SymPy

SymPy is a symbolic manipulation module for Python. We have covered it before (starting with 'Maths in the shell' in week 1), but here are some notes including some examples of how it works for different areas of maths.

Powers

An important thing to remember is how Python handles powers. Remember that x^a is coded as `x**a`. Some other computer systems use `^` rather than `**`, so you may find yourself using `^` by accident and getting unexpected results.

Importing SymPy

The commands in SymPy are not automatically available to Python, we must tell Python which commands we want to use. Here we import the commands `diff` and `cos`, and define a variable `x`, then use them to print the value of $\frac{\mathrm{d}(\cos(x))}{\mathrm{d}x}$. (There's more on `diff`, `cos` and `sympy.abc` below.)

```
from sympy import diff, cos
from sympy.abc import x

print(diff(cos(x),x))
```

If you are going to use a lot of SymPy commands in the same program, you might prefer to import the whole SymPy module rather than listing which commands you plan to use.

```
from sympy import *
```

It is up to you which you choose to use. One issue with importing a whole package (`*`) is if that package includes a command name that is already used in Python. For example, if you do `from math import *` and `from sympy import *` you will end up with duplicate commands for e.g. `sin`, `cos`, `tan`, `sqrt`, etc. and might not be sure which you are using in your code.

Declaring symbols using `sympy.abc`

Python doesn't naturally know what to do with mathematical variables, so you need to tell it what these are. For example, here we tell Python to treat the letters `x`, `y` and `n` as mathematical variables, then we can use them mathematically.

```
from sympy.abc import x, y, n
```

Without this line, we would get an error because we haven't given values to `x`, `y` and `n`.

Once you have told SymPy to handle the variable names, you can refer to mathematical expressions in code. Specifically, you don't need to put mathematical expressions in quotes like they were a string.

For example, here we expand $(x^9-2)^4(x^4+x^2+7)$.

```
from sympy import expand
from sympy.abc import x

# expand (note we don't need "quotes")
print(expand((x**9-2)**4*(x**4+x**2+7)))
```

Calculus

Let's do a bit of calculus.

Differentiation

Here we differentiate $\cos^3(t)+\sin^2(t)$ with respect to t.

```
from sympy import diff, cos, sin
from sympy.abc import t
```

```
print(diff(cos(t)**3 + sin(t)**2,t))
```

Here is a more complicated example using `solve` to find the stationary points of a function

$$f(x) = 25x^6 + 78x^5 - 45x^4$$

First we define `f` as the function above. Then we use `solve` to find the x coordinates of the stationary points by solving $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$. Finally, we use a `for` loop to print out the x coordinates of the stationary points and the value of $f(x)$ at this point.

```
from sympy import solve, diff
from sympy.abc import x

f = 25 * x**6 + 78 * x**5 - 45 * x**4
stat_points = solve(diff(f,x))

for x_value in stat_points:
    print (x_value, f.subs(x,x_value))
```

Integration

Here we evaluate $\int t^2 \cos(t)\,\mathrm{d}t$.

```
from sympy import integrate, cos
from sympy.abc import t

print(integrate(t**2 * cos(t),t))
```

To solve a definite integral, we pass a list that includes the variable, the lower limit and the upper limit. Here we compute

$$\int_{2\pi}^{9\pi/4} t^2 \cos(t)\,\mathrm{d}t.$$

```
from sympy import integrate, cos, pi
from sympy.abc import t

print(integrate(t**2 * cos(t),[t,2*pi,9/4*pi]).evalf())
```

Matrices

Inputting a matrix involves a lot of brackets. The command `Matrix` uses double brackets, then each row of the matrix is enclosed in square brackets. Rows are separated by a comma, as are items within each row.

Here we define the matrix

$$\begin{bmatrix} 4 & -21 & 9 & 1 \\ 6 & 15 & 7 & -1 \\ 2 & 89 & 19 & 2 \end{bmatrix}$$

```
from sympy import Matrix

M = Matrix(( [4, -21, 9, 1], [6, 15, 7, -1], [2, 89, 19, 2] ))
print(M)
```

We can do operations on matrices as you might expect. Here we compute

$$\begin{bmatrix} 1 & -4 \\ 7 & 1 \end{bmatrix} \begin{bmatrix} 6 & -3 \\ -4 & 1 \end{bmatrix}$$

```
from sympy import Matrix

print(Matrix([[1,-4],[7,1]]) * Matrix([[6,-3],[-4,1]]))
```

There are also special commands that act on matrices. For example, here we find the determinant and inverse of

$$\begin{bmatrix} 1 & 2 \\ -3 & 47 \end{bmatrix}$$

```
from sympy import Matrix

M = Matrix(( [1, 2], [-3, 47] ))
```

```
print(M.det())
print(M.inv())
```

There is a lot more on using Python for matrices in the linear algebra notes.

Using SymPy in program code

An advantage of having a symbolic manipulation package inside Python is that we can do mathematical operations alongside other programming constructs.

For example, here we use Python to compute $\int (x+3)^k \, \mathrm{d}x$ for $k \in \{0, \ldots 10\}$ using a loop. The loop has counter `i` and we use `f.subs(k, i)` to substitute the current value of `i` in place of the variable `k`.

```
from sympy import integrate
from sympy.abc import f, k, x

f = (x+3)**k
for i in range(0,11):
    print(integrate(f.subs(k, i),x))
```