TECHNISCHE UNIVERSITÄT DRESDEN

**Fakultät Elektrotechnik und Informationstechnik** Institut für Regelungs- und Steuerungstheorie

# PYTHONKURS FÜR INGENIEUR:INNEN

## Symbolisch Rechnen mit Python – das Paket `sympy`

**Carsten Knoll**

`tu-dresden.de/pythonkurs`

Dresden, 16.11.2020

# Vorbemerkungen

- Ziel: grober Überblick über Computer-Algebra Möglichkeiten (`sympy`)
- Aufbau:
  - Allgemeines
  - Rechnen
  - Substituieren
  - Wichtige Funktionen / Datentypen
  - Formeln numerisch auswerten

# Das Paket `sympy`

- Python-Bibliothek für symbolische Berechnungen („mit Buchstaben rechnen“)

$\rightarrow$ Backend eines Computer Algebra Systems (CAS)

- Mögliche Frontends: eigenes Python-Skript, IPython-Shell, IPython-Notebook (im Browser)

- Vorteil: CAS-Funktionalität zusammen mit richtiger Programmiersprache

# `sympy` Überblick

- Varianten das Paket bzw. Objekte daraus zu importieren:
  1. `import sympy as sp`
  2. `from sympy import sin, cos, pi`
  3. `from sympy import *` ⚠ Vorisicht: „namespace pollution“

# `sympy` Überblick

- Varianten das Paket bzw. Objekte daraus zu importieren:
  1. `import sympy as sp`
  2. `from sympy import sin, cos, pi`
  3. `from sympy import *` ⚠ Vorisicht: „namespace pollution"
- `sp.symbols, sp.Symbol` : Symbole erzeugen ($\neq$ Variable)
- `sp.sin(x), sp.cos(2*pi*t), sp.exp(x),...` : mathematische Funktionen
- `sp.Function('f')(x)` : eigene Funktion anlegen (und auswerten)
- `sp.diff(<expr>, <var>)` oder `<expr>.diff(<var>)` : ableiten
- `sp.simplify(<expr>)` : vereinfachen
- `<expr>.expand()` : ausmultiplizieren
- `<expr>.subs(...)` : substituieren
- `sp.pprint(<expr>)` : „pretty printing"

# Rechnen mit sympy

Listing: sympy1.py

```
import sympy as sp
x = sp.Symbol('x')
a, b, c = sp.symbols('a b c') # verschiedene Wege Symbole zu erzeugen

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 (Ausmutiplizieren)

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

# Eigene Funktionen definieren
f1 = sp.Function('f') # -> sympy.core.function.f (nicht ausgewertet)
g1 = sp.Function('g')(x) # -> g(x)     (Funktion ausgewertet bei x)


# Differenzieren
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)

# Vereinfachungen:
print(sp.trigsimp(sp.sin(x)**2+sp.cos(x)**2)) # -> 1
```

# Substituieren: `<expr>.subs(...)`

- Vergleichbar mit `<str>.replace(alt, neu)`
- Nützlich für: manuelle Vereinfachungen, (partielle) Funktionsauswertungen, Koordinatentransformationen

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

# Substituieren: `<expr>.subs(...)`

- Vergleichbar mit `<str>.replace(alt, neu)`
- Nützlich für: manuelle Vereinfachungen, (partielle) Funktionsauswertungen, Koordinatentransformationen

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

- Aufrufmöglichkeiten: 1. direkt, 2. Liste mit Tupeln, 3. dict (nicht empfohlen)
  1. `<expr>.subs(alt, neu)`
  2. `<expr>.subs([(alt1, neu1), (alt2, neu2), ...])`
     - Reihenfolge der Liste → Substitutionsreihenfolge
     - Relevant beim Substituieren von Ableitungen (siehe Beispiel-Notebook)
- Wichtig: `subs(...)` liefert Rückgabewert (original-Ausdruck bleibt unverändert)

# Weitere wichtige Methoden / Funktionen / Typen

- `sp.Matrix([[x, a+b], [c*x, sp.sin(x)]])` : Matrizen
- `<mtrx>.jacobian(xx)` : Jacobi-Matrix eines Vektors
- `sp.solve(x**2 + x - a, x)` : Gleichungen und Gleichungssysteme lösen
- `<expr>.atoms(), <expr>.atoms(sp.sin)` : „Atome“ (bestimmten Typs)
- `<expr>.args` : Argumente der jeweiligen Klasse (Summanden, Faktoren, ...)

- `sp.sympify(...)` : Datentypen-Anpassung
- `sp.integrate(<expr>, <var>)` : Integration
- `sp.series(...)` : Reihenentwicklung
- `sp.limit(<var>, <value>)` : Grenzwert
- `<expr>.as_num_denom()` : Zähler-Nenner-Aufspaltung
- `sp.Polynomial(x**7+a*x**3+b*x+c, x, domain='EX')` : Polynome
- `sp.Piecewise(...)` : Stückweise definierte Funktionen

# Numerische Formelauswertung

- Gegeben: Formel und Werte der einzelnen Variablen
- Gesucht: numerisches Ergebnis

# Numerische Formelauswertung

- Gegeben: Formel und Werte der einzelnen Variablen
- Gesucht: numerisches Ergebnis
- Prinzipiell möglich: `expr.subs(num_werte).evalf()`

# Numerische Formelauswertung

- Gegeben: Formel und Werte der einzelnen Variablen
- Gesucht: numerisches Ergebnis
- Prinzipiell möglich: `expr.subs(num_werte).evalf()`
- Besser (bzgl. Geschwindigkeit): `lambdify` (Namensherkunft: Pythons `lambda`-Funktionen)
- Erzeugt eine Python-Funktion, die man dann mit den Argumenten aufrufen kann

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

# Funktion erzeugen
df_xa_fnc = sp.lambdify((a, b, x), df_xa, modules='numpy')

# Funktion auswerten
print( f_xa_fnc(1.2, 0.5, 3.14) )
```

# Sympy-Unterstützung 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 $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

---

Siehe Beispiel-Notebook

`../notebooks/sympy-notebook1.html`

# Sympy-Unterstützung 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
```

Siehe Beispiel-Notebook

`../notebooks/sympy-notebook1.html`

Für $\LaTeX$-Ausgabe

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

---

# Sympy-Unterstützung 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 $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

---

Siehe Beispiel-Notebook

`../notebooks/sympy-notebook1.html`

Für $\LaTeX$-Ausgabe

Aktiviert speziellen Kommentar:

`##:`

# Sympy-Unterstützung 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 $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

---

Siehe Beispiel-Notebook

`../notebooks/sympy-notebook1.html`

Für $\LaTeX$-Ausgabe

Aktiviert speziellen Kommentar:

`##:`

Dieser zeigt Ergebnis von Zuweisungen an

# Sympy-Unterstützung 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 $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

---

Siehe Beispiel-Notebook

`../notebooks/sympy-notebook1.html`

Für $\LaTeX$-Ausgabe

Aktiviert speziellen Kommentar:

`##:`

Dieser zeigt Ergebnis von Zuweisungen an

# Sympy-Unterstützung 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 $:= 3\sqrt{a}e^{3x}\sin(x) + \sqrt{a}e^{3x}\cos(x)$

---

Siehe Beispiel-Notebook

`../notebooks/sympy-notebook1.html`

Für $\LaTeX$-Ausgabe

Aktiviert speziellen Kommentar:

`##:`

Dieser zeigt Ergebnis von Zuweisungen an

# Doku und Links

- Doc-Strings der einzelenen Funktionen (meist ausreichend)

- `http://docs.sympy.org/latest/tutorial/index.html`
- `http://docs.sympy.org/latest/tutorial/gotchas.html` (Fallstricke)
- Modul-Referenz (Bsp: `solve` -Funktion)

# Zusammenfassung

- Rechnen
- Substituieren
- Wichtige Funktionen / Datentypen
- Formeln numerisch auswerten
  $\rightarrow$ **Übungsaufgabe** („Lagrange-Gleichungen“)