Series Expansion

In this lesson, we will learn about Taylor series and formal power series.

WE'LL COVER THE FOLLOWING

- Taylor series
  - Substituting values
- Formal power series
  - The real 'power' of the formal power series

Taylor series

SymPy expressions can be expanded as Taylor series using the `series()` function from the SymPy module.

Taylor series about $x = a$ is given by:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}a}{n!}(x-a)^n$$

where $f^{(n)}$ is the n^{th} derivative of f.

For a generic expansion, we only need to specify the expression and the variable in the expansion for which the expression is to be expanded.

```
series(f(x), x)
```

or

```
f(x).series(x)
```

Let's see a basic implementation of the `series()` function below:

```
from sympy import *

x = Symbol('x')


ts_sin = series(sin(x), x)      # using first syntax
ts_cos = cos(x).series(x)       # using second syntax
print("Taylor series for sin(x):", ts_sin)
print("Taylor series for cos(x):", ts_cos)
```

For numerical evaluations and plotting, we need to remove the trailing `O(n)` term. This can be done using the `removeO()` method:

```
from sympy import *

x = Symbol('x')

ts_sin = series(sin(x), x).removeO()
ts_cos = series(cos(x), x).removeO()
print("Taylor series for sin(x):", ts_sin)
print("Taylor series for cos(x):", ts_cos)
```

Additionally, we can also specify the point `a` around which we want to expand and the maximum term power, `maxTerm`:

```
series(f(x), x, a, maxTerm)
```

By default, the `series()` function expands the expression around 0, i.e `a=0`.

💡 Use the `help(Basic.series)` command for more information on `series()`.

```
from sympy import *

x = Symbol('x')

ts_sin = series(sin(x), x, 0.1, 5) # taylor series around pi/4 with max power 5
ts_cos = series(cos(x), x, 0.5, 4) #  taylor series around pi/3 with max power 4
print("Taylor series for sin(x):", ts_sin)
print("Taylor series for cos(x):", ts_cos)
```

Substituting values

Using `subs()`, we can substitute the value of variables, which is useful when plotting data.

`subs()` takes a dictionary as its input argument, where variables are the keys.

```
subs({x: 1, y: 4})
```

```
from sympy import *

x = Symbol('x')

s8 = sin(x).series(x, 0, 8).removeO()
print("Taylor series for sin(x):", s8.subs({x: pi/10}))
```

Formal power series

A more useful series that can be computed using SymPy is the formal power series using the `fps()` function.

```
fps(f(x), x, a)
```

`x` is the variable for which the series is computed and `a` is the variable about which the series is computed.

The advantage this has over the power series is that it returns the explicit formula for the coefficients of the series rather than just computing the first few terms:

$$\sum_{k=0}^{\infty} a_k x^k$$

The `fps()` function returns the formal series expansion of the function in the form of a `FormalPowerSeries` object. `fps()` is used in combination with its `truncate(n)` method which returns the first `n` terms of the series.

```
fps(f(x), x, a).truncate(n)
```

`removeO()` is used to remove the trailing `O(n)` term.

`subs()` is used to substitute the value of a variable.

Let's compute the formal power series of e^x:

```
from sympy import *

x = Symbol('x')

fps_exp = fps(exp(x), x, 0).truncate(6).removeO() # formal power series of exp around 0
print(fps_exp.subs({x: 2}))
```

The real 'power' of the formal power series

This may seem similar to the computation of the power series but let's see how the formal power series is more useful. Since we have the generic formula of the coefficients, we can extract out a single term as well. Suppose we want to extract the 15^{th} term from the expansion of e^x:

```
from sympy import *

x = Symbol('x')

fps_exp = fps(exp(x), x, 0) # formal power series of exp around 0
print(fps_exp[15])
```

In line 6, we used the index to extract the specific term that we needed.

The value of the index corresponds to the power of the variable for which we computed the series, that is, to get the coefficient of the 15^{th} value, we used the index 15.

In the next lesson, we will learn about solving equations in SymPy.