# Computing with formulas

## Foundation of programming (CK0030)

Francesco Corona



Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## FdP

- ☹ **Intro to variables, objects, modules, and text formatting**
- ☹ Programming with WHILE- and FOR-loops, and lists
- ☹ Functions and IF-ELSE tests

- ☹ Data reading and writing
- ☹ Error handling
- ☹ Making modules

- ☹ Arrays and array computing
- ☹ Plotting curves and surfaces

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

# Complex numbers

## Computing with formulas

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex numbers

Consider the second-order (algebraic) equation $x^2 = 2$

We know the solution

$$\rightsquigarrow \quad x = \pm\sqrt{2}$$

What if we are interested in an equation like $x^2 = -2$?

We need to define complex numbers

## Complex numbers (cont.)

**Definition**

*Complex numbers*

*A complex number is a pair of jointly written real numbers* $a$ *and* $b$

$$\rightsquigarrow \quad a + ib$$
$$\rightsquigarrow \quad a + bi$$

$i$ *is called the imaginary unit*

$$\rightsquigarrow \quad i = \sqrt{-1}$$



Computing with formulas

FC
CK0030
2018.1

Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

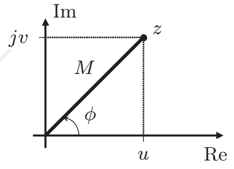
## Complex numbers (cont.)

**Complex numbers (Cartesian representation)**

Consider the set of complex numbers $\mathcal{C} = \{u + jv | u, v \in \mathcal{R}\}$ $(j = \sqrt{-1})$

A **complex number**

$$z = \text{Re}(z) + \text{Im}(z)$$
$$= u + jv$$




It consists of two parts

- **Real part**, $\text{Re}(z) = u$
- **Imaginary part**, $\text{Im}(z) = v$

The **complex conjugate** of $z$

$$z' = \text{Re}(z) - j\text{Im}(z)$$

Computing with formulas

FC
CK0030
2018.1

Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex numbers (cont.)

**Complex numbers (Polar representation)**

Consider the set of complex numbers $\mathcal{C} = \{u + jv | u, v \in \mathcal{R}\}$ $(j = \sqrt{-1})$

The **complex number** $z = \text{Re}(z) + \text{Im}(z) = u + jv$

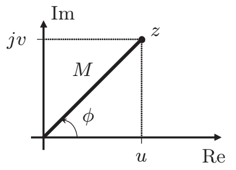
We can define

**Module**

- $M = |z| = \sqrt{u^2 + v^2}$

**Phase**

- $\phi = \arg(z) = \arctan(v/u)$




The inverse formula

- $u = M\cos(\phi)$
- $v = M\sin(\phi)$

Computing with formulas

FC
CK0030
2018.1

Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex numbers (cont.)

One very important *feature* of the set of complex numbers

The possibility to take square roots of negative numbers

$$\rightsquigarrow \quad \sqrt{-2} = \sqrt{2}i = \sqrt{2}\sqrt{-1}$$
$$\rightsquigarrow \quad \sqrt{-2} = \pm\sqrt{2}i$$

## Complex numbers (cont.)

Consider the two complex numbers $u = a + bi$ and $v = c + di$

We have,

$$
\begin{aligned}
u &= v \quad (\longrightarrow a = c, b = d) \\
-u &= -a - bi \\
u* &= a - bi \text{ (complex conjugate)} \\
u + v &= (a + c) + (b + d)i \\
u - v &= (a - c) + (b - d)i \\
uv &= (ac - bd) + (bc + ad)i \\
u/v &= \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad}{c^2 + d^2} i \\
|u| &= \sqrt{a^2 + b^2} \\
e^{iq} &= \cos(q) + i \sin(q)
\end{aligned}
$$



## Complex numbers (cont.)

Rules for addition/subtraction/multiplication/division of complex numbers

Rules for calculating transcendental functions of complex numbers

$$\sin(z), \cos(z), \tan(z), e^z, \ln(z), \sinh(z), \cosh(z), \tanh(z), \cdots$$

Rules for raising a complex number $z = a + ib$ to a real power



# Complex arithmetics

Complex numbers



## Complex arithmetics

The Python language supports computation with complex numbers

In Python the imaginary unit is `j` (the $i$ in mathematics)

A complex number $2 - 3i$ is expressed as `(2-3j)`

- Number $i$ is written as `1j`, not just `j`

We study the definition of complex numbers and some simple arithmetics

# Complex arithmetics

**Example**

```
>>> u = 2.5 + 3j                               # create a complex number
>>> v = 2                                      #       this is an integer
>>> w = u+v                                    #        complex + integer
>>> w
    (4.5+3j)


>>> a = -2                                     # from two floats
>>> b = 0.5                                    #
>>> s1 = a+b*1j                                #
>>> s1
    (-2+0.5j)


>>> s2 = complex(a,b)                          # alternative way
>>> s2
    (-2+0.5j)


>>> s*w                                        # complex*complex
    (-10.5-3.75j)


>>> s/w                                        # complex/complex
    (-0.25641025641025639+0.28205128205128205j)
```

`s` (`s1` and `s2`) is an object of type `complex`



# Complex arithmetics (cont.)

A complex object has functionalities for extracting real/imaginary parts

It is also possible to compute the complex conjugate

**Example**

```
>>> s = -2+0.5j

>>> s.real
    -2.0

>>> s.imag
    0.5

>>> s.conjugate()
    (-2-0.5j)
```



# Complex functions

## Complex numbers



# Complex functions

Computing the sine of a complex number requires some extra steps

**Example**

```
>>> from math import sin

>>> r = sin(w)
    Traceback (most recent call last):
     File "<input>", line 1, in ?
    TypeError: can't convert complex to float; use abs(z)
```

Function `sin` from `math module` only works with real (float) arguments

- Complex number are not valid arguments to its functions

## Complex functions

There exist a similar (mathematical) module, `cmath`

It contains functions that accept a complex number (object) as argument

⇝ The functions return a complex number (object)

Computing with formulas

FC
CK0030
2018.1

Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex functions (cont.)

**Example**

We want to show with Python that the following identity holds true

$$\sin(ai) = i\sinh(a),$$

for some scalar $a$ (say, $a = 8$)

We have,

```
>>> from cmath import sin, sinh


>>> r1 = sin(8j)
>>> r1
    1490.4788257895502j


>>> r2 = 1j*sinh(8)
>>> r2
    1490.4788257895502j
```

Computing with formulas

FC
CK0030
2018.1

Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex functions (cont.)

Mathematically, from the sine exponential formulation

$$\sin(ix) = i\frac{\left(e^{-i^2x} - e^{i^2x}\right)}{2} = i\frac{\left(e^{x} - e^{-x}\right)}{2} = i\sinh(x)$$

■

Computing with formulas

FC
CK0030
2018.1

Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex functions (cont.)

**Example**

We want to show with Python that the Euler's formula holds true

$$e^{iq} = \cos(q) + i\sin(q),$$

for some scalar $q$ (say, $q = 8$)

We have,

```
>>> from cmath import sin, cos, exp

>>> q = 8

>>> exp(1j*q)
    (-0.14550003380861354+0.98935824662338179j)

>>> cos(q) + 1j*sin(q)
    (-0.14550003380861354+0.98935824662338179j)
```

## Complex functions (cont.)

**The complex exponential function**

Consider an imaginary number $z = 0 + j\phi$

We have,

$$\rightsquigarrow \quad e^{j\phi} = \cos(\phi) + j\sin(\phi)$$

The exponential of an imaginary number is a complex number

- Real part, $\cos(\phi)$
- Imaginary part, $\sin(\phi)$

**Euler's formula**

Relationships to write a periodic function as a sum of exponential functions

$$\cos(\phi) = \frac{e^{j\phi} + e^{-j\phi}}{2}$$

$$\sin(\phi) = \frac{e^{j\phi} - e^{-j\phi}}{2j}$$



Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

# Complex and real functions

Complex numbers

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex and real functions

The functions in the `math module` do not accept complex numbers (objects)

The functions in the `cmath module` return complex numbers (objects)

It is useful to have smarter functions that return appropriate results

- $\rightsquigarrow$ A `complex object`, if the result is a complex number
- $\rightsquigarrow$ A `float object`, if the result is a real number

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex and real functions (cont.)

`NumPy` package has such versions of basic mathematical functions

- As those in the `math module` and in the `cmath module`

`NumPy` offers a unified treatment of real and complex functions

How to access these, more flexible, versions of basic mathematical functions

```
from numpy.lib.scimath import *
```

or

```
from scipy import *
```

or

```
from scitools.std import *
```

## Complex and real functions (cont.)

**Example**

Consider the problem of computing the square root of some numbers

Suppose that we want to use function `sqrt` in the `math module`

```
>>> from math import sqrt

>>> sqrt(4)                                  # float
    2.0

>>> sqrt(-1)                                 # illegal
    Traceback (most recent call last):
     File "<input>", line 1, in ?
    ValueError: math domain error

>>> help(sqrt)                               # help file
                                             #     ?sqrt
```

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex and real functions (cont.)

Suppose that we want to use function `sqrt` from the `cmath module`

```
>>> from cmath import sqrt

>>> sqrt(4)                                  # complex
    (2+0j)

>>> sqrt(-1)                                 # complex
    1j

>>> help(sqrt)                               # help file
                                             #     ?sqrt
```

Note that function `sqrt` from the `math module` is no longer available

- It has been overwritten by `sqrt` from the `cmath module`
- Function name `sqrt` is bounded to this new function

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex and real functions (cont.)

We can use, among other things, yet another version of the `sqrt` function

```
>>> from numpy.lib.scimath import *

>>> sqrt(4)                                  # float
    2.0

>>> sqrt(-1)                                 # complex
    1j

>>> help(sqrt)                               # help file
                                             #     ?sqrt
```

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex and real functions (cont.)

This last `sqrt` function is slower than the ones from `math` and `cmath`

It returns a `float object` if possible, or a `complex one`

- Yet, it is more flexible

## Complex and real functions (cont.)

**Example**

We further illustrate a flexible treatment of both complex and real numbers

We want to compute the roots of a quadratic function

$$f(x) = ax^2 + bx + c$$

for some constants $a$, $b$ and $c$

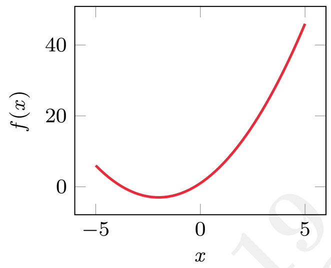
That is, we are interested in the values of $x$ such that $f(x) = 0$

$$\rightsquigarrow \quad x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex and real functions (cont.)




Let $a = 1$, $b = 4$ and $c = 1$

We have,

$$f(x) = x^2 + 4x + 1 = 0$$

```
>>> a = 1; b = 4; c = 1                              # polynomial coefficients

>>> from numpy.lib.scimath import sqrt                # import sqrt from numpy

>>> r1 = (-b + sqrt(b**2 - 4*a*c))/(2*a)                       # calculate roots
>>> r2 = (-b - sqrt(b**2 - 4*a*c))/(2*a)

>>> r1
    -0.267949192431

>>> r2
    -3.73205080757
```

The results, the roots, are two distinct `float objects`

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex and real functions (cont.)

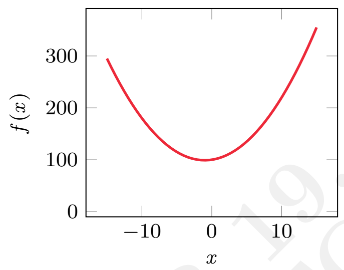
Using arrow-up (⇑), we can go back to the definitions of the coefficients

- ⇝ We can change them to be different numbers
- ⇝ Then, we can recompute `r1` and `r2`

MAR 19, 2018 — FC

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Complex and real functions (cont.)




Let $a = 1$, $b = 2$ and $c = 100$

We have,

$$f(x) = x^2 + 2x + 100 = 0$$

```
>>> a = 1; b = 2; c = 100                            # polynomial coefficients

>>> r1 = (-b + sqrt(b**2 - 4*a*c))/(2*a)
>>> r2 = (-b - sqrt(b**2 - 4*a*c))/(2*a)

>>> r1
    (-1+9.94987437107j)

>>> r2
    (-1-9.94987437107j)
```

## Complex and real functions (cont.)

Had we used `sqrt` from `cmath`, `r1` and `r2` would always be complex objects

With `sqrt` from `math`, we would not have been able to do the complex case

■





# Symbolic computing

Computing with formulas





## Symbolic computing

Python has a package `SymPy` for doing symbolic computing

- Equation solving, Taylor series expansion
- Symbolic integration and differentiation
- ...

Two options to perform interactive work with `SymPy`

- Conventional iterative shell, `ipython`
- Special interactive shell, `isympy`

`isympy` is installed along with `SymPy` itself





## Symbolic computing (cont.)

It is a good practice to explicitly import each symbol we need from `SymPy`

- It emphasises that those symbols come from that package

`sin` could mean the sine function from the `math module`

⇝ It is aimed only at real numbers

`sin` could mean the sine function from `SymPy`

⇝ It is aimed at symbolic expressions

# Differentiation and integration

## Symbolic computing

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Differentiation and integration

We are interested in differentiating a function w.r.t. an independent variable

- (or an expression like $y(t) = v_0 t - \frac{1}{2} g t^2$, with respect to $t$)

Suppose that we are then interested in integrating the answer

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Differentiation and integration (cont.)

**Example**

```
>>> from sympy import (
... symbols,      # define symbols for symbolic math
... diff,         # differentiate expressions
... integrate,    # integrate expressions
... Rational,     # define rational numbers
... lambdify,     # turn symbolic expressions into Python functions
... )             # ...

>>> t, v0, g = symbols('t v0 g')
>>> y = v0*t - Rational(1,2)*g*t**2

>>> dydt = diff(y,t)
>>> dydt
    -g*t + v0

>>> # 2nd derivative acceleration: -g
>>> print 'acceleration:', diff(y,t,t)

>>> y2 = integrate(dydt, t)
>>> y2
    -g*t**2/2 + t*v0
```

`t` (`v0`, `g`) is a `symbolic variable` (not a `float`, as in numerical computing)

`y` (`y2`, `y2`) is a `symbolic expression` (not a `float`)

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Differentiation and integration (cont.)

`symbolic expressions` can be turned into ordinary `numerical functions`

- The `lambdify` command of `SymPy`

**Example**

Take the `dydt` expression and turn it into a Python function `v(t,v0,g)`

```
>>> v = lambdify([t,v0,g],dydt)    # from symbolic to numerical function
>>> v(t=0,v0=5,g=9.81)             #              arguments in function v
    5

>>> v(2,5,9.81)
    -14.62

>>> 5 - 9.81*2                           # check the previous calculation
     -14.62
```

# Equation solving

## Symbolic computing

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Equation solving

Consider an equation defined through some expression $f$ (`f`) that is zero

The equation can be solved using `solve(f,x)`

- $x$ (`x`) is the unknown in the equation

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Equation solving (cont.)

**Example**

We want to find the roots of the expression

$$y = v_0 t - \frac{1}{2} g t^2 = 0$$

We have,

```
>>> from sympy import solve

>>> t, v0, g = symbols('t v0 g')
>>> y = v0*t - Rational(1,2)*g*t**2

>>> roots = solve(y,t)
>>> roots
    [0, 2*v0/g]
```

We can easily check the correctness of the answer

- We substitute the (values of the) roots in $y$

```
>>> y.subs(t, roots[0])
    0

>>> y.subs(t, roots[1])
    0
```

■

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Equation solving (cont.)

Substituting/inserting expressions (not values) `f2` for `f1` in expression `f`

⇝ We used `f.subs(f1,f2)`

# Taylor series and more

Symbolic computing

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Taylor series and more

Suppose that we are interested in computing the Taylor polynomial of order $n$ (`n`) of function $f$ (`f`) around some arbitrary point $t_0$ (`t0`)

- The independent variable $t$ (`t`)

The Taylor's polynomial can be computed using `f.series(t,t0,n)`

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Taylor series and more (cont.)

**Taylor expansion**

The Taylor series of a real- (complex-) function $f(x)$ that is infinitely differentiable at a real (complex) argument $x = a$ is the power-series

$$f(a) + f^{(1)}(a)(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots$$
$$= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$f^{(n)}(a)$ is the $n$-th derivative of $f(x)$ evaluated at point $x = a$

$n!$ is the factorial of $n$

$$n! = \prod_{k=1}^{n} k, \quad \text{(by definition, } n! = 1\text{, for } n = 0)$$

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Taylor series and more (cont.)

**Example**

Consider the two functions $e^t$ and $e^{\sin(t)}$

We are interested in their Taylor's expansion

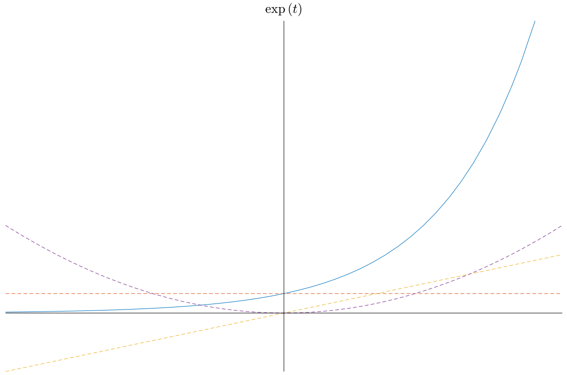
```
>>> from sympy import exp, sin

>>> f = exp(t)
>>> f.series(t,0,3)
    1+t+t**2/2+O(t**3)

>>> f = exp(sin(t))
>>> f.series(t,0,8)
    1 + t + t**2/2 - t**4/8 - t**5/15 - t**6/240 + t**7/90 + O(t**8)
```

## Taylor series and more (cont.)

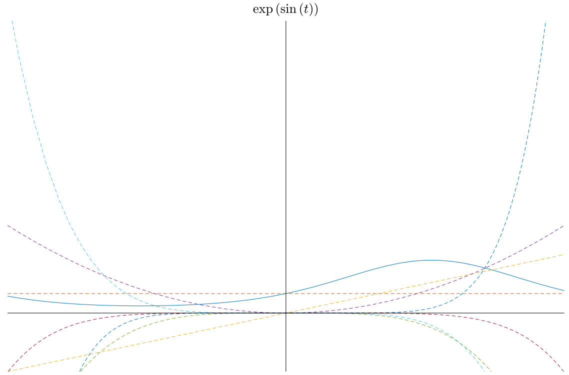
$$f(t) = \exp(t) \approx 1 + t + \frac{t^2}{2} + \mathcal{O}(t^3)$$




Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Taylor series and more (cont.)

$$f(t) = \exp[\sin(t)] \approx 1 + t + \frac{t^2}{2} - \frac{t^4}{8} - \frac{t^5}{15} - \frac{t^6}{240} + \frac{t^7}{90} + \mathcal{O}(t^8)$$




Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Taylor series and more (cont.)

The output math expressions can displayed using the syntax of LaTeX

```
>>> from sympy import latex

>>> print latex(f.series(t,0,7))
    '1 + t + \frac{t^{2}}{2} - \frac{t^{4}}{8} -
    \frac{t^{5}}{15} - \frac{t^{6}}{240} +
    \mathcal{O}\left(t^{7}\right)'
```

$$\rightsquigarrow \quad 1 + t + \frac{t^2}{2} - \frac{t^4}{8} - \frac{t^5}{15} - \frac{t^6}{240} + \mathcal{O}\left(t^7\right)$$

■

Computing with formulas
FC
CK0030
2018.1
Complex numbers
Complex arithmetics
Complex functions
Symbolic computing
Differentiation and integration
Equation solving
Taylor series and more

## Taylor series and more (cont.)

Python offers also tools to expand and simplify mathematical expressions

**Example**

Consider the angle sum/difference identities used in trigonometry

$$\cos(x \pm y) = \cos(x)\cos(y) \mp \sin(x)\sin(y)$$
$$\sin(x \pm y) = \sin(x)\cos(y) \pm \sin(x)\cos(y)$$

```
>>> from sympy import simplify, expand
>>> from sympy import cos

>>> x, y = symbols('x y')

>>> f = -sin(x)*sin(y) + cos(x)*cos(y)
>>> simplify(f)
    cos(x + y)                          # Known trigonometric identity

>>> expand(sin(x + y), trig=True)          # Needs trigonometric hint
    sin(x)*cos(y) + sin(y)*cos(x)
```

■