Symbolic computation

Required packages: Sympy

Install using:

Pip install sympy

Or

Install using your IDE

Purpose:

Symbolic computation allows you to work using symbols, i.e, x, y, z directly as opposed to working with numbers.

Sympy is a very powerful symbolic computation package capable of manipulating equations, substitution, differentiation, integration, and much more.

Getting started:

Try the following code:

```
import sympy as sym

#defines x as a symbol.
x = sym.symbols('x')

#defines an equation in x, the symbol we defined before.
eq = 2*x + 5
print(eq)
```

It is important to distinguish between similarly named functions in sympy and numpy (e.g.: trigonometric functions, pi) so make sure you import sympy as sym.

The code given defines x as a symbol and defines an equation in x.

Substitution:

The code given below substitutes x = 1 in the given equation:

```
#substitutes x = 1 in equation defined earlier.
print(eq.subs(x,1))
```

Multiple variables:

The following code snippet defined multiple variables and uses them in an equation:

```
#define multiple symbols
x,y,z = sym.symbols('x,y,z')
eq1 = 2*x + 3*y + 4*z
print(eq1)
```

It is also possible to substitute multiple variables at once:

```
#substitute x = y = z = 1
print(eq1.subs([(x,1),(y,1),(z,1)]))
```

Solving equations:

It is also possible to solve equations. The solve function solves eq = 0 for the symbol given.

```
#solve the first equation (2x + 5 = 0) for x
print(sym.solve(eq,x))

#solve (2x + 5 = 2) for x
print(sym.solve(eq - 2,x))
```

This can also be used to solve simultaneous equations:

```
#define simultaneous equations
#2x + 3y = 13
#3x - 5y = -9
eq_a = 2*x + 3*y - 13
eq_b = 3*x - 5*y + 9

#solve
print(sym.solve([eq_a,eq_b],(x,y)))
```

Differentiation:

Try the following code:

```
eq = 3*x**2 -5*x -2
print (eq)

#differentiate with respect to x
eq_diff = sym.diff(eq,x)
print(eq_diff)
```

This differentiates the equation with respect to x.

It is also possible to apply higher order differentiation. It can be done in one of two ways:

```
#2nd order differentiation
eq_diff_2ndOrder = sym.diff(eq,x,x)
print(eq_diff_2ndOrder)

#does the same thing, but more convenient for higher orders
eq_diff_2ndOrder = sym.diff(eq,x,2)
print(eq_diff_2ndOrder)
```

Next week: partial derivatives, trigonometric and logarithmic functions, combining with numpy and matplotlib