Einführung in Sage - Einheit 2

Grundlagen, Symbolisches Rechnen, Gleichungen

Jochen Schulz

Georg-August Universität Göttingen

Aufbau

1. **Grundlagen**
   - Sage
   - Python
2. **Symbolisches Rechnen I**
3. **Gleichungen**

Aufbau

Aufbau

Beispiel

Betrachte:

```
f = x^2-3*x-18
```

- Wie geht Sage mit der Unbekannten x um?
- Welchen Datentyp hat f?
- Was kann ich mit f machen?

Bezeichner

- Bezeichner sind Namen, wie z.B. x oder f. Sie können im mathematischen Kontext sowohl Variablen als auch Unbestimmte repräsentieren.
- Bezeichner sind aus Buchstaben, Ziffern und Unterstrich _ zusammengesetzt.
- Sage unterscheidet zwischen Groß- und Kleinschreibung.
- Bezeichner dürfen nicht mit einer Ziffer beginnen

Beispiele

- zulässige Bezeichner: `x`, `f`, `x23`, `_x_1`
- unzulässige Bezeichner: `12x`, `p~`, `x>y`, `Das System`

Wert eines Bezeichners

- Der Wert eines Bezeichners ist ein Objekt eines bestimmten Datentyps.
- Ein Datentyp ist durch seine Eigenschaften gegeben.
  Beispiel: Natürliche Zahlen, rationale Zahlen, Bezeichner, Zeichenketten, ...
- Ein Objekt ist eine Instanz (Einheit) eines Datentyps.

Zuweisungsoperator =

- `bez=wert`: Zuweisung des Wertes `wert` zu dem Bezeichner `bez`.
- `func(arg)=expr(arg)`: Definition der Funktion `func` mit dem Argument `arg` und Zuweisung des Ausdrucks `expr` zu (abhängig von `arg`)
- **Warnung:** Unterscheiden Sie stets zwischen dem Zuweisungsoperator = und dem logischen Operator ==.
- `reset('bezeichner')`: Löschen von Zuweisungen/Variablen.

Beispiele: Zuweisung

```
N=6; N
```

```
6
```

```
x,y = var('x,y'); f = x+2*x*x-y; g(x) = x^2; f,g
```

```
(2*x^2 + x - y, x |--> x^2)
```

```
x=pi;y = cos(x); x,y
```

```
(pi, -1)
```

Auswertung

- Der Bezeichner ist der Name einer Unbekannten.
- Die Auswertung eines Bezeichners erfolgt ohne die Benutzung von bekannten Zuweisungen.
- Der Wert bezeichnet die Auswertung zum Zeitpunkt der Zuweisung.

Beispiele: Auswertung

```
var('a') ; f(x) = x*x-3*x-a
```

```
x |--> x^2 - a - 3*x
```

```
f(a=2)
```

```
x^2 - 3*x - 2
```

```
f(1)
```

```
-a - 2
```

```
f(1,a=2)
```

```
-4
```

Einige Datentypen

Typ	Bedeutung	Beispiel
`integer`	ganze Zahlen	`-3,0,100`
`rational`	rationale Zahlen	`7/11`
`float`	Gleitpunktzahl	`0.123`
`complex`	komplexe Zahlen	`complex(1,3)`
`expression`	symbolische Ausdrücke	`x+y`
`bool`	logische Werte: true/false	`bool(1<2)`

Beispiele für Datentypen

```
type(5)
```

```
<type 'sage.rings.integer.Integer'>
```

```
f = x^2-3*x-18; type(f)
```

```
<type 'sage.symbolic.expression.Expression'>
```

```
type(x)
```

```
<type 'sage.symbolic.expression.Expression'>
```

```
f+f
```

```
2*x^2 - 6*x - 36
```

Operatoren

- Typische Operatoren sind `+`,`-`,`*`,`/`,...
- In Sage werden Objekte immer durch Funktionen miteinander verbunden. Operatoren sind äquivalent zu Funktionen.
- Kombination verschiedener Operatoren: Die Regeln der Bindungsstärke gelten (Punktrechnung vor Strichrechnung); Die Ordnung kann durch Klammersetzung geändert werden.

Wichtige mathematische Operatoren

Operator/Funktion	Erklärung
`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`/`	Division
`^`	Potenz
`%`	Rest bei Division
`factorial()`	Fakultät

Zerlegen von Ausdrücken

- Bausteine von Ausdrücken heißen Operanden.
- `Ausdruck.nops()`: Anzahl der Operanden.
- `Ausdruck.operands()`: Operanden selbst.
- `Ausdruck.has(a)`: Untersuchung ob a ein Operand von `Ausdruck` ist.
- Die Befehle beziehen sich jeweils auf die automatisch vereinfachten Objekte.

Beispiele II

```
_=var('z,y');f = x*z+3*x+sqrt(y)
f.operands(),(f.operands())[1],((f.operands())[1]).nops()
```

```
([x*z, 3*x, sqrt(y)], 3*x, 2)
```

```
f.has(z), f.has(6)
```

```
(True, False)
```

Automatische Vereinfachung

Sage führt oft automatische Vereinfachungen durch. Ansonsten muß der Benutzer gezielt Vereinfachungen anfordern.

```
sin(15*pi), exp(0)
```

```
(0, 1)
```

```
2*Infinity-5
```

```
+Infinity
```

```
y = (-4*x+x^2+4)*(7*x+x^2+12); y
```

```
(x^2 - 4*x + 4)*(x^2 + 7*x + 12)
```

```
y.full_simplify()
```

```
x^4 + 3*x^3 - 12*x^2 - 20*x + 48
```

Aufbau

1. **Grundlagen**
   - Sage
   - Python
2. **Symbolisches Rechnen I**
3. **Gleichungen**

Ausgabe von Variablen in Texten

```
print "Text %<format> und %<format> ... " % (x,y,...)}
```

- <format> kann z.B. `i` für `integer` oder `f` für `float` sein
- Es kann beliebig viele format-Platzhalter geben

Beispiele:

```
x = 4;y = 6
print "x ist %i und y ist %i" % (x,y)
```

```
x ist 4 und y ist 6
```

```
x = [var("x%i" % k) for k in [1..3]]; x
```

```
[x1, x2, x3]
```

Wörterbücher (Dictionaries)

- Dictionaries bestehen aus Assoziationen der Form `<Index>:<Wert>`.
- Definition:

```
d = {<Index1>:<Wert1>,<Index2>:<Wert2>,...}
```

- Zugriff:

```
d[<Index>]
```

- **Beispiel:**

```
var('x,y,z');d = {x:21,y:5,z:42==y}
d[x],d[z],d[z].rhs()
```

```
(21, 42 == y, y)
```

Anonyme Funktionen

- Anonyme Funktionen sind Funktionen die keinen aufrufbaren Namen besitzen (Sie sind ein Funktionen-Objekt)
- Dies ist sinnvoll innerhalb von Konstrukten oder Aufrüfen anderer Funktionen.
- Syntax:

```
lambda <parameter_list>: <expression>
```

- **Beispiel**

```
menge = [1,2,3,4,5]
map(lambda x: x^2,menge)
```

```
[1, 4, 9, 16, 25]
```

Funktionen / Objekt-Methoden

- Objekte besitzen Funktionen, sogenannte Objekt-Methoden.
- Ein Objekt kennt alle auf sich selbst anwendbaren Funktionen; Nur einige Methoden sind als globale Funktionen benutzbar (wie z.B. `map()`)
- Beispiel

```
f = 1 +x -x^2; f.operands(); operands(f)
```

```
[-x^2, x, 1]
Traceback (click to the left of this block for
    traceback)
...
NameError: name 'operands' is not defined
```

normale Schleifen - `for`

Bereits bekannte Schleifen: `[.. for .. in ..]`-Konstrukt. Die `for`-Schleife kann auch ganze Blöcke wiederholen.

```
for <indexvar> in <expr>/<list>:
    <Code-Block>
else:
    <Code-Block>
```

Beispiel:

```
x = 2; y = 2
for k in [1..4]:
    x = x +k
    y = y^k
x,y
```

```
(12, 16777216)
```

Wichtig: In Python/Sage ist solch ein Code-Block dadurch gekennzeichnet, dass er mit einem **:** eingeleitet wird und um ein TAB eingerückt ist!

Abfragen - `if/else`

einfache Abfragen mittels `if`. Diese können für ganze Blöcke genutzt werden:

```
if <boolean expr>:
    <Code-Block>
```

Beispiel:

```
x = 3; y = 2
if x > y :
    z = x
else:
    z = y
z
```

3

Aufbau

Verbinden von Ausdrücken

Ausdrücke können beliebig addiert, subtrahiert, multipliziert und dividiert werden.

Beispiel:

- Definition

```
var('x,y'); f = x*x+3*x+y; g = x-y
```

- Potenz

```
f^g
```

```
(x^2 + 3*x + y)^(x - y)
```

Verbinden von Ausdrücken II

- Addition / Subtraktion

```
f+g, f-g
```

```
(x^2 + 4*x, x^2 + 2*x + 2*y)
```

- Multiplikation/ Division

```
f*g, f/g
```

```
((x - y)*(x^2 + 3*x + y), (x^2 + 3*x + y)/(x - y))
```

Sortieren und Zusammenfassen von Ausdrücken

`a.collect(<unknown>)`: der Ausdruck `a` wird bzgl. der `unknown` Variablen sortiert.

Beispiel:

```
f = a*x^2+a*x+x^3+sin(x)+b*x+4*x+x*sin(x):
f.collect(x)
```

```
a*x^2 + x^3 + (a + b + sin(x) + 4)*x + sin(x)
```

```
f.collect(x*sin(x))
```

```
a*x^2 + x^3 + a*x + b*x + x*sin(x) + 4*x + sin(x)
```

`a.combine()`: der Ausdruck wird durch die Potenzgesetze zusammengefaßt.

Beispiel:

```
g = x^(a)*x^(b)
g.combine()
```

```
x^(a + b)
```

Ausmultiplizieren - expand()

a.expand(): Ausmultiplizieren.

a.expand_trig(): Ausmultiplizieren mit Nutzung der trigonometrischen Eigenschaften.

Beispiel:

```
expand((x+2)^4)
```

```
x^4 + 8*x^3 + 24*x^2 + 32*x + 16
```

```
(sin(x+y)).expand_trig()
```

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

Ausmultiplizieren bei Gleichungen

```
a = (16*x-13)^2 == (3*x+5)^2/2
a.expand()
```

```
256*x^2 - 416*x + 169 == 9/2*x^2 + 15*x + 25/2
```

```
a.expand('left')
```

```
256*x^2 - 416*x + 169 == 1/2*(3*x + 5)^2
```

```
a.expand('right')
```

```
(16*x - 13)^2 == 9/2*x^2 + 15*x + 25/2
```

Faktorisieren - `factor()`

`factor(Ausdruck)`: Faktorisieren von Ausdrücken.

- Sage faktorisiert nur, wenn die resultierenden Koeffizienten rationale Zahlen sind.
- Bei rationalen Funktionen wird ein gemeinsamer Hauptnenner gesucht.

Beispiel:

```
factor(x^2-2), factor(x^2-9/4)
```

```
(x^2 - 2, 1/4*(2*x - 3)*(2*x + 3))
```

```
factor(2 - 2/(x^2-1))
```

```
2*(x^2 - 2)/((x - 1)*(x + 1))
```

Partialbruchzerlegung - partial_fraction()

f.partial_fraction(): Zerlegung eines rationalen Ausdruck f in eine Summe rationaler Terme (Zählergrad < Nennergrad) (Partialbruchzerlegung).

Beispiel:

```
f = x^2/(x^2-1); f.partial_fraction()
```

```
1/2/(x - 1) - 1/2/(x + 1) + 1
```

```
f = (x^2+2*x+3)/(x^3+4*x^2+5*x+2); f
```

```
(x^2 + 2*x + 3)/(x^3 + 4*x^2 + 5*x + 2)
```

Vereinfachen - `simplify`

`f.simplify()`: simple Vereinfachung (Potenzen). `f.simplify_<target>()`: vereinfacht den Ausdruck f mit der Methode `<target>`. Mögliche targets sind

- `trig`: trigonometrisch
- `rational`: rational
- `radical`: log/ln/exp
- `factorial`: Nutzung der Fakultät
- `full`: alle Vereinfachungen

Beispiel:

```
(2 - 2/(x^2-1)).simplify_rational()
```

```
2*(x^2 - 2)/(x^2 - 1)
```

Beispiele - Simplify I

```
f = x/(x+y)+y/(x+y)-sin(x)^2-cos(x)^2
f.simplify_rational()
```

```
-sin(x)^2 - cos(x)^2 + 1
```

```
g = sqrt(997)-(997^3)^(1/6)
g.simplify()
```

```
0
```

Beispiele - Simplify II

```
(tan(x)).simplify_trig()
```

```
sin(x)/cos(x)
```

```
a = (2^(1/3)+4^(1/3))^3-6*(2^(1/3) + 4^(1/3))-6
a.simplify_full()
```

```
0
```

Aufbau

Gleichungen

- lineares Beispiel

```
var('x,y')
Gleichungen = [x+y == 1, x-y == 1]
solve(Gleichungen,x,y)
```

```
[[x == 1, y == 0]]
```

- nichtlineares Beispiel

```
Gleichungen1 = [x+y == 1,(x-y)^2 == 1]
solve(Gleichungen1,x,y)
```

```
[[x == 0, y == 1], [x == 1, y == 0]]
```

Vergleiche/Logik

- Operator `==`: Überprüfung zweier Objekte auf Gleichheit.
  `a==b` ist wahr (richtig), wenn *a* und *b* die gleichen Auswertungen besitzen (und vom gleichen Typ sind).
- `bool(Ausdruck)`: Überprüfung von Aussagen. Ergebnis: `True` oder `False`.
- Operator `<>`: Überprüfung zweier Objekte auf Ungleichheit.
  `a<>b` ist `True`, falls *a* nicht gleich *b* ist.

Beispiele - Vergleiche I

```
bool(4-3==1)
```

```
True
```

```
bool(4*x==x); x=0; bool(4*x==x)
```

```
False
True
```

```
bool(x==0); bool(x<>0)
```

```
True
False
```

Beispiele - Vergleiche II

```
bool(0.5==1/2)
```

```
True
```

```
type(0.5); type(1/2)
```

```
<type 'sage.rings.real_mpfr.RealLiteral'>
<type 'sage.rings.rational.Rational'>
```

Verknüpfungen

- Logisches 'und' (`and`)
- Logisches 'oder' (`or`)
- Logisches 'nicht' (`not`)

Beispiel:

```
true and false, true or false, not true
```

```
(False, True, False)
```

- Logiktabelle:

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

Lösen von Gleichungssystemen

```
solve(<gleichungen>,<variablen>,<optionen>)
```

- `<gleichungen>`: System von Gleichungen.
- `<variablen>`: Danach wird aufgelöst.
- `<optionen>`:
  - `multiplicities=True`: Vielfachheit der Lösungen wird mit ausgegeben.
  - `solution_dict=True`: Lösung wird als Dictonary zurück gegeben.
- Bei einzelnen Gleichungen wird der Lösungswert zurückgegeben. Bei mehreren Gleichungen wird ein System äquivalenter Gleichungen zurückgegeben.

Beispiele - Solve I

```
solve(x^2+x == y/4,x)
```

```
[x == -1/2*sqrt(y + 1) - 1/2, x == 1/2*sqrt(y + 1) -
    1/2]
```

```
f = (x-1)^5*(x^2+1)
solve(f == 0, x)
```

```
[x == -I, x == I, x == 1]
```

```
solve(f == 0, x, multiplicities=True)
```

```
([x == -I, x == I, x == 1], [1, 1, 5])
```

Beispiele - Solve II

```
assume(x>0); solve(x^2+x == y/4,y)
```

```
[y == 4*x^2 + 4*x]
```

```
solve([x^2-y^2 == 0],[x,y])
```

```
([x == -y, x == y], [1, 1])
```

```
solve([x^2-y^2 == 0, x+y == 1],x,y)
```

```
[[x == (1/2), y == (1/2)]]
```

Beispiele - Solve III

```
sol = solve([x^2-y^2 == 0, x+y == 1],x,y, solution_dict=
    True); sol
```

```
[{y: 1/2, x: 1/2}]
```

```
sol[0][x], sol[0][y]
```

```
(1/2, 1/2)
```

numerisches Lösen von Gleichungen (1D)

```
f.find_root(a,b)
```

Finden von Nullstellen der Gleichung f im Intervall $[a, b]$.

Beispiel:

```
(x == sin(x)).find_root(-2,2)
```

```
0.0
```