Computational Analysis of Physical Systems (Lecture 11)

Symbolic computation
with Python

SymPy module

1. Evaluate expressions with arbitrary precission.
2. Perform algebraic manipulations on symbolic expressions.
3. Perform basic calculus tasks (limits, differentiation and integration) with symbolic expressions.
4. Solve polynomial and trascendental equations.
5. Solve some differential equations.

SymPy is a Python library for symbolic mathematics. It aims become a full featured computer algebra system thatn can compete directly with commercial alternatives (Mathematica, Maple) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python and does not require any external libraries.

Using as a calculator

SymPy has three built-in numeric types: Float, Rational and Integer.

The Rational class represents a rational number as a pair of two Integers: the numerator and the denominator. So Rational(1,2) represents 1/2, Rational(5,2) represents 5/2, and so on.

```
>>> from sympy import Rational
>>> a = Rational(1,2)

>>> a
1/2

>>> a*2
1

>>> Rational(2)**50/Rational(10)**50
1/888178419700125232338905334472656 25
```

Using as a calculator

Proceed with caution while working with Python int's and floating point numbers, especially in division, since you may create a Python number, not a SymPy number. A ratio of two Python ints may create a float – the "true division" standard of Python 3 and the default behavior of `isympy` which imports division from __future__:

```
>>> 1/2
0.5
```

But in earlier Python versions where division has not been imported, a truncated int will result:

```
>>> 1/2
0
```

In both cases, however, you are not dealing with a SymPy Number because Python created its own number. Most of the time you will probably be working with Rational numbers, so make sure to use Rational to get the SymPy result. One might find it convenient to equate R and Rational:

```
>>> R = Rational
>>> R(1, 2)
1/2
>>> R(1)/2 # R(1) is a SymPy Integer and Integer/int gives a Rational
1/2
```

Using as a calculator

We also have some special constants, like e and pi, that are treated as symbols (1+pi won't evaluate to something numeric, rather it will remain as 1+pi), and have arbitrary precision:

```
>>> from sympy import pi, E
>>> pi**2
pi**2

>>> pi.evalf()
3.14159265358979

>>> (pi + E).evalf()
5.85987448204884
```

as you see, evalf evaluates the expression to a floating-point number

The symbol oo is used for a class defining mathematical infinity:

```
>>> from sympy import oo
>>> oo > 99999
True
>>> oo + 1
oo
```

Symbols

In contrast to other Computer Algebra Systems, in SymPy you have to declare symbolic variables explicitly:

```
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> y = Symbol('y')
```

On the left is the normal Python variable which has been assigned to the SymPy Symbol class. Predefined symbols (including those for symbols with Greek names) are available for import from abc:

```
>>> from sympy.abc import x, theta
```

Symbols can also be created with the `symbols` or `var` functions, the latter automatically adding the created symbols to the namespace, and both accepting a range notation:

```
>>> from sympy import symbols, var
>>> a, b, c = symbols('a,b,c')
>>> d, e, f = symbols('d:f')
>>> var('g:h')
(g, h)
>>> var('g:2')
(g0, g1)
```

Symbols

Instances of the Symbol class “play well together” and are the building blocks of expresions:

```
>>> x+y+x-y
2*x

>>> (x+y)**2
(x + y)**2

>>> ((x+y)**2).expand()
x**2 + 2*x*y + y**2
```

They can be substituted with other numbers, symbols or expressions using `subs(old, new)`:

```
>>> ((x+y)**2).subs(x, 1)
(y + 1)**2

>>> ((x+y)**2).subs(x, y)
4*y**2

>>> ((x+y)**2).subs(x, 1-y)
1
```

Symbolic manipulations

```
In [23]: expand((x+y)**3)
Out[23]: 3*x*y**2 + 3*y*x**2 + x**3 + y**3
```

```
In [28]: expand(x+y, complex=True)
Out[28]: I*im(x) + I*im(y) + re(x) + re(y)

In [30]: expand(cos(x+y), trig=True)
Out[30]: cos(x)*cos(y) - sin(x)*sin(y)
```

```
In [19]: simplify((x+x*y)/x)
Out[19]: 1 + y
```

Printing the results better

```
>>> from sympy import init_printing
>>> init_printing(use_unicode=False, wrap_line=False, no_global=True)
```

This will make things look better when printed.

Algebra

For partial fraction decomposition, use `apart(expr, x)`:

```
>>> from sympy import apart
>>> from sympy.abc import x, y, z

>>> 1/( (x+2)*(x+1) )
       1
---------------
(x + 1)*(x + 2)

>>> apart(1/( (x+2)*(x+1) ), x)
    1       1
- ----- + -----
  x + 2   x + 1

>>> (x+1)/(x-1)
x + 1
-----
x - 1

>>> apart((x+1)/(x-1), x)
      2
1 + -----
    x - 1
```

Algebra

To combine things back together, use `together(expr, x)`:

```
>>> from sympy import together
>>> together(1/x + 1/y + 1/z)
x*y + x*z + y*z
---------------
     x*y*z

>>> together(apart((x+1)/(x-1), x), x)
x + 1
-----
x - 1

>>> together(apart(1/( (x+2)*(x+1) ), x), x)
       1
---------------
(x + 1)*(x + 2)
```

Calculus - limits

Limits are easy to use in SymPy, they follow the syntax `limit(function, variable, point)`, so to compute the limit of f(x) as x -> 0, you would issue `limit(f, x, 0)`:

```
>>> from sympy import limit, Symbol, sin, oo
>>> x = Symbol("x")
>>> limit(sin(x)/x, x, 0)
1
```

you can also calculate the limit at infinity:

```
>>> limit(x, x, oo)
oo

>>> limit(1/x, x, oo)
0

>>> limit(x**x, x, 0)
1
```

Calculus - differentiation

You can differentiate any SymPy expression using `diff(func, var)`. Examples:

```
>>> from sympy import diff, Symbol, sin, tan
>>> x = Symbol('x')
>>> diff(sin(x), x)
cos(x)
>>> diff(sin(2*x), x)
2*cos(2*x)

>>> diff(tan(x), x)
   2
tan (x) + 1
```

You can check, that it is correct by:

```
>>> from sympy import limit
>>> from sympy.abc import delta
>>> limit((tan(x + delta) - tan(x))/delta, delta, 0)
   2
tan (x) + 1
```

Calculus - differentiation

Higher derivatives can be calculated using the `diff(func, var, n)` method:

```
>>> diff(sin(2*x), x, 1)
2*cos(2*x)

>>> diff(sin(2*x), x, 2)
-4*sin(2*x)

>>> diff(sin(2*x), x, 3)
-8*cos(2*x)
```

Calculus – series expansion

Use .series(var, point, order):

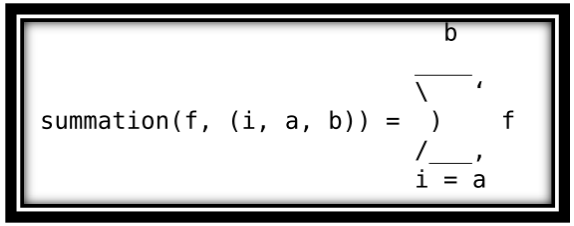
```
>>> from sympy import Symbol, cos
>>> x = Symbol('x')
>>> cos(x).series(x, 0, 10)
     2    4     6      8
    x    x     x      x      / 10\
1 - -- + -- - --- + ----- + O\x  /
    2    24   720   40320
>>> (1/cos(x)).series(x, 0, 10)
     2      4       6        8
    x    5*x    61*x    277*x     / 10\
1 + -- + ---- + ----- + ------ + O\x  /
    2     24     720     8064
```

Another simple example:

```
>>> from sympy import Integral, pprint

>>> y = Symbol("y")
>>> e = 1/(x + y)
>>> s = e.series(x, 0, 5)

>>> print(s)
1/y - x/y**2 + x**2/y**3 - x**3/y**4 + x**4/y**5 + O(x**5)
>>> pprint(s)
          2    3    4
1   x    x    x    x     / 5\
- - -- + -- - -- + -- + O\x /
y    2    3    4    5
    y    y    y    y
```

Calculus - summation

```
>>> from sympy import summation, oo, symbols, log
>>> i, n, m = symbols('i n m', integer=True)

>>> summation(2*i - 1, (i, 1, n))
 2
n
>>> summation(1/2**i, (i, 0, oo))
2
>>> summation(1/log(n)**n, (n, 2, oo))
  oo
 ____,
 \   ,
  \      -n
  /   log (n)
 /__,
n = 2
>>> summation(i, (i, 0, n), (n, 0, m))
      3    2
m    m    m
-- + -- + -
6    2    3
>>> from sympy.abc import x
>>> from sympy import factorial
>>> summation(x**n/factorial(n), (n, 0, oo))
 x
e
```



If it cannot compute the sum, it prints the corresponding summation formula.

Calculus - integration

SymPy has support for indefinite and definite integration of transcendental elementary and special functions via `integrate()` facility, which uses powerful extended Risch-Norman algorithm and some heuristics and pattern matching:

```
>>> from sympy import integrate, erf, exp, sin, log, oo, pi, sinh, symbols
>>> x, y = symbols('x,y')
```

You can integrate elementary functions:

```
>>> integrate(6*x**5, x)
 6
x
>>> integrate(sin(x), x)
-cos(x)
>>> integrate(log(x), x)
x*log(x) - x
>>> integrate(2*x + sinh(x), x)
 2
x  + cosh(x)
```

Also special functions are handled easily:

```
>>> integrate(exp(-x**2)*erf(x), x)
  ____      2
\/ pi *erf (x)
--------------
      4
```

Calculus - integration

It is possible to compute definite integrals:

```
>>> integrate(x**3, (x, -1, 1))
0
>>> integrate(sin(x), (x, 0, pi/2))
1
>>> integrate(cos(x), (x, -pi/2, pi/2))
2
```

Also, improper integrals are supported as well:

```
>>> integrate(exp(-x), (x, 0, oo))
1
>>> integrate(log(x), (x, 0, 1))
-1
```

Complex numbers

Besides the imaginary unit, I, which is imaginary, symbols can be created with attributes (e.g. real, positive, complex, etc...) and this will affect how they behave:

```
>>> from sympy import Symbol, exp, I
>>> x = Symbol("x") # a plain x with no attributes
>>> exp(I*x).expand()
 I*x
e
>>> exp(I*x).expand(complex=True)
   -im(x)                -im(x)
I*e      *sin(re(x)) + e      *cos(re(x))
>>> x = Symbol("x", real=True)
>>> exp(I*x).expand(complex=True)
I*sin(x) + cos(x)
```

Functions - trigonometric

```
>>> from sympy import asin, asinh, cos, sin, sinh, symbols, I
>>> x, y = symbols('x,y')

>>> sin(x+y).expand(trig=True)
sin(x)*cos(y) + sin(y)*cos(x)

>>> cos(x+y).expand(trig=True)
-sin(x)*sin(y) + cos(x)*cos(y)

>>> sin(I*x)
I*sinh(x)

>>> sinh(I*x)
I*sin(x)

>>> asinh(I)
I*pi
----
 2
```

Functions - trigonometric

```
>>> asinh(I*x)
I*asin(x)

>>> sin(x).series(x, 0, 10)
     3     5     7       9
    x     x     x       x       / 10\
x - -- + --- - ---- + ------ + O\x  /
    6    120   5040   362880

>>> sinh(x).series(x, 0, 10)
     3     5     7       9
    x     x     x       x       / 10\
x + -- + --- + ---- + ------ + O\x  /
    6    120   5040   362880

>>> asin(x).series(x, 0, 10)
     3      5      7       9
    x    3*x    5*x    35*x     / 10\
x + -- + ---- + ---- + ----- + O\x  /
    6     40     112     1152

>>> asinh(x).series(x, 0, 10)
     3      5      7       9
    x    3*x    5*x    35*x     / 10\
x - -- + ---- - ---- + ----- + O\x  /
    6     40     112     1152
```

Functions – spherical harmonics

```
>>> from sympy import Ylm
>>> from sympy.abc import theta, phi

>>> Ylm(1, 0, theta, phi)

  ___
\/ 3 *cos(theta)
----------------

    _____
  2*\/ pi

>>> Ylm(1, 1, theta, phi)
      ___     I*phi
-\/ 6 *e     *sin(theta)
------------------------

         _____
       4*\/ pi

>>> Ylm(2, 1, theta, phi)
      _____    I*phi
-\/ 30 *e     *sin(theta)*cos(theta)
-------------------------------------

              _____
            4*\/ pi
```

Functions – factorials and gamma function

```
>>> from sympy import factorial, gamma, Symbol
>>> x = Symbol("x")
>>> n = Symbol("n", integer=True)
>>> factorial(x)
x!

>>> factorial(n)
n!

>>> gamma(x + 1).series(x, 0, 3) # i.e. factorial(x)
                     /          2     2\
                   2 |EulerGamma    pi |    / 3\
1 - EulerGamma*x + x *|----------- + ---| + O\x /
                     \     2         12/
```

Functions – zeta function

```
>>> from sympy import zeta
>>> zeta(4, x)
zeta(4, x)

>>> zeta(4, 1)
  4
pi
---
 90

>>> zeta(4, 2)
       4
     pi
-1 + ---
      90

>>> zeta(4, 3)
         4
  17   pi
- -- + ---
  16    90
```

Polynomials

```
>>> from sympy import assoc_legendre, chebyshevt, legendre, hermite
>>> chebyshevt(2, x)
   2
2*x  - 1

>>> chebyshevt(4, x)
   4      2
8*x  - 8*x  + 1

>>> legendre(2, x)
   2
3*x    1
---- - -
 2     2

>>> legendre(8, x)
      8         6          4         2
6435*x    3003*x    3465*x    315*x     35
------- - ------- + ------- - ------ + ---
  128        32         64       32     128

>>> assoc_legendre(2, 1, x)
        ___________
       /     2
-3*x*\/   - x  + 1

>>> assoc_legendre(2, 2, x)
     2
- 3*x  + 3

>>> hermite(3, x)
   3
8*x  - 12*x
```

Differential equations

```
>>> from sympy import Function, Symbol, dsolve
>>> f = Function('f')
>>> x = Symbol('x')
>>> f(x).diff(x, x) + f(x)
         2
        d
f(x) + ---(f(x))
         2
       dx

>>> dsolve(f(x).diff(x, x) + f(x), f(x))
f(x) = C1*sin(x) + C2*cos(x)
```

Algebraic equations

```
>>> from sympy import solve, symbols
>>> x, y = symbols('x,y')
>>> solve(x**4 - 1, x)
[-1, 1, -I, I]

>>> solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y])
{x: -3, y: 1}
```

Linear algebra

```
>>> from sympy import Matrix, Symbol
>>> Matrix([[1,0], [0,1]])
[1  0]
[    ]
[0  1]
```

They can also contain symbols:

```
>>> x = Symbol('x')
>>> y = Symbol('y')
>>> A = Matrix([[1,x], [y,1]])
>>> A
[1  x]
[    ]
[y  1]

>>> A**2
[x*y + 1    2*x  ]
[                ]
[  2*y    x*y + 1]
```

Printing

```
>>> from sympy import Integral
>>> from sympy.abc import x
>>> print x**2
x**2
>>> print 1/x
1/x
>>> print Integral(x**2, x)
Integral(x**2, x)
```

```
>>> from sympy import Integral, pprint
>>> from sympy.abc import x
>>> pprint(x**2)
 2
x
>>> pprint(1/x)
1
-
x
>>> pprint(Integral(x**2, x))
  /
 |
 |  2
 | x  dx
 |
/
```

```
>>> pprint(Integral(x**2, x), use_unicode=True)
⌠
⎮  2
⎮ x  dx
⌡
```

LaTeX printing

```
>>> from sympy import Integral, latex
>>> from sympy.abc import x
>>> latex(x**2)
x^{2}
>>> latex(x**2, mode='inline')
$x^{2}$
>>> latex(x**2, mode='equation')
\begin{equation}x^{2}\end{equation}
>>> latex(x**2, mode='equation*')
\begin{equation*}x^{2}\end{equation*}
>>> latex(1/x)
\frac{1}{x}
>>> latex(Integral(x**2, x))
\int x^{2}\, dx
```