TECHNISCHE UNIVERSITÄT DRESDEN

Carsten Knoll

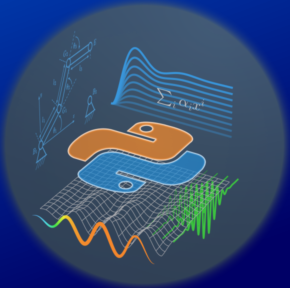
Chair of Fundamentals of Electrical Engineering

Python for Engineers
Pythonkurs für Ingenieur:innen

Symbolic Computation with Sympy
Symbolisch Rechnen mit Sympy

Dresden (Online), 2023-12-12

`https://tu-dresden.de/pythonkurs`
`https://python-fuer-ingenieure.de`



Preliminary Remarks

- goal: overview of computer algebra possibilities (`import sympy`)
- structure:
  - introduction
  - calculating
  - substituting
  - important functions & data types
  - evaluate formulas numerically

The Package Sympy

- Python library for symbolic computations ('calculations with letters')

$\rightarrow$ backend of a **c**omputer **a**lgebra **s**ystem (CAS)

- possible frontends: own Python script, IPython shell, Jupyter notebook (in browser)

- advantage over other products: CAS-functionality inside a <u>real</u> programming language

Sympy Overview

- ways to import the package or objects from it:

1. `import sympy as sp`
2. `from sympy import sin, cos, pi`
3. `from sympy import *`   ⚠ caution: “namespace pollution”

Sympy Overview

- ways to import the package or objects from it:
  1. `import sympy as sp`
  2. `from sympy import sin, cos, pi`
  3. `from sympy import *` ⚠ caution: “namespace pollution”
- `sp.symbols, sp.Symbol`: create symbols ($\neq$ variables)
- `sp.sin(x), sp.cos(2*pi*t), sp.exp(x),...`: mathematical functions
- `sp.Function('f')(x)`: create and evaluate custom functions
- `sp.diff(<expr>, <var>)` or `<expr>.diff(<var>)`: taking derivatives
- `sp.simplify(<expr>)`: simplification
- `<expr>.expand()`: expansion of brackets ($a(x+y) \to ax+ay$)
- `<expr>.subs(...)`: substitution
- `sp.pprint(<expr>)`: “pretty printing”

Performing Calculations

Listing: sympy1.py

```python
import sympy as sp
x = sp.Symbol("x")
a, b, c = sp.symbols("a b c")  # different ways to create symbols

z = a*b*x*b + b**2*a*x - c*b*(2*a/c*x*b-1/(b*2))
print(z)  # -> -b*c*(-1/(2*b) + 2*a*b*x/c) + 2*a*x*b**2
print(z.expand())  # -> c/2 (multiply out)

# apply functions:
y = sp.sin(x)*sp.exp(3*x)*sp.sqrt(a)
print(y)  # -> a**(1/2)*exp(x)*sin(x)

# define custom function
f1 = sp.Function("f")  # -> sympy.core.function.f (not evaluated)
g1 = sp.Function("g")(x)  # -> g(x)     (function evaluated at x)

# taking derivatives
print(y.diff(x))  # -> 3*sqrt(a)*exp(3*x)*sin(x) + sqrt(a)*exp(3*x)*cos(x)
print(g1.diff(x))  # -> Derivative(g(x), x)

# simplification (example):
y = sp.sin(x)**2+sp.cos(x)**2)
print(y == 1)  # -> False
print(sp.simplify(y))  # -> 1
print(sp.simplify(y) == 1)  # -> True
```

Substitutions: `<expr>.subs(...)`

- comparable to `<str>.replace(old, new)`
- useful for: manual simplifications, (partial) function evaluations, coordinate transformations.

```
term1 = a*b*sp.exp(c*x)
term2 = term1.subs(a, 1/b)
print(term2) # -> exp(c*x)
```

Substitutions: `<expr>.subs(...)`

- comparable to `<str>.replace(old, new)`
- useful for: manual simplifications, (partial) function evaluations, coordinate transformations.

```
term1 = a*b*sp.exp(c*x)
term2 = term1.subs(a, 1/b)
print(term2) # -> exp(c*x)
```

- call options: 1. two arguments, 2. list with 2-tuples, 3. dict
  1. `<expr>.subs(old, new)`
  2. `<expr>.subs([(old1, new1), (old2, new2), ...])`
  3. `<expr>.subs({old1: new1, old2: new2, ...})`
     - option 2: order of list $\rightarrow$ substitution order
     - relevant when substituting derivatives (see example notebook)
- important: `.subs(...)` returns new expression (original expr. remains unchanged)

More Important Functions, Methods, Types

- `A = sp.Matrix([[x1, a*x2], [c*x3, sp.sin(x1)]])` : create a matrix
- `A.jacobian([x1, x2, x3])` : Jacobian matrix of a vector (matrix of partial derivatives)
- `sp.solve(x**2 + x - a, x)` : solve (systems of) equations
- `<expr>.atoms(), <expr>.atoms(sp.sin)` : “atoms” (of a certain type)
- `<expr>.args` : arguments of the respective class (summands, factors, ...)

- `sp.sympify(...)` : adapt data type (plain Python $\rightarrow$ Sympy)
- `sp.integrate(<expr>, <var>)` : symbolic integration (anti derivative)
- `sp.series(...)` : Taylor series expansion of expression $\left(\text{like } \mathrm{e}^x\Big|_{x=0} = \sum_{k=0}^{n} \frac{1}{k!}x^k\right)$
- `sp.limit(<var>, <value>)` : limit $\left(\text{like } \lim_{x\to 0} \frac{\sin(x)}{x} = 1\right)$
- `<expr>.as_num_denom()` : decomposition into 2-tuple: $(\mathit{numerator}, \mathit{denominator})$
- `sp.poly(x**7+a*x**3+b*x+c, gens=[x], domain="EX")` : Polynomial
- `sp.Piecewise(...)` : piecewise defined function

$\rightarrow$ see docs (and docstrings) for more info (or just try things out)

Numeric Formula Evaluation

- given: expression and values of the individual variables
- wanted: numerical result

Numeric Formula Evaluation

- given: expression and values of the individual variables
- wanted: numerical result
- possible in principle: `expr.subs(num_values).evalf()`

Numeric Formula Evaluation

- given: expression and values of the individual variables
- wanted: numerical result
- possible in principle: `expr.subs(num_values).evalf()`
- better (in terms of speed): `lambdify` (origin of the namet: Pythons `lambda` functions)
- creates a Python function that you can then call with the arguments

```
f = a*sp.sin(b*x)
df_xa = f.diff(x)

# create the function
df_xa_fnc = sp.lambdify((a, b, x), df_xa, modules='numpy')

# call (= evaluate) the function
print( f_xa_fnc(1.2, 0.5, 3.14) )
```

Sympy Support in Jupyter

Another trick: let sympy expressions be nicely rendered by $\LaTeX \Rightarrow$ readability $\uparrow$.

In [7]:
```
from sympy.interactive import printing
printing.init_printing()

%load_ext ipydex.displaytools
```

In [8]:
```
# same code with special-comments (`##:`)

x = sp.Symbol("x")
a, b, c, z = sp.symbols("a b c z") # create several symbols at once

some_formula = a*b*x*b + b**2*a*x - c*b*(2*a/c*x*b-1/(b*2)) ##:

# some calculus
y = sp.sin(x)*sp.exp(3*x)*sp.sqrt(a) ##:

# derive
yd = y.diff(x) ##:
```

some-formula $:= 2ab^2x - bc\left(\frac{2a}{c}bx - \frac{1}{2b}\right)$

---

y $:= \sqrt{a}e^{3x}\sin(x)$

---

yd $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

See Example Notebook

`../notebooks/course09-sympy-demo.html`

Sympy Support in Jupyter

Another trick: let sympy expressions be nicely rendered by $\LaTeX \Rightarrow$ readability $\uparrow$.

In [7]:
```
from sympy.interactive import printing
printing.init_printing()

%load_ext ipydex.displaytools
```

In [8]:
```
# same code with special-comments (`##:`)

x = sp.Symbol("x")
a, b, c, z = sp.symbols("a b c z") # create several symbols at once

some_formula = a*b*x*b + b**2*a*x - c*b*(2*a/c*x*b-1/(b*2)) ##:

# some calculus
y = sp.sin(x)*sp.exp(3*x)*sp.sqrt(a) ##:

# derive
yd = y.diff(x) ##:
```

some-formula $:= 2ab^2x - bc\left(\frac{2a}{c}bx - \frac{1}{2b}\right)$

---

y $:= \sqrt{a}e^{3x}\sin(x)$

---

yd $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

See Example Notebook

`../notebooks/course09-sympy-demo.html`

for $\LaTeX$ output

Sympy Support in Jupyter

Another trick: let sympy expressions be nicely rendered by $\LaTeX \Rightarrow$ readability ↑.

In [7]:
```
from sympy.interactive import printing
printing.init_printing()

%load_ext ipydex.displaytools
```

In [8]:
```
# same code with special-comments ( ##:`)

x = sp.Symbol("x")
a, b, c, z = sp.symbols("a b c z") # create several symbols at once

some_formula = a*b*x*b + b**2*a*x - c*b*(2*a/c*x*b-1/(b*2)) ##:

# some calculus
y = sp.sin(x)*sp.exp(3*x)*sp.sqrt(a) ##:

# derive
yd = y.diff(x) ##:
```

some-formula $:= 2ab^2x - bc\left(\frac{2a}{c}bx - \frac{1}{2b}\right)$

---

y $:= \sqrt{a}e^{3x}\sin(x)$

---

yd $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

See Example Notebook

`../notebooks/course09-sympy-demo.html`

for $\LaTeX$ output

enables special comment:

`##:`

Sympy Support in Jupyter

Another trick: let sympy expressions be nicely rendered by $\LaTeX \Rightarrow$ readability $\uparrow$.

```
In [7]: from sympy.interactive import printing
        printing.init_printing()

        %load_ext ipydex.displaytools
```

```
In [8]: # same code with special-comments ( ##:`)

        x = sp.Symbol("x")
        a, b, c, z = sp.symbols("a b c z") # create several symbols at once

        some_formula = a*b*x*b + b**2*a*x - c*b*(2*a/c*x*b-1/(b*2)) ##:

        # some calculus
        y = sp.sin(x)*sp.exp(3*x)*sp.sqrt(a) ##:

        # derive
        yd = y.diff(x) ##:
```

$\texttt{some-formula} := 2ab^2x - bc\left(\frac{2a}{c}bx - \frac{1}{2b}\right)$

$y := \sqrt{a}e^{3x}\sin(x)$

$yd := 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

See Example Notebook

`../notebooks/course09-sympy-demo.html`

for $\LaTeX$ output

enables special comment:

`##:`

shows the result of assignments

Sympy Support in Jupyter

Another trick: let sympy expressions be nicely rendered by $\LaTeX \Rightarrow$ readability ↑.

In [7]:
```
from sympy.interactive import printing
printing.init_printing()

%load_ext ipydex.displaytools
```

In [8]:
```
# same code with special-comments ( ##:`)

x = sp.Symbol("x")
a, b, c, z = sp.symbols("a b c z") # create several symbols at once

some_formula = a*b*x*b + b**2*a*x - c*b*(2*a/c*x*b-1/(b*2)) ##:

# some calculus
y = sp.sin(x)*sp.exp(3*x)*sp.sqrt(a) ##:

# derive
yd = y.diff(x) ##:
```

some-formula $:= 2ab^2x - bc\left(\frac{2a}{c}bx - \frac{1}{2b}\right)$

---

y $:= \sqrt{a}e^{3x}\sin(x)$

---

yd $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

See Example Notebook

`../notebooks/course09-sympy-demo.html`

for $\LaTeX$ output

enables special comment:

`##:`

shows the result of assignments

Sympy Support in Jupyter

Another trick: let sympy expressions be nicely rendered by $\LaTeX \Rightarrow$ readability ↑.

In [7]:
```
from sympy.interactive import printing
printing.init_printing()

%load_ext ipydex.displaytools
```

In [8]:
```
# same code with special-comments ( ##:`)

x = sp.Symbol("x")
a, b, c, z = sp.symbols("a b c z") # create several symbols at once

some_formula = a*b*x*b + b**2*a*x - c*b*(2*a/c*x*b-1/(b*2)) ##:

# some calculus
y = sp.sin(x)*sp.exp(3*x)*sp.sqrt(a) ##:

# derive
yd = y.diff(x) ##:
```

some-formula $:= 2ab^2x - bc\left(\frac{2a}{c}bx - \frac{1}{2b}\right)$

---

y $:= \sqrt{a}e^{3x}\sin(x)$

---

yd $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

See Example Notebook

`../notebooks/course09-sympy-demo.html`

for $\LaTeX$ output

enables special comment:

`##:`

shows the result of assignments

Docs and Links

- docstrings of the respective objects (in Jupyter: e.g. `sp.solve?`)
- `http://docs.sympy.org/latest/tutorial/index.html`
- module reference (e.g.: `solve`-function)
- `http://docs.sympy.org/latest/tutorial/gotchas.html` (pitfalls)

Summary

- symbolic calculations
- substitution
- important functions / data types
- numerical evaluation of functions