

SYMPY

Symbolisch Rechnen mit Python

**Carsten Knoll,
Institut für Regelungs- und Steuerungstheorie, TU Dresden**

Dresden, 28.10.2013

Vorbemerkungen

- Ziel: grober unvollständiger Überblick über die Möglichkeiten
- Aufbau:
 - Grundlegendes
 - Rechnen
 - Substituieren
 - Wichtige Funktionen / Datentypen
 - Formeln numerisch auswerten

Das Modul `sympy`

- Python-Bibliothek für symbolische Berechnungen („mit Buchstaben rechnen“)
- Backend eines Computer Algebra Systems (CAS)
- Frontend: python-shell, ipython, eigenes skript
- Vorteil: CAS-Funktionalität zusammen mit richtiger Programmiersprache

Das wichtigste von `sympy`

- Modul importieren:
- `import sympy as sp`
- `from sympy import sin, cos, pi`
- `from sympy import *` ⚡ „namespace pollution“
- `sp.symbols`, `sp.Symbol`: Symbole erzeugen ($\neq$ Variable)
- `sp.sin(x)`, `sp.cos(2*pi*t)`, `sp.exp(x)`, ...: wichtige 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

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

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 (Ausmultiplizieren)

# 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.Funtion('f') # -> sympy.core.function.f
f2 = sp.Funtion('f')(x) # -> f(x) (Funktion ausgewertet bei x)

# Differenzieren
y.diff(x) # -> 3*sqrt(a)*exp(3*x)*sin(x) + sqrt(a)*exp(3*x)*cos(x)
f2.diff(x) # -> Derivative(f(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. Dictionary, 3. Liste mit Tupeln
 1. `<expr>.subs(alt, neu)`
 2. `<expr>.subs({alt1: neu1, alt2: neu2, ...})`
 3. `<expr>.subs([(alt1, neu1), (alt2, neu2), ...])`
- Vorteil von Variante 3: beachtet Reihenfolge
`<expr>.subs([(a, b), (b, c)])`
- Wichtig: `subs(...)` → Rückgabewert (ursprünglicher Ausdruck 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)`: Gleichung und -systeme lösen
- `<expr>.atoms()`, `<expr>.atoms(sp.sin)`: „Atome“ (bestimmten Typs)
- `<expr>.args`: Argumente der jeweiligen Klasse (Summanden, Faktoren, ...)

- `sp.simplify(...)`: 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, 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
```

```
f_xa_fnc = sp.lambdify((a, b, x), f, modules='numpy')
```

```
# Funktion auswerten
```

```
print f_xa_fnc(1.2, 0.5, 3.14)
```

Doku und Links

- Doc-Strings der einzelnen Funktionen (meist ausreichend)
- <http://docs.sympy.org/dev/tutorial.html>
- <http://docs.sympy.org/dev/gotchas.html> (Fallstricke)

Zusammenfassung

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