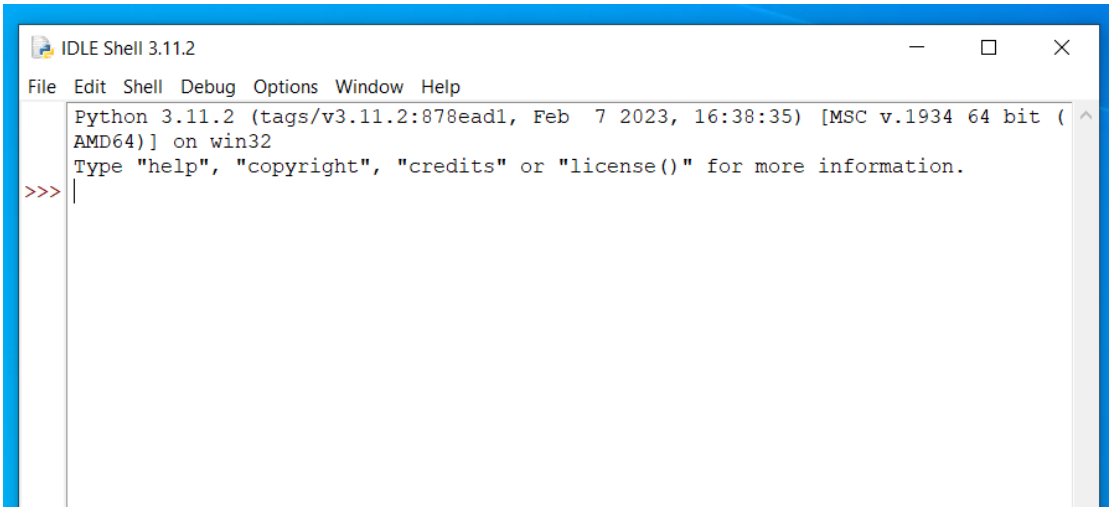
Introduction to Linear Algebra/ Using Python/ Matrices using Python

Matrix methods using Python

Matrices come into their own if they get big, in which case you are much better off using technology. These instructions explain how to work with matrices in Python.

Importing mathematical functionality

Run IDLE, so you are looking at a window like this.



At this stage, we are going to type commands into the IDLE Shell, rather than writing a program.

To import the mathematical functions we need, run these two commands:

```
from sympy import *
```

and

```
init_printing(use_unicode=True)
```

The second line asks Python to make the output a bit prettier, which can be a help when dealing with matrices.

Matrix operations

Inputting a matrix

To input a matrix

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

use this command in Python:

```
M = Matrix([[1,2],[4,3]])
```

This breaks down like this:

- The matrix is called $\mathbf{M}$, so we start `M =`.
- We tell Python we are entering a matrix using `Matrix()`.
- Inside this, the whole matrix is in a set of square brackets `[...]`.
- Inside *this*, each row is within its own pair of square brackets `[...]`, with a comma between each entry and between each row.

It's quite fiddly, but if you can keep track of the brackets that'd be good. If you want to see whether you've entered it correctly, type `M` and press enter and it will display the rows and columns for you to check.

For example, to input the matrices

$$\mathbf{A} = \begin{bmatrix} -8 & -5 & -4 & 2 & 1 \\ 2 & 7 & 2 & -1 & 1 \\ 4 & 1 & -6 & -3 & 3 \\ 0 & -1 & 1 & 11 & 3 \\ 1 & 1 & -1 & 0 & -4 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 9 & -2 & 0 & -3 \\ -3 & -10 & -7 & -2 & 0 \\ -11 & 12 & 13 & 0 & 11 \\ -7 & 10 & 7 & -7 & 12 \\ 8 & -9 & 11 & 4 & 0 \end{bmatrix}$$

We use

```
A = Matrix([[-8,-5,-4,2,1],[2,7,2,-1,1],[4,1,-6,-3,3],[0,-1,1,11,3],[1,1,-1,0,
B = Matrix([[0,9,-2,0,-3],[-3,-10,-7,-2,0],[-11,12,13,0,11],[-7,10,7,-7,12],[8
```

Matrix addition

Python will store these two matrices as `A` and `B`, and we can ask for them by those names. So if we want to add $\mathbf{A} + \mathbf{B}$, we do this using

```
A+B
```

Multiply a matrix by a scalar

To multiply a matrix by a scalar, simply use the number and `*` for multiplication. For example, to calculate $4\mathbf{A}$ use

```
4*A
```

Matrix multiplication

To multiply two matrices together, again we use `*` for multiplication. To calculate $\mathbf{AB}$ we use

```
A*B
```

Transpose

To calculate $\mathbf{A}^{\mathsf{T}}$, use

```
A.T
```

Powers

To raise a matrix to a power, note that Python uses `**` and not `^`. For example, to calculate $\mathbf{A}^5$ use

```
A**5
```

Identity matrix

If you need an identity matrix, you can use a Python command `eye`, so for example to get a 5×5 identity matrix use

```
I = eye(5)
```

Zero matrix

You can get a matrix of all zeros using `zeros`. For example, a 10×10 zero matrix is given by

```
Z = zeros(10,10)
```

Exact values and decimal approximations

If you are entering a matrix and want to use exact values for, say, fractions, you can do this using `S()`. For example if I enter

```
C = Matrix([[0,1/2,1/2,1/3],[1,0,1/2,1/3],[0,0,0,1/3],[0,1/2,0,0]])
```

and ask Python to tell me what `A` is, I get this:

```
>>> C
C
⎡0  0.5  0.5  0.333333333333333⎤
⎢                              ⎥
⎢1   0   0.5  0.333333333333333⎥
⎢                              ⎥
⎢0   0    0   0.333333333333333⎥
⎢                              ⎥
⎣0  0.5   0           0        ⎦
```

Instead of using the exact values, Python has converted these into decimals. This can cause small rounding errors that can introduce small errors into the final result.

We can enclose the entire `Matrix()` command inside an `S()`, like this:

```
C = S('Matrix([[0,1/2,1/2,1/3],[1,0,1/2,1/3],[0,0,0,1/3],[0,1/2,0,0]])')
```

Now when we ask to see `A`, we get fractions rather than decimals, meaning Python is treating these as exact (symbolic) values.

```
>>> C
C
⎡0  1/2  1/2  1/3⎤
⎢                ⎥
⎢1   0   1/2  1/3⎥
⎢                ⎥
⎢0   0    0   1/3⎥
⎢                ⎥
⎣0  1/2   0    0 ⎦
```

If Python is displaying fractions but you would like decimal values, you can use `.evalf()`.

For example, to view the matrix `C` above with all values given as decimal approximation, run

```
C.evalf()
```

Referring to matrix elements

If you need to use a matrix element for further calculation, it is a very bad idea to copy and paste a number that may be truncated. Instead, you can refer to elements in a matrix directly. Python counts from `0` not `1`, so to access the first element on the first row use

```
A[0,0]
```

To get the 5th row, 3rd colum, use

```
A[4,2]
```

You can also do this on matrices you have worked on, for example to get the element on the 2nd row, 3rd column of the result of AB, use

```
(A*B)[1,2]
```