More Matrices

This notebook continues your exploration of matrices.

Matrix Multiplication

Multiplying matrices is a little more complex than the operations we've seen so far. There are two cases to consider, *scalar multiplication* (multiplying a matrix by a single number), and *dot product matrix multiplication* (multiplying a matrix by another matrix.

Scalar Multiplication

To multiply a matrix by a scalar value, you just multiply each element by the scalar to produce a new matrix:

$$2 \times \begin{bmatrix}1 & 2 & 3 \\ 4 & 5 & 6\end{bmatrix} = \begin{bmatrix}2 & 4 & 6 \\ 8 & 10 & 12\end{bmatrix}$$

In Python, you perform this calculation using the ***** operator:

In [1]:
```python
import numpy as np

A = np.array([[1,2,3],
              [4,5,6]])
print(2 * A)
```

```
[[ 2  4  6]
 [ 8 10 12]]
```

Dot Product Matrix Multiplication

To mulitply two matrices together, you need to calculate the *dot product* of rows and columns. This means multiplying each of the elements in each row of the first matrix by each of the elements in each column of the second matrix and adding the results. We perform this operation by applying the *RC* rule - always multiplying ***R***ows by ***C***olumns. For this to work, the number of ***columns*** in the first matrix must be the same as the number of ***rows*** in the second matrix so that the matrices are *conformable* for the dot product operation.

Sounds confusing, right?

Let's look at an example:

$$\begin{bmatrix}1 & 2 & 3 \\ 4 & 5 & 6\end{bmatrix} \cdot \begin{bmatrix}9 & 8 \\ 7 & 6 \\ 5 & 4\end{bmatrix}$$

Note that the first matrix is 2x3, and the second matrix is 3x2. The important thing here is that the first matrix has two rows, and the second matrix has two columns. To perform the multiplication, we first take the dot product of the first ***row*** of the first matrix (1,2,3) and the first ***column*** of the second matrix (9,7,5):

$$(1,2,3) \cdot (9,7,5) = (1 \times 9) + (2 \times 7) + (3 \times 5) = 38$$

In our resulting matrix (which will always have the same number of ***rows*** as the first matrix, and the same number of ***columns*** as the second matrix), we can enter this into the first row and first column element:

$$\begin{bmatrix}38 & ? \\ ? & ?\end{bmatrix}$$

Now we can take the dot product of the first row of the first matrix and the second column of the second matrix:

$$(1,2,3) \cdot (8,6,4) = (1 \times 8) + (2 \times 6) + (3 \times 4) = 32$$

Let's add that to our resulting matrix in the first row and second column element:

$$\begin{bmatrix}38 & 32 \\ ? & ?\end{bmatrix}$$

Now we can repeat this process for the second row of the first matrix and the first column of the second matrix:

$$(4,5,6) \cdot (9,7,5) = (4 \times 9) + (5 \times 7) + (6 \times 5) = 101$$

Which fills in the next element in the result:

$$\begin{bmatrix}38 & 32 \\ 101 & ?\end{bmatrix}$$

Finally, we get the dot product for the second row of the first matrix and the second column of the second matrix:

$$(4,5,6) \cdot (8,6,4) = (4 \times 8) + (5 \times 6) + (6 \times 4) = 86$$

Giving us:

$$\begin{bmatrix}38 & 32 \\ 101 & 86\end{bmatrix}$$

In Python, you can use the *numpy.**dot* function or the **@ operator to multiply matrices and two-dimensional arrays:

In [2]:
```
import numpy as np

A = np.array([[1,2,3],
              [4,5,6]])
B = np.array([[9,8],
              [7,6],
              [5,4]])
print(np.dot(A,B))
print(A @ B)
```
```
[[ 38  32]
 [101  86]]
[[ 38  32]
 [101  86]]
```

This is one case where there is a difference in behavior between numpy.*array* and *numpy.matrix*, **You can also use a regular multiplication (*****) operator with a matrix, but not with an array:

In [3]:
```
import numpy as np

A = np.matrix([[1,2,3]
               ,[4,5,6]])
B = np.matrix([[9,8],
               [7,6],
               [5,4]])
print(A * B)
```
```
[[ 38  32]
 [101  86]]
```

Note that, unlike with multiplication of regular scalar numbers, the order of the operands in a multiplication operation is significant. For scalar numbers, the *commmutative law* of multiplication applies, so for example:

$$2 \times 4 = 4 \times 2$$

With matrix multiplication, things are different, for example:

$$\begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} \neq \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} \cdot \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

Run the following Python code to test this:

In [4]:
```
import numpy as np

A = np.array([[2,4],
              [6,8]])
B = np.array([[1,3],
              [5,7]])
print(A @ B)
print(B @ A)
```
```
[[22 34]
 [46 74]]
[[20 28]
 [52 76]]
```

Identity Matrices

An *identity* matrix (usually indicated by a capital **I**) is the equivalent in matrix terms of the number **1**. It always has the same number of rows as columns, and it has the value **1** in the diagonal element positions $I_{1,1}$, $I_{2,2}$, etc; and 0 in all other element positions. Here's an example of a 3x3 identity matrix:

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

Multiplying any matrix by an identity matrix is the same as multiplying a number by 1; the result is the same as the original value:

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix} \cdot \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

If you doubt me, try the following Python code!

In [5]:
```
import numpy as np

A = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
B = np.array([[1,0,0],
              [0,1,0],
              [0,0,1]])
print(A @ B)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Matrix Division

You can't actually divide by a matrix; but when you want to divide matrices, you can take advantage of the fact that division by a given number is the same as multiplication by the reciprocal of that number. For example:

$$6 \div 3 = \frac{1}{3} \times 6$$

In this case, $^1/_3$ is the reciprocal of 3 (which as a fraction is $^3/_1$ - we "flip" the numerator and denominator to get the reciprocal). You can also write $^1/_3$ as 3^{-1}.

Inverse of a Matrix

For matrix division, we use a related idea; we multiply by the *inverse* of a matrix:

$$A \div B = A \cdot B^{-1}$$

The inverse of B is B^{-1} as long as the following equation is true:

$$B \cdot B^{-1} = B^{-1} \cdot B = I$$

I, you may recall, is an *identity* matrix; the matrix equivalent of 1.

So how do you calculate the inverse of a matrix? For a 2x2 matrix, you can follow this formula:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

What happened there?

- We swapped the positions of *a* and *d*
- We changed the signs of *b* and *c*
- We multiplied the resulting matrix by 1 over the *determinant* of the matrix (*ad-bc*)

Let's try with some actual numbers:

$$\begin{bmatrix} 6 & 2 \\ 1 & 2 \end{bmatrix}^{-1} = \frac{1}{(6 \times 2) - (2 \times 1)} \begin{bmatrix} 2 & -2 \\ -1 & 6 \end{bmatrix}$$

So:

$$\begin{bmatrix} 6 & 2 \\ 1 & 2 \end{bmatrix}^{-1} = \frac{1}{10} \begin{bmatrix} 2 & -2 \\ -1 & 6 \end{bmatrix}$$

Which gives us the result:

$$\begin{bmatrix} 6 & 2 \\ 1 & 2 \end{bmatrix}^{-1} = \begin{bmatrix} 0.2 & -0.2 \\ -0.1 & 0.6 \end{bmatrix}$$

To check this, we can multiply the original matrix by its inverse to see if we get an identity matrix. This makes sense if you think about it; in the same way that 3 x $^1/_3$ = 1, a matrix multiplied by its inverse results in an identity matrix:

$$\begin{bmatrix} 6 & 2 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0.2 & -0.2 \\ -0.1 & 0.6 \end{bmatrix}$$

$$= \begin{bmatrix} (6 \times 0.2) + (2 \times -0.1) & (6 \times -0.2) + (2 \times 0.6) \\ (1 \times 0.2) + (2 \times -0.1) & (1 \times -0.2) + (2 \times 0.6) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

In [6]:
```python
import numpy as np

B = np.array([[6,2],
              [1,2]])

print(np.linalg.inv(B))
```
```
[[ 0.2 -0.2]
 [-0.1  0.6]]
```

Additionally, the *matrix* type has an ***I*** method that returns the inverse matrix:

In [7]:
```python
import numpy as np

B = np.matrix([[6,2],
              [1,2]])

print(B.I)
```
```
[[ 0.2 -0.2]
 [-0.1  0.6]]
```

For larger matrices, the process to calculate the inverse is more complex. Let's explore an example based on the following matrix:

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix}$$

The process to find the inverse consists of the following steps:

1: Create a matrix of *minors* by calculating the *determinant* for each element in the matrix based on the elements that are <u>not</u> in the same row or column; like this:

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (2 \times 8) - (4 \times 2) = 8 \quad \begin{bmatrix} 8 & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (6 \times 8) - (4 \times 2) = 40 \quad \begin{bmatrix} 8 & 40 & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (6 \times 2) - (2 \times 2) = 8 \quad \begin{bmatrix} 8 & 40 & 8 \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (2 \times 8) - (2 \times 2) = 12 \quad \begin{bmatrix} 8 & 40 & 8 \\ 12 & ? & ? \\ ? & ? & ? \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (4 \times 8) - (2 \times 2) = 28 \quad \begin{bmatrix} 8 & 40 & 8 \\ 12 & 28 & ? \\ ? & ? & ? \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (4 \times 2) - (2 \times 2) = 4 \quad \begin{bmatrix} 8 & 40 & 8 \\ 12 & 28 & 4 \\ ? & ? & ? \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (2 \times 4) - (2 \times 2) = 4 \quad \begin{bmatrix} 8 & 40 & 8 \\ 12 & 28 & 4 \\ 4 & ? & ? \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (4 \times 4) - (2 \times 6) = 4 \quad \begin{bmatrix} 8 & 40 & 8 \\ 12 & 28 & 4 \\ 4 & 4 & ? \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 2 \\ 6 & 2 & 4 \\ 2 & 2 & 8 \end{bmatrix} \quad (4 \times 2) - (2 \times 6) = -4 \quad \begin{bmatrix} 8 & 40 & 8 \\ 12 & 28 & 4 \\ 4 & 4 & -4 \end{bmatrix}$$

2: Apply *cofactors* to the matrix by switching the sign of every alternate element in the matrix of minors:

$$\begin{bmatrix} 8 & -40 & 8 \\ -12 & 28 & -4 \\ 4 & -4 & -4 \end{bmatrix}$$

In [8]:

```python
import numpy as np

B = np.array([[4,2,2],
              [6,2,4],
              [2,2,8]])

print(np.linalg.inv(B))
```

```
[[-0.25   0.375 -0.125]
 [ 1.25  -0.875  0.125]
 [-0.25   0.125  0.125]]
```

Multiplying by an Inverse Matrix

Now that you know how to calculate an inverse matrix, you can use that knowledge to multiply the inverse of a matrix by another matrix as an alternative to division:

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \cdot \begin{bmatrix}6 & 2\\1 & 2\end{bmatrix}^{-1}$$

$$= \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \cdot \begin{bmatrix}0.2 & -0.2\\-0.1 & 0.6\end{bmatrix}$$

$$= \begin{bmatrix}(1 \times 0.2) + (2 \times -0.1) & (1 \times -0.2) + (2 \times 0.6)\\(3 \times 0.2) + (4 \times -0.1) & (3 \times -0.2) + (4 \times 0.6)\end{bmatrix}$$

$$= \begin{bmatrix}0 & 1\\0.2 & 1.8\end{bmatrix}$$

Here's the Python code to calculate this:

In [9]:

```python
import numpy as np

A = np.array([[1,2],
              [3,4]])

B = np.array([[6,2],
              [1,2]])


C = A @ np.linalg.inv(B)

print(C)
```

```
[[0.  1. ]
 [0.2 1.8]]
```

Solving Systems of Equations with Matrices

One of the great things about matrices, is that they can help us solve systems of equations. For example, consider the following system of equations:

$$\begin{aligned} 2x + 4y &= 18 \\ 6x + 2y &= 34 \end{aligned}$$

We can write this in matrix form, like this:

$$\begin{bmatrix} 2 & 4 \\ 6 & 2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 18 \\ 34 \end{bmatrix}$$

Note that the variables (***x*** and ***y***) are arranged as a column in one matrix, which is multiplied by a matrix containing the coefficients to produce as matrix containing the results. If you calculate the dot product on the left side, you can see clearly that this represents the original equations:

$$\begin{bmatrix} 2x + 4y \\ 6x + 2y \end{bmatrix} = \begin{bmatrix} 18 \\ 34 \end{bmatrix}$$

Now. let's name our matrices so we can better understand what comes next:

$$A = \begin{bmatrix} 2 & 4 \\ 6 & 2 \end{bmatrix} \quad X = \begin{bmatrix} x \\ y \end{bmatrix} \quad B = \begin{bmatrix} 18 \\ 34 \end{bmatrix}$$

We already know that ***A • X = B***, which arithmetically means that ***X = B ÷ A***. Since we can't actually divide by a matrix, we need to multiply by the inverse; so we can find the values for our variables (*X*) like this: ***X = A^{-1} • B***

So, first we need the inverse of A:

$$\begin{aligned} \begin{bmatrix} 2 & 4 \\ 6 & 2 \end{bmatrix}^{-1} &= \frac{1}{(2 \times 2) - (4 \times 6)} \begin{bmatrix} 2 & -4 \\ -6 & 2 \end{bmatrix} \\ &= \frac{1}{-20} \begin{bmatrix} 2 & -4 \\ -6 & 2 \end{bmatrix} \\ &= \begin{bmatrix} -0.1 & 0.2 \\ 0.3 & -0.1 \end{bmatrix} \end{aligned}$$

Then we just multiply this with B:

$$X = \begin{bmatrix} -0.1 & 0.2 \\ 0.3 & -0.1 \end{bmatrix} \cdot \begin{bmatrix} 18 \\ 34 \end{bmatrix}$$

$$X = \begin{bmatrix} (-0.1 \times 18) + (0.2 \times 34) \\ (0.3 \times 18) + (-0.1 \times 34) \end{bmatrix}$$

$$X = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

The resulting matrix (*X*) contains the values for our *x* and *y* variables, and we can check these by plugging them into the original equations:

$$\begin{aligned} (2 \times 5) + (4 \times 2) &= 18 \\ (6 \times 5) + (2 \times 2) &= 34 \end{aligned}$$

These of course simplify to:

$$10 + 8 = 18$$

In [10]:
```python
import numpy as np

A = np.array([[2,4],
              [6,2]])

B = np.array([[18],
              [34]])

C = np.linalg.inv(A) @ B

print(C)
```

```
[[5.]
 [2.]]
```