Einführung in Sage

Einheit 2

Jochen Schulz

Georg-August Universität Göttingen

16. Januar 2010

Aufbau

Aufbau

1. **Grundlagen von Sage**
2. Symbolisches Rechnen I
3. Gleichungen

Ein erstes Beispiel

```
>> f = x^2-3*x-18; solve(f==0,x)
```

```
[x == 6, x == -3]
```

```
>> solve(f<=0,x) ??
```

```
 [-3, 6] union
 {I x + 3/2 | x in (-infinity, infinity)}
```

```
>> assume(x,Type::Real): solve(f<=0,x) ??
```

```
 [-3, 6]
```

```
>> x:=1: f  ??
```

```
 -20
```

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 für Bezeichner

- 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 $:=$

- Die Operation `bez=wert` weist dem Bezeichner `bez` den Wert `wert` zu.
- Beispiele: `N=5`, `f = x^2-3*x-18`
- Rückgabeparameter ist die rechte Seite (Eine Ausgabe erfolgt jedoch normalerweise nicht)
- Warnung: Unterscheiden Sie stets zwischen dem Zuweisungsoperator $=$ und dem logischen Operator $==$.

Beispiele

```
>> N=6; N
```

```
  6
```

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

```
    2*x^2 + x - y
```

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

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

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
```

Einige Datentypen

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

Befehle im Umgang mit $=$

- Löschen von Zuweisungen: `reset('bezeichner')`

Beispiel: Auswertung

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

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

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

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

```
>> f(1)
```

```
  -a - 2
```

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

```
  -4
```

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.

Operatoren

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

Wichtige mathematische Operatoren

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

Zerlegen von Objekten

- Viele Objekte sind zusammengesetzt. Ihre Bausteine heißen Operanden. coeff arguments has nops() number_of_operands() operands() operator()

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

Verbinden von Ausdrücken

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

- 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))
```

collect()

Durch `a.collect(Unbestimmte)` wird der Ausdruck a bzgl. der `Unbestimmten` sortiert.

```
>> 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)
```

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

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

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

expand()

Ausmultiplizieren von Ausdrücken erfolgt durch `a.expand()` und `a.expand_trig()`.

```
>> 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)
```

expand() 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
```

factor()

Der Befehl `factor(Ausdruck)` faktorisiert Polynome und Ausdrücke.

- Sage faktorisiert nur, wenn die resultierenden Koeffizienten rationale Zahlen sind.
- Auch anwendbar auf rationale Funktionen. Es wird ein gemeinsamer Hauptnenner gesucht.

```
>> 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))
```

simplify_rational()

Durch `f.simplify_rational()` wird eine 'Normalform' eines rationalen Ausdrucks `f` erzeugt.

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

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

mehr simplify ??

partial_fraction()

Durch `a.partial_fraction()` wird ein rationaler Ausdruck in eine Summe rationaler Terme zerlegt, in denen jeweils der Zählergrad kleiner als der Nennergrad ist. (Partialbruchzerlegung)

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

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

simplify

Durch `rewrite(Ausdruck, Option)` wird versucht, den `Ausdruck` so umzuformen, das gewisse Funktionen aus dem Ausdruck eliminiert werden.

- Beispielsweise können `sin` und `cos` immer durch `tan` ausgedrückt werden (Option: `tan`).
- Optionen sind `diff`, `exp`, `fact`, `gamma`, `heavyside`, `ln`, `sign`, `sincos`, `sinhcosh`, `tan`.
- Man versucht die Ausdrücke mit Hilfe der in der Option genannten Funktion(en) auszudrücken.

Beispiele - simplify_trig()

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

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

Simplify ??

- Durch `simplify(f,target)` wird versucht den Ausdruck f zu vereinfachen. Optional können durch `target` spezielle Vereinfachungen angefordert werden.
- Mögliche `targets` sind `exp`, `ln`,`cos`, `sin`, `sqrt`, `logic` und `relation`.
- Die Optionen `logic` und `relation` dienen zur Vereinfachung von logischen Ausdrücken bzw. von Gleichungen und Ungleichungen.

Beispiele: Simplify I

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

```
-sin(x)^2 - cos(x)^2 + x/(x + y) + y/(x + y)
```

Beispiele: Simplify II ??

westers paper ?

```
>> g:= sqrt(4+2*sqrt(3)):
```

```
>> g=simplify(g,sin)
```

```
 1/2    1/2       1/2     1/2    1/2       1/2
2     (3     + 2)     = 2     (3     + 2)
```

```
>> g=simplify(g)
```

```
    1/2    1/2       1/2     1/2           1/2
  (2     (3     + 2)     = 3     + 1) = 3     + 1
```

Aufbau

Gleichungen

- lineares Beispiel

```
>> Gleichungen := {x+y = 1, x-y = 1}:
>> solve(Gleichungen)
```

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

- nichtlineares Beispiel

```
>> Gleichungen1:={x+y=1,(x-y)^2=1}:
>> solve(Gleichungen1)
```

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

Vergleiche

- Der Operator `=` vergleicht zwei Objekte.
- `a=b` ist wahr (richtig), wenn *a* und *b* die gleichen Auswertungen besitzen (und vom gleichen Typ sind).
- Zur Überprüfung von Aussagen gibt es die Funktion `bool(Ausdruck)`. Sie liefert als Ergebnis `TRUE` oder `FALSE`.
- Die inverse Operation zu `'='` ist `'<>'`, also `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)
```

```
  FALSE
```

```
>> domtype(1.0), domtype(1)
```

```
  DOM_FLOAT, DOM_INT
```

Solve

- Solve ist der universelle Befehl zum Lösen von Gleichungen und Ungleichungen und auch Differentialgleichungen.
- Der Befehl ist von der Form `solve(Gleichungen,Variablen)`.
- `Gleichungen` kann ein System von Gleichungen sein.
- `Variablen` gibt an, wonach aufgelöst wird.
- Bei einzelnen Gleichungen wird der Lösungswert zurückgegeben. Bei mehreren Gleichungen wird ein System äquivalenter Gleichungen zurückgegeben.
- Weitere Optionen werden später erklärt.

Beispiele - Solve I

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

```
  {             1/2                         1/2         }
  {    (y + 1)                     (y + 1)              }
  { - ---------- - 1/2, ---------- - 1/2 }
  {        2                           2                }
```

Beispiele - Solve II

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

```
              2
  {4 x + 4 x }
```

```
>> solve({x^2-y^2=0},{x,y})
```

```
  {[x = z, y = z], [x = -z, y = z]}
```

```
>> solve({x^2-y^2=0,x+y=1},{x,y})
```

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