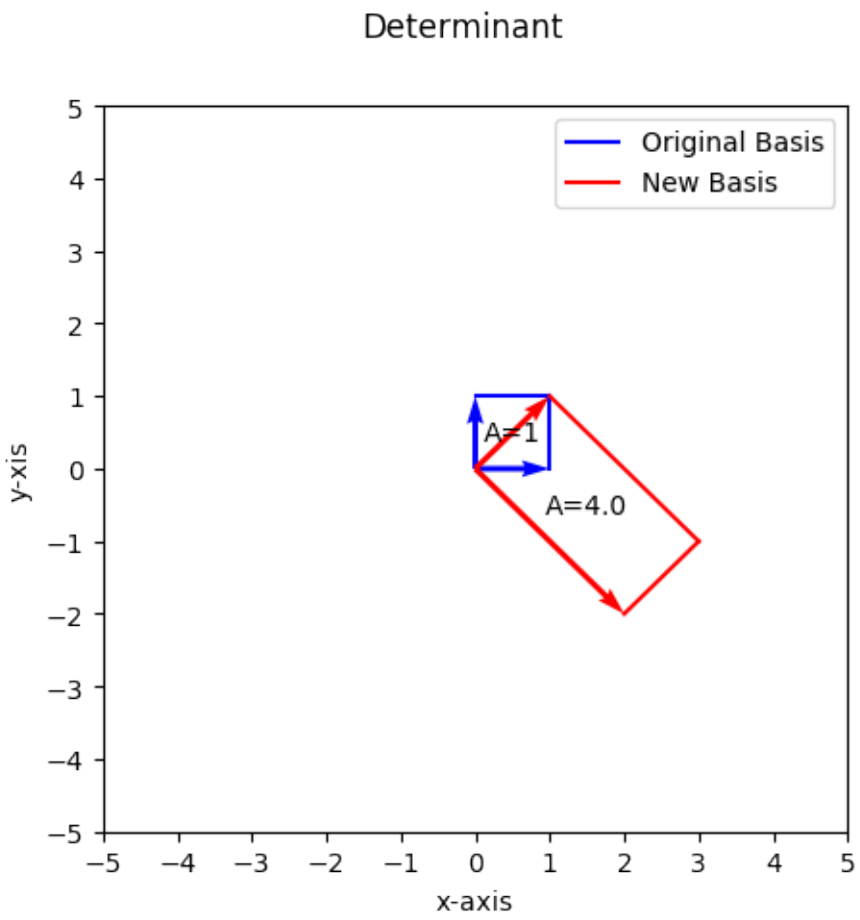
For example what is space scaled by when applying the the linear transformation $K = \begin{vmatrix} 2 & 1 \\ -2 & 1 \end{vmatrix}$? Using the above formula we can easily find out:

$$\begin{aligned} det\,|K| &= det \begin{vmatrix} 2 & 1 \\ -2 & 1 \end{vmatrix} \\ &= (2 * 1) - (1 * -2) \\ &= 4 \end{aligned}$$

So K scales space by a factor of 4. This is demonstrated in the code below:

In [6]:
```
K = np.array([[2, 1],[-2, 1]])

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

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

fig = plt.figure(figsize=(5,5))

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.text(basis_x[0] / 2, basis_y[1] / 2, 'A=1',
         horizontalalignment='center',
         verticalalignment='center')

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

plt.text((K[0, 0] + K[0, 1]) / 2, (K[1, 0] + K[1, 1]) / 2,
         'A=' + str(np.linalg.det(K)),
         horizontalalignment='center',
         verticalalignment='center')

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('Determinant')
plt.show()
```



Our new basis vectors are still orthogonal and so our unit square remains rectangular but with an area of 4. We can check this value by noticing that the area of our new ractangle can be calculated as follows (e'_1 and e'_2 denote our new basis vectors):

$$\begin{aligned}\|e'_1\| * \|e'_2\| &= \|\begin{bmatrix}2\\-2\end{bmatrix}\| * \|\begin{bmatrix}1\\1\end{bmatrix}\| \\ &= \sqrt{2^2 + -2^2} * \sqrt{1^2 + 1^2} \\ &= \sqrt{8} * \sqrt{2} \\ &= \sqrt{4} * \sqrt{2} * \sqrt{2} \\ &= 4\end{aligned}$$

For larger matrices the calculation of the detemrinant because much more complex and is beyond the scope of this notebook. The main purpose of this notebook is to give you a strong conceptual

understanding of the determinant rather than teach you how to calculate everything by hand. These days we have computers to do the tedious work for us!

The final concept I want to cover in this notebook and one that links the inverse and the determinant together is the concept of a zero determinant ($det\left|L\right| = 0$). A zero determinant means that a transformation has caused the dimensionality of our co-ordinate space to decrease. Another way of saying this is that if the determinant of a matrix is zero then that matrix has caused space to collapse. For example imagine $L = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$. Using our formula for calculating the determinant of a 2 X 2 matrix we have:

$$\begin{aligned} det\left|L\right| &= det\begin{vmatrix} 1 & -1 \\ -1 & 1 \end{vmatrix} \\ &= (1 * 1) - (-1 * -1) \\ &= 0 \end{aligned}$$

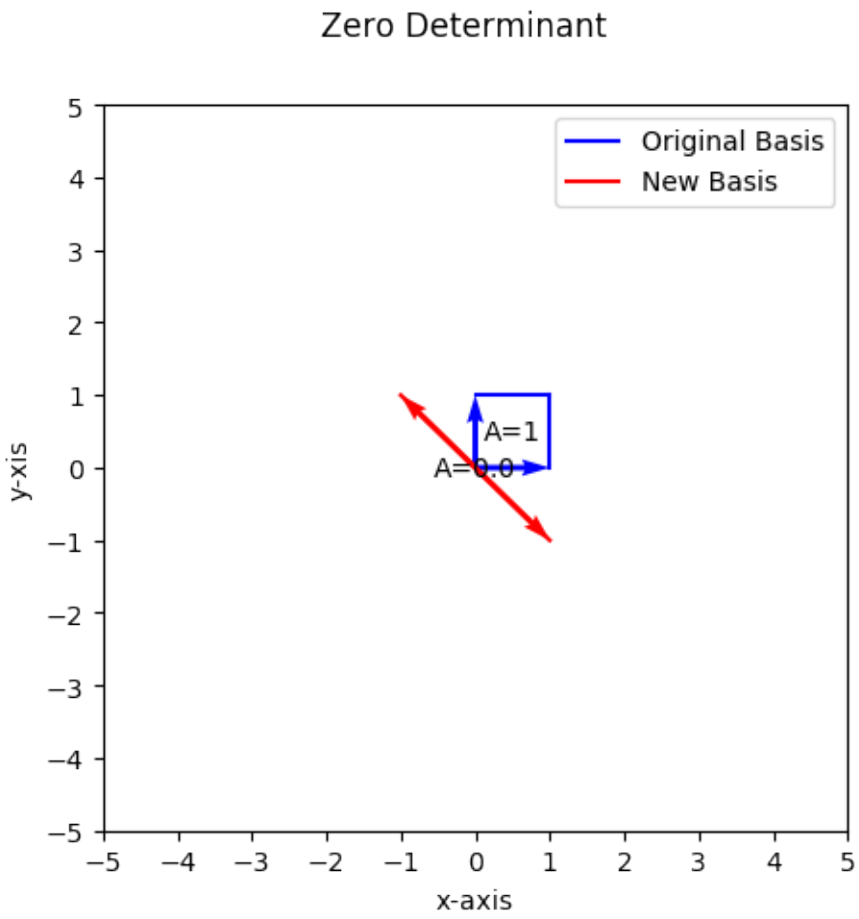
So the matrix L must decrease the dimensionality of our co-ordiante space i.e. our space is no longer a 2D plane but just a 1D line. We can see why this is the case because the new basis vectors are no longer linearly indepedent. The first new basis vector $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ can be expressed as a linear combination of the second new basis vector $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} = -1 * \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Therefore the new basis vectors cover the same span! The code below demonstrates this transformation from a 2D plane to a 1D line. Note how our unit square goes from having an area of 1 to an area of 0.

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

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

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

        fig = plt.figure(figsize=(5,5))

        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.text(basis_x[0] / 2, basis_y[1] / 2, 'A=1',
         horizontalalignment='center',
         verticalalignment='center')

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

plt.text((L[0, 0] + L[0, 1]) / 2, (L[1, 0] + L[1, 1]) / 2,
         'A=' + str(np.linalg.det(L)),
         horizontalalignment='center',
         verticalalignment='center')

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('Zero Determinant')
plt.show()
```



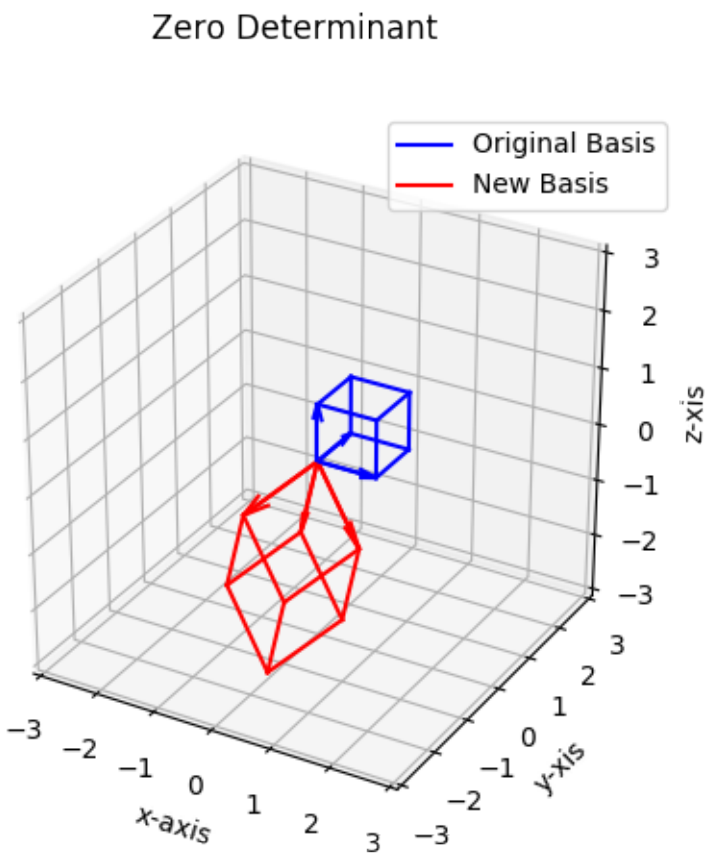
What happens when our original co-ordinate system is 3 dimensional? In this case a matrix with zero determinant will reduce space to either 1 or 2 dimensions. If we collapse from a 3 dimensional space to a 2 dimensional space then the volume of our unit cube has become zero. Similarly, any one of the new basis vectors can be described as a linear combination of the other two new basis vectors. The code below demonstrates this with the following matrix:

$$M = \begin{bmatrix} 1 & -1 & 0 \\ -.5 & -.5 & -.5 \\ -1 & -1 & -1 \end{bmatrix}$$

```
In [11]: standard_basis = np.array([[1, 0, 0],[0, 1, 0],[0, 0, 1]])
         M = np.array([[1, -1, 0],[-.5, -.5, -.5], [-1, -1, -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(M, point1)
point2 = np.matmul(M, point2)
point3 = np.matmul(M, point3)
point4 = np.matmul(M, point4)
point5 = np.matmul(M, point5)
point6 = np.matmul(M, point6)
point7 = np.matmul(M, point7)
point8 = np.matmul(M, 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, M[0,0], M[1, 0], M[2, 0], color='r')
ax.quiver(0, 0, 0, M[0,1], M[1, 1], M[2, 1], color='r')
ax.quiver(0, 0, 0, M[0,2], M[1, 2], M[2, 2], color='r')

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('Zero Determinant')
plt.show()
```



While it is hard to see in a static 3D plot, our unit cube has actually been flattened into a 2D shape, with all the points projected onto the 2D plane spanned by the new basis vectors. Any of the new basis vectors can be expressed as a linear combination of the other two. This means there are only two linearly independent basis vectors present and the dimensionality of our new space is 2:

$$\begin{bmatrix} 1 \\ -.5 \\ -1 \end{bmatrix} = (2 * \begin{bmatrix} 0 \\ -.5 \\ -1 \end{bmatrix}) - \begin{bmatrix} -1 \\ -.5 \\ -1 \end{bmatrix}$$
$$= \begin{bmatrix} 0 \\ -1 \\ -2 \end{bmatrix} - \begin{bmatrix} -1 \\ -.5 \\ -1 \end{bmatrix}$$
$$= \begin{bmatrix} 1 \\ -.5 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 \\ -.5 \\ -1 \end{bmatrix} = (2 * \begin{bmatrix} 0 \\ -.5 \\ -1 \end{bmatrix}) - \begin{bmatrix} 1 \\ -.5 \\ -1 \end{bmatrix}$$
$$= \begin{bmatrix} 0 \\ -1 \\ -2 \end{bmatrix} - \begin{bmatrix} 1 \\ -.5 \\ -1 \end{bmatrix}$$
$$= \begin{bmatrix} -1 \\ -.5 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ -.5 \\ -1 \end{bmatrix} = \frac{1}{2} * (\begin{bmatrix} 1 \\ -.5 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ -.5 \\ -1 \end{bmatrix})$$
$$= \frac{1}{2} * \begin{bmatrix} 0 \\ -1 \\ -2 \end{bmatrix}$$
$$= \begin{bmatrix} 0 \\ -.5 \\ -1 \end{bmatrix}$$

In linear algebra we say the rank of the matrix M is therefore equal to 2. It is often useful to calculate the rank of an arbitrary matrix (i.e. the number of linearly independent basis vectors) because it can tell us if the determinant will be 0 (rank < dimensionality of original space). To do this we can use gaussian elimination to convert the matrix M into row echelon form. The number of linearly indepedent basis vectors and therefore rank of the matrix is then equal to the number of non-zero rows in the row echelon matrix. Below we use this method on M to confirm that it has rank 2:

$$M = \begin{bmatrix} 1 & -1 & 0 \\ -.5 & -.5 & -.5 \\ -1 & -1 & -1 \end{bmatrix} \qquad \text{subtract 2 * row 2 from row 3}$$
$$= \begin{bmatrix} 1 & -1 & 0 \\ -.5 & -.5 & -.5 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{add } \frac{1}{2} \text{ * row 1 to row 2}$$
$$= \begin{bmatrix} 1 & -1 & 0 \\ 0 & -1 & -.5 \\ 0 & 0 & 0 \end{bmatrix}$$

The final concept to cover in this notebook is the link between the determinant and the inverse of a matrix. Put simply if a matrix has zero determinant then no inverse exists. Intuitively it makes sense that once you have collapsed space and one of your basis vectors has become linearly depedent then you have no way of going back! In effect you have lost the ability to express

that dimension for good. Officially we define non-invertible, zero determinant matrices as being singular.

Just as a final note you may be wandering what happens when the matrix is non-square i.e. the number of rows is larger than the number of columns or vice versa. For these matrices you either end up gaining a dimension (number of rows > number of columns) or losing a dimension (number of rows < number of cols). As a result the determinant and inverse of a non-square matrix is not defined.