# DIP L2

First Steps with SymPy

- Using SymPy as a calculator
- Symbols

Algebraic manipulations

- Expand
- Simplify

Calculus

- Limits
- Differentiation
- Series expansion
- Integration

Equation solving

Linear Algebra

- Matrices
- Differential Equations

First Steps with SymPy

- Using SymPy as a calculator
- Symbols

Sympy defines three numerical types Real, Rational and Integer

Rational class represents a rational number as a pair of two integers: the numerator and denominator

Example : Rational (1,2)

Represents 1/2

Example : Rational (5,2)

Represents 5/2

```
>>> import sympy as sym

>>>a = sym.Rational(1,2)

>>>a = ½

>>>a*2

>>>1
```

Sympy

uses mpmath in the background, which makes it possible to perform computations using arbitrary-precision arithmetic. That way, some special constants, like e, pi, oo (Infinity), are treated as symbols and can be evaluated with arbitrary precision:

Example :

```
>>>sym.pi**2
pi**2
```

```
>>>sym.pi.evalf{2}
3.14159265358979
```

Example :

```
>>>(sym.pi + sym.exp(1) ).evalf()
5.85987448204884
```

Evalf evaluates expression in floating point number.

There is also a class representing mathematical infinity, called `oo`

Example : Rational (5,2)

```
>>>sym.oo > 99999
sym.oo + 1:
```

## Symbols

in SymPy you have to declare symbolic variables explicitly:

```
>>> x = sym.Symbol('x')
>>> y = sym.Symbol('y')
```

Then you can manipulate them:

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

Symbols can now be manipulated using some of python operators: `+`, `-`, `` `*` ``, `**` (arithmetic), &, |, ~ , >>, << (boolean).

## Printing

Sympy allows for control of the display of the output. From here we use the following setting for printing:

```
>>>
sym.init_printing(use_unicode=False, wrap_line=True)
```

## Algebraic Manipulations

SymPy is capable of performing powerful algebraic manipulations.

Use this to expand an algebraic expression. It will try to denest powers and multiplications:

```
>>> sym.expand((x + y) ** 3)
```

$x^3 + 3*x^2*y + 3*x^2*y + y^3$

```
>>> 3 * x * y ** 2 + 3 * y * x ** 2 + x ** 3 + y ** 3
```

$x^3 + 3*x^2*y + 3*x^2*y + y^3$

## Example in form of words

sym.expand(x + y, complex=True)

re(x) + re(y) + I*im(x) + I*im(y)

sym.I * sym.im(x) + sym.I * sym.im(y) + sym.re(x) + sym.re(y)

re(x) + re(y) + I*im(x) + I*im(y)

sym.expand(sym.cos(x + y), trig=True)

-sin(x)*sin(y) + cos(x)*cos(y)

sym.cos(x) * sym.cos(y) - sym.sin(x) * sym.sin(y)

-sin(x)*sin(y) + cos(x)*cos(y)

## Algebraic Manipulations

Use simplify if you would like to transform an expression into a simpler form:

```
>>> sym.simplify((x + x * y) / x)
y+1
```

Simplification is a somewhat vague term, and more precises alternatives to simplify exists: powsimp (simplification of exponents), trigsimp (for trigonometric expressions) , logcombine, radsimp, together.

## Example in form of words

```
sym.expand(x + y, complex=True)
re(x) + re(y) + I*im(x) + I*im(y)

sym.I * sym.im(x) + sym.I * sym.im(y) + sym.re(x) + sym.re(y)
re(x) + re(y) + I*im(x) + I*im(y)

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

sym.cos(x) * sym.cos(y) - sym.sin(x) * sym.sin(y)
-sin(x)*sin(y)  + cos(x)*cos(y)
```

## 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 \rightarrow 0, you would issue limit(f, x, 0):

```
>>> sym.limit(sym.sin(x) / x, x, 0)
1
```

calculate the limit at infinity:

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

```
>>> sym.limit(1 / x, x, sym.oo)
0
```

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

## Differentiation

SymPy expression using diff(func, var)

Examples:

```
>>> sym.diff(sym.sin(x), x)
cos(x)
>>> sym.diff(sym.sin(2 * x), x)
2*cos(2*x)

>>> sym.diff(sym.tan(x), x)
```
$\tan^2(x) + 1$

To check

```
>>> sym.limit((sym.tan(x + y) - sym.tan(x)) / y, y, 0)
```
$\tan^2(x) + 1$

## Derivatives

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

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

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

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

# Integration

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

Elementary function

```
sym.integrate(6 * x ** 5, x)
x⁶
```

```
>>> sym.integrate(sym.sin(x), x)
-cos(x)
```

```
>>> sym.integrate(sym.log(x), x)
x*log(x) - x
```

```
>>> sym.integrate(2 * x + sym.sinh(x), x)
x²  + cosh(x)
```

# Integration

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

Elementary function

```
>>> sym.integrate(6 * x ** 5, x)
x⁶
```

```
>>> sym.integrate(sym.sin(x), x)
-cos(x)
```

```
>>> sym.integrate(sym.log(x), x)
x*log(x) - x
```

```
>>> sym.integrate(2 * x + sym.sinh(x), x)
x²  + cosh(x)
```

# Integration

## special function

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

## Definite Integral

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

# Integration

## special function

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

## Definite Integral

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

# Integration

## Improper Integral

```
>>> sym.integrate(sym.exp(-x), (x, 0, sym.oo))
1
>>> sym.integrate(sym.exp(-x ** 2), (x, -sym.oo, sym.oo))
  ____
\/ pi
```

# Equation Solving

SymPy is able to solve algebraic equations, in one and several variables using **`solveset()`**:

```
>>> sym.solveset(x ** 4 - 1, x)
{-1, 1, -I, I}
```

It takes as first argument an expression that is supposed to be equaled to 0. It also has (limited) support for transcendental equations:

```
>>> sym.solveset(sym.exp(x) + 1, x)
{I*(2*n*pi + pi) | n in Integers}
```

# Systems of Linear Equations

Sympy is able to solve a large part of polynomial equations, and is also capable of solving multiple equations with respect to multiple variables giving a tuple as second argument. To do this you use the

```
>>> solution = sym.solve((x + 5 * y - 2, -3 * x + 6 * y - 15), (x, y))
>>> solution[x], solution[y]
(-3, 1)
```

Another alternative in the case of polynomial equations is factor. factor returns the polynomial factorized into irreducible terms, and is capable of computing the factorization over various domains:

```
>>> f = x ** 4 - 3 * x ** 2 + 1
>>> sym.factor(f)
```

$(x^2 - x - 1) * (x^2 - x + 1)$

```
>>> sym.factor(f, modulus=5)
```

$(x - 2)^2 * (x - 2)^2$

# Systems of Linear Equations

SymPy is also able to solve boolean equations, that is, to decide if a certain boolean expression is satisfiable or not. For this, we use the function satisfiable:

```
>>> sym.satisfiable(x & y)
{x: True, y: True}
```

This tells us that `(x & y)` is True whenever `x` and `y` are both True. If an expression cannot be true, i.e. no values of its arguments can make the expression True, it will return False:

```
>>> sym.satisfiable(x & ~x)
False
```

# Linear Algebra

Matrices are created as instances from the Matrix class:

```
>>> sym.Matrix([[1, 0], [0, 1]])
[1  0]
[    ]
[0  1]
```

unlike a NumPy array, you can also put Symbols in it:

```
>>> x, y = sym.symbols('x, y')
>>> A = sym.Matrix([[1, x], [y, 1]])
>>> A
[1  x]
[    ]
[y  1]
>>> A**2
[x*y + 1    2*x  ]
[                ]
[  2*y    x*y + 1]
```

# Differential Equations

SymPy is capable of solving (some) Ordinary Differential. To solve differential equations, use dsolve. First, create an undefined function by passing cls=Function to the symbols function:

```
>>> f, g = sym.symbols('f g', cls=sym.Function)
```

---

f and g are now undefined functions. We can call f(x), and it will represent an unknown function:

```
>>> f (x)
```

f(x)

```
>>> f(x).diff(x, x) + f(x)
```

$f(x) + d^2/dx^2 f((x))$

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

# Differential Equations

Keyword arguments can be given to this function in order to help if find the best possible resolution system. For example, if you know that it is a separable equations, you can use keyword `hint='separable'` to force dsolve to resolve it as a separable equation:

```
>>> sym.dsolve(sym.sin(x) * sym.cos(f(x)) + sym.cos(x) * sym.sin(f(x)) *
f(x).diff(x), f(x), hint='separable')
```

```
f(x) = - acos(c1/cos(x)) + 2*pi,  f(x) = acos(c1/cos(x))
```

# Seatwork - individual

- Calculate $\sqrt{2}$ with 100 decimals.
- Calculate 1/2 + 1/3 in rational arithmetic.
- Calculate the expanded form of (x+y)^6.
- Simplify the trigonometric expression sin(x) / cos(x)
- Calculate $\lim_{x \to 0}$ sin(x)/x
- Calculate the derivative of log(x) for x.
- Solve the system of equations x + y = 2, 2x + y = 0
- Are there boolean values x, y that make (~x | y) & (~y | x) true?
- Solve the Bernoulli differential equation

$$x\frac{df(x)}{x} + f(x) - f(x)^2 = 0$$

- Solve the same equation using hint='Bernoulli'. What do you observe ?