Solving Systems of Linear Equations with NumPy - Code Along

Introduction

In this lesson, you'll learn how to solve a system of linear equations using matrix algebra and Numpy. You'll also learn about the identity matrix and inverse matrices, which have some unique properties that can be used to solve for unknown values in systems of linear equations. You'll also learn how to create these using Numpy.

Objectives

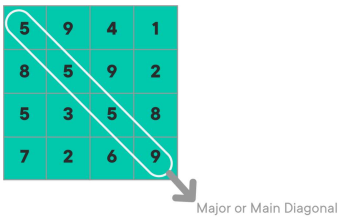
You will be able to:

- Define the identity matrix and its dot product
- Define the inverse of a matrix
- Calculate the inverse of a matrix in order to solve linear problems
- Use matrix algebra and Numpy to solve a system of linear equations given a real-life example

Identity matrix

An identity matrix is a matrix whose dot product with another matrix M equals the same matrix M.

The identity matrix is a square matrix which contains **1**s along the major diagonal (from the top left to the bottom right), while all its other entries are **0**s. The main diagonal is highlighted in the image below:



An identity matrix with the same (3×3)-shape contains all **1**s along this diagonal and **0**s everywhere else as shown below:

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

This would be called a (3×3) identity matrix. The $(n \times n)$ identity matrix is usually denoted by I_n which is a matrix with n rows and n columns.

The identity matrix is also called the *unit matrix* or *elementary matrix*.

Dot product of a matrix and its identity matrix

Let's try to multiply a matrix with its identity matrix and check the output. Let's start with the coefficient matrix from the previous problem:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

The identity matrix for this matrix would look like:

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

The dot product for these two matrices can be calculated as:

```
import numpy as np
A = np.array([[2,1],[3,4]])
I = np.array([[1,0],[0,1]])
print(I.dot(A))
print('\n', A.dot(I))
```

In [11]:
```
# Code here
```

You see that the dot product of any matrix and the appropriate identity matrix is always the original matrix, regardless of the order in which the multiplication was performed! In other words,

$$A \cdot I = I \cdot A = A$$

NumPy comes with a built-in function `np.identity()` to create an identity matrix. Just pass in the dimension (number of rows or columns) as the argument. You can add an argument `dtype=int` to make sure the elements are integers (if not, your identity matrix will contain floats):

```
print(np.identity(4, dtype=int))
print(np.identity(5, dtype=int))
```

In [12]:
```
# Code here
```

Inverse matrix

The *inverse* of a square matrix *A*, sometimes called a *reciprocal matrix*, is a matrix A^{-1}such that

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

where I is the identity matrix

The inverse of a matrix is analogous to taking reciprocal of a number and multiplying by itself to get a 1, e.g. $5 * 5^{-1} = 1$. Let's see how to get inverse of a matrix in NumPy. `numpy.linalg.inv(a)` takes in a matrix *A* and calculates its inverse as shown below:

```
A = np.array([[4, 2, 1],[4, 8, 3],[1, 1, 0]])
A_inv = np.linalg.inv(A)
print(A_inv)
```

In [18]:
```
# Code here
```

This is great. So according to the principle shown above, if we multiply A with A^{-1}, we should get an identity matrix I as the output:

```
A_product = np.dot(A, A_inv)
A_product
```

In [19]:
```
# Code here
```

Note that the expected output was an identity matrix. Although you have **1**s along the major diagonal, the float operations returned not zeros but numbers very close to zero off-diagonal. Numpy has a `np.matrix.round()` function to convert each element of the above matrix into a decimal form.

```
np.matrix.round(A_product)
```

In [20]:
```
# Code here
```

This looks more like the identity matrix that we saw earlier. The negative signs remain after rounding off as the original small values were negative. This, however, won't affect computation in any way.

Why do we need an inverse?

You need an inverse because you can't perform division operations with matrices! **There is no concept of dividing by a matrix**. However, you can multiply by an inverse, which achieves the same thing.

Imagine you want to share 10 apples with 2 people.

You can divide 10 by 2, or you can take the reciprocal of 2 (which is 0.5), so the answer is:

$10 \times 0.5 = 5$ - which means they get 5 apples each.

We use the very same idea here and this can be used to solve a system of linear equation in the problems we saw earlier in the section where:

$A \cdot X = B$ (remember A is the matrix of coefficients, X is the unknown variable and B is the output)

Say you want to find matrix X, when you already know matrices A and B:

It would've been great if you could divide both sides by A to get $X = B/A$, but remember that you can't divide. You can obtain this if you multiply both sides by A^{-1}, as shown below:

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

From above, we that A . A^{-1} = I, so:

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

We can remove I (because multiplying with the identity matrix doesn't change a matrix). so:

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

And there we have it, our answer.

Solve a system of equations with matrix algebra

Now that you know everything about converting a simple real world problem into matrix format, and steps to solve the problem, let's try it out with the apples and bananas problem:

Let's say you go to a market and buy 2 apples and 1 banana. For this you end up paying 35 pence. If you denote apples by a and bananas by b, the relationship between bought items bought and price paid can be written down as:

$2a + b = 35$ - (Eq. A)

In your next trip to the market, you buy 3 apples and 4 bananas, and the cost is 65 pence:

$3a + 4b = 65$ - (Eq. B)

As seen before, this is what that looks like in matrix notation:

$$\begin{pmatrix} 2 & 1 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 35 \\ 65 \end{pmatrix}$$

So first we'll need to calculate the inverse of the square matrix containing coefficient values:

```
# Define A and B
A = np.matrix([[2, 1], [3, 4]])
B = np.matrix([35, 65])

# Take the inverse of Matrix A
A_inv = np.linalg.inv(A)
A_inv
```

In [13]:
```
# Code here
```

You can now take a dot product of `A_inv` and `B`. Also, as you want the output in the vector format (containing one column and two rows), you would need to transpose the matrix `B` to satisfy the multiplication rule you saw previously.

> **The product of an $M \times N$ matrix and an $N \times K$ matrix is an $M \times K$ matrix. The new matrix takes the number of rows from the first matrix and the number of columns from the second matrix**

```
# Check the shape of B before after transposing
print(B.shape)
B = B.T
print (B.shape)
B
```

In [14]:
```
# Code here
```

Now, you can easily calculate X as below:

```
X = A_inv.dot(B)
X
```

In [15]:
```
# Code here
```

You can see that the prices of apples and bananas have been calculated as 15p per apple and 5p per banana, and these values satisfy both equations. Great!

The dot product of A and X should give matrix B. Let's try it:

```
print(A.dot(X))
print(B)
```

In [16]:
```
# Code here
```

Success!

You can also use `numpy.linalg.solve()` to solve a system of linear equations!

Numpy has a built-in function to solve such equations as `numpy.linalg.solve(a, b)` which takes in matrices in the correct orientation, and gives the answer by calculating the inverse. Here is how to use it.

```
# Use Numpy's built in function solve() to solve linear equations
x = np.linalg.solve(A, B)
x
```

In [17]:
```
# Code here
```

Further Reading

- Youtube: Solving System of Linear Equations using Python (https://youtu.be/AqIrdW2-K6k)
- Inverse of a matrix (http://www.mathwords.com/i/inverse_of_a_matrix.htm)
- Don't invert that matrix (https://www.johndcook.com/blog/2010/01/19/dont-invert-that-matrix/)

Summary

In this lesson, you learned how to calculate the inverse of a matrix in order to solve a system of linear equations. You applied the skills learned on the apples and bananas problem introduced earlier. The result of the calculations helped us get unit values of variables that satisfy both equations. In the next lab, you'll go through some other similar problems.