Workshop 4: Matrices and Vectors

In [1]:

```
from sympy import *
%matplotlib inline
init_printing()
```

In [2]:

```
x,y,z = symbols('x y z')
a,b,c,d= symbols('a b c d')
```

1. A note on names

Something some of you have run into already is that Python's names for functions and so on do not have to be single letters. For example:

In [3]:

```
expr = x**2 + 3*x + 2
expr
```

Out[3]:

$$x^2 + 3x + 2x^2 + 3x + 2$$

We can then manipulate this in the normal way:

In [4]:

```
expr.subs(x,-2)
```

Out[4]:

$$00$$

We shall use names with more than one character this week as we are meeting lots of different *data structures* (ways to store and represent data) and it helps to use names which are clear.

2. Packages

We have already seen in the first workshop how to import the SymPy package into Python. As mentioned before, this package will provide nearly everything we will need for these workshops. However, like LATEX, Python offers a wide range of other useful packages, all of which can be used simultaneously when set up correctly.

We saw how to import the entire SymPy package (using *), but if you know what you are looking for, you can choose to only import specific functions and commands. For example:

In [5]:

```
from sympy import integrate
```

This is a good idea if you plan on using just one or two things from a package - especially if you are importing multiple packages.

If you require multiple packages but also a lot from each package, it can be sensible to import each package 'as' something. For example:

In [6]:

```
import math as something
```

Once you have imported a package as something, you must then call everything in the package using the name you imported the package as. So for example:

In [7]:

```
something.cos(1)
```

Out[7]:

$$0.54030230586813980.5403023058681398$$

In [8]:

```
something.log(1)
```

Out[8]:

$$0.00.0$$

This allows you to keep tabs on what you are using from each package, whilst also avoiding conflict between packages (for example, two packages may contain functions with the same name).

3. Defining matrices and vectors

3.1. Making lists

Python has the notion of a *list* (of numbers, parameters, variables etc.). It stores these as lists in *square brackets*. You can assign names to lists:

In [9]:

```
ListA = [5,2,1]
ListA
```

Out[9]:

$[5,\ 2,\ 1]$[5, 2, 1]

In [10]:

```
ListB = [4,5,7]
ListB
```

Out[10]:

$[4,\ 5,\ 7]$[4, 5, 7]

You can manipulate lists in Python in many ways. We will only have time to cover a few of these over the following workshops. A simple example is adding two lists together:

In [11]:

```
ListA + ListB
```

Out[11]:

$[5,\ 2,\ 1,\ 4,\ 5,\ 7]$[5, 2, 1, 4, 5, 7]

Feel free to try making and manipulating your own lists. Python is able to handle many kinds of lists. A few of these can be seen below:

In [12]:

```
ListC = ['cat', 'dog', 'mouse']
ListC
```

Out[12]:

```
['cat', 'dog', 'mouse']
```

In [13]:

```
ListD = [3.4, 7, 'hello', 2.63, [8,9,3],]
ListD
```

Out[13]:

```
[3.4, 7, 'hello', 2.63, [8, 9, 3]]
```

Python can even handle lists that include SymPy objects (assuming they have been appropriately declared beforehand):

In [14]:

```
ListE = [1,x,x**2]
ListE
```

Out[14]:

$$\left[1,\ x,\ x^{2}\right]\left[1,\ x,\ x^{2}\right]$$

And make functions whose values are lists. There will be more on functions in **Workshop 8 (Advanced)**.

3.2. Using lists to define matrices and vectors

SymPy has a few different ways to define a matrix. Perhaps the most natural way to define a matrix using SymPy is to provide a list of lists, where each list corresponds to a row of the matrix.

A more advanced method of defining a matrix in SymPy is to provide the dimensions of the desired matrix, and then a single list of values.

Whichever method you choose to use, you will need to use the SymPy object `Matrix`.

Let us make the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ using SymPy.

Method 1 (list of lists)**:**

In [15]:

```
Matrix([[1,2,3],[4,5,6]])
```

Out[15]:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Method 2 (single list)**:**

In [16]:

```
Matrix(2,3,[1,2,3,4,5,6])
```

Out[16]:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Note: Since Method 1 is a *list* of lists, it is important to enclose all lists in square brackets.

To make a vector in SymPy, we still make use of the Matrix object, but we simply provide a single list:

In [17]:

```
Matrix([3,1,6,2,7])
```

Out[17]:

$$\begin{bmatrix} 3 \\ 1 \\ 6 \\ 2 \\ 7 \end{bmatrix}$$

4. Accessing matrix cells

To access an individual row or column of a matrix, use `row(.)` or `col(.)`, with the desired row or column number.

Warning: Be aware that Python begins counting from 0, not 1. So to access the second row of a matrix, we call `row(1)`.

To access an individual matrix entry (or cell), as opposed to an entire row or column, combine `row(.)` and `col(.)`. Below are some examples.

In [18]:

```
M = Matrix([[1,2,3],[x,y,z],[a,b,c]])
M
```

Out[18]:

$$\begin{bmatrix} 1 & 2 & 3 \\ x & y & z \\ a & b & c \end{bmatrix}$$

In [19]:

```
M.row(0)
```

Out[19]:

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

In [20]:

```
M.col(2)
```

Out[20]:

$$\begin{bmatrix} 3 \\ z \\ c \end{bmatrix}$$

In [21]:

```
M.row(2).col(1)
```

Out[21]:

$$\begin{bmatrix} b \end{bmatrix}$$

5. Matrix methods

Basic matrix operations such as addition and multiplication can be carried out using the usual operators: +, *, and **.

Warning: To avoid errors, always check that the matrices you are attempting to operate on have sensible dimensions. Clearly you do not want to try and add a 2x2 matrix to a 6x3 matrix, for example.

Below we have some examples of basic matrix operations. First we define some matrices to operate on:

In [22]:

```
P = Matrix([[4,2],[3,7]])
Q = Matrix([[5,3],[9,1]])
```

Adding and multiplying matrices is fairly straightforward:

In [23]:

```
P + Q
```

Out[23]:

$$\begin{bmatrix} 9 & 5 \\ 12 & 8 \end{bmatrix}$$

Recall that the order of multiplication is important:

In [24]:

```
P*Q
```

Out[24]:

$$\begin{bmatrix} 38 & 14 \\ 78 & 16 \end{bmatrix}$$

In [25]:

```
Q*P
```

Out[25]:

$$\begin{bmatrix} 29 & 31 \\ 39 & 25 \end{bmatrix}$$

In [26]:

```
P*P
```

Out[26]:

$$\begin{bmatrix} 22 & 22 \\ 33 & 55 \end{bmatrix}$$

We can multiply matrices by themselves by raising them to a power:

In [27]:

```
P**2
```

Out[27]:

$$\begin{bmatrix} 22 & 22 \\ 33 & 55 \end{bmatrix}$$

In [28]:

```
P**5
```

Out[28]:

$$\begin{bmatrix} 9922 & 14278 \\ 21417 & 31339 \end{bmatrix}$$

This is also how we find the inverse of matrix (when possible):

In [29]:

```
P**-1
```

Out[29]:

$$\begin{bmatrix} \frac{7}{22} & -\frac{1}{11} \\ -\frac{3}{22} & \frac{2}{11} \end{bmatrix}$$

In the space below, define X and Y as the following matrices, and find both XY and YX:

$$X = \begin{bmatrix} 3 & 7 & 2 & 4 \\ 2 & 4 & 4 & 3 \end{bmatrix}, Y = \begin{bmatrix} 4 & 5 \\ 9 & 3 \\ 6 & 6 \\ 3 & 2 \end{bmatrix}$$

In [33]:

```
X = Matrix([[3,7,2,4],[2,4,4,3]])
Y = Matrix([[4,5],[9,3],[6,6],[3,2]])
```

In [34]:

```
X*Y
```

Out[34]:

$$\begin{bmatrix} 99 & 56 \\ 77 & 52 \end{bmatrix}$$

In [35]:

```
Y*X
```

Out[35]:

$$\begin{bmatrix} 22 & 48 & 28 & 31 \\ 33 & 75 & 30 & 45 \\ 30 & 66 & 36 & 42 \\ 13 & 29 & 14 & 18 \end{bmatrix}$$

Find $(YX)^2 + 5YX$:

In [36]:

```
(Y*X)**2 + 5*Y*X
```

Out[36]:

$$\begin{bmatrix} 3421 & 7643 & 3638 & 4731 \\ 4851 & 10869 & 5034 & 6693 \\ 4614 & 10314 & 4884 & 6378 \\ 1962 & 4390 & 2060 & 2710 \end{bmatrix}$$

Find $(XY)^{-1}$, but do **not** attempt to find $(YX)^{-1}$:

In [39]:

```
(X*Y)**-1
```

Out[39]:

$$\begin{bmatrix} \frac{13}{209} & -\frac{14}{209} \\ -\frac{7}{76} & \frac{9}{76} \end{bmatrix}$$

6. A more complicated example

Now that we have the seen the basics, let us run through a slightly more complicated example that should have greater relevance to your course. Suppose we have been presented with the question below. We will work through it step-by-step.

Question:

Define the matrices $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, and $B = \begin{bmatrix} 2 & -1 \\ 3 & 5 \end{bmatrix}$.

Use A and B to find expressions for the coeffients a to d such that $AB = BA$.

Hint: solve the equation $AB - BA = 0$ for a to d. You may wish to look up `solve` in the SymPy documentation to find out how to use it for simultaneous equations.

Solution:

First we define A and B, and check to see if we get some sensible output for the products AB and BA.

In [40]:

```
A = Matrix([[a,b],[c,d]])
B = Matrix([[2,-1],[3,5]])
A*B
```

Out[40]:

$$\begin{bmatrix} 2a+3b & -a+5b \\ 2c+3d & -c+5d \end{bmatrix}$$

In [41]:

```
B*A
```

Out[41]:

$$\begin{bmatrix} 2a-c & 2b-d \\ 3a+5c & 3b+5d \end{bmatrix}$$

Using the hint, we declare a new matrix $C = AB - BA$.

In [42]:

```
C = A*B-B*A
C
```

Out[42]:

$$\begin{bmatrix} 3b+c & -a+3b+d \\ -3a-3c+3d & -3b-c \end{bmatrix}$$

Now we want to set each element of the matrix to 0, and solve. To do this, we're goiing to need to access each element individually. Let's call the top-left c_0, top-right c_1, bottom-left c_2 etc. Then to access each cell we could try:

In [43]:

```
c0 = C.row(0).col(0)
c0
```

Out[43]:

$$\begin{bmatrix} 3b+c \end{bmatrix}$$

But note that this returns a 1x1 matrix (or a 1 element list). To get around this we instead use:

In [44]:

```
c0 = C.row(0).col(0)[0]
c0
```

Out[44]:

$3b + c$

Note: Don't worry about what `[0]` does at the moment. We will discuss how to access lists in more detail in later workshops.

And so, doing the same for the other elements we have:

In [45]:

```
c1 = C.row(0).col(1)[0]
c2 = C.row(1).col(0)[0]
c3 = C.row(1).col(1)[0]
```

We are now in a position to solve the above as a set of simultaneous equations.

Recall that when using `solve` for a single equation, the first argument should be an equation, and the second argument should tell SymPy what to solve the equation with respect to. For a set of simultaneous equations, it is sufficient to pass just one argument - a single list of the simultaneous equations.

Additionally, recall from workshop 2 that, SymPy will automatically assume c_0, c_1, c_2 and c_3 are equations set to equal 0 by default (unless otherwise explicitly stated). Thus we solve to reach our solution using:

In [46]:

```
solve([c0,c1,c2,c3])
```

Out[46]:

$$\left\{a : -c + d,\ b : -\frac{c}{3}\right\}$$

In [ ]:

```

```