Sage 2: Formales Rechnen, Gleichungen, Polynome

14.11.2018

Inhaltsverzeichnis

Rechnen mit formalen Ausdrücken

Um mit formalen Ausdrücken zu rechnen, muß man entsprechende Variablennamen deklarieren, ausgenommen die Variable `x`

```
x
```

```
x
```

```
parent(x)
```

```
Symbolic Ring
```

```
x^2+4
```

```
x^2 + 4
```

Formale Ausdrücke können natürlich auch Variablen zugeordnet werden.

```
a=x^3
a
```

```
x^3
```

Man mache sich den prinzipiellen Unterschied zwischen einer "Variable" im Sinn der Informatik (hier z.B. die Variable `a`) und der symbolischen "Variable" `x` klar: `a` bezeichnet einen Platz im Speicher des Computers, in dem ein gewisser Wert abgelegt ist, und dieser Wert hat einen Typ. Dagegen ist `x` hier eine Variable im Sinne der Analysis, d.h., ein Platzhalter, für den verschiedene Werte eingesetzt werden *können*.

Nicht die Variable a hat einen festen Typ, sondern ihr Wert; mit anderen Worten, die Typen sind *dynamisch*.

```
parent(a)
```

```
Symbolic Ring
```

```
a=1
a
```

```
1
```

```
parent(a)
```

```
Integer Ring
```

alle formalen Symbole außer x müssen als solche deklariert werden.

```
y
```

```
Traceback (click to the left of this block for traceback)
...
NameError: name 'y' is not defined
```

```
var('y')
y
```

```
y
```

```
parent(y)
```

```
Symbolic Ring
```

das kann auch gruppenweise passieren.

```
var('u,v')
```

```
(u, v)
```

```
parent(u)
```

```
Symbolic Ring
```

Formale Ausdrücke werden vorerst nicht ausmultipliziert

```
f=(x+y)^2
f
```

```
(x + y)^2
```

das muß explizit angefordert werden.

```
g=expand(f)
g
```

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

alternativ kann die Methode `expand` direkt angefordert werden.

```
f.expand()
```

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

dabei wird der ursprüngliche Ausdruck nicht verändert

```
f
```

```
(x + y)^2
```

Die inverse Operation zum ausmultiplizieren ist **factor**. Zum Vergleich: Wenn man **factor** auf eine ganze Zahl anwendet, wird trotz gleichen Namens der Funktion eine ganz andere Methode aufgerufen:

```
factor(g)
```
```
    (x + y)^2
```
```
factor(442)
```
```
    2 * 13 * 17
```

- Mit formalen Ausdrücken können alle herkömmlichen symbolischen Operationen durchgeführt werden. Es können alle herkömmlichen Ausdrücke differenziert werden.

```
diff(sin(x),x)
```
```
    cos(x)
```

Integration ist bekanntlich nicht immer möglich. Für sogenannte elementare Funktionen (Polynome, rationale Funktionen, exp, log, sin, cos und alles, was daraus erzeugt werden kann, sowie algebraische Funktionen können mit dem Risch-Algorithmus (1967) gelöst werden. Wenn eine gegebene elementare Funktion eine elementare Stammfunktion hat, kann sie berechnet werden, wenn die Stammfunktion nicht elementar ist, dann erkennt es der Algorithmus und liefert eine entsprechende Fehlermeldung.

```
integrate(sin(x),x)
```
```
    -cos(x)
```
```
integrate(sin(x),x,0,1)
```
```
    -cos(1) + 1
```
```
integrate(sin(exp(sqrt(x))),x)
```
```
    integrate(sin(e^sqrt(x)), x)
```

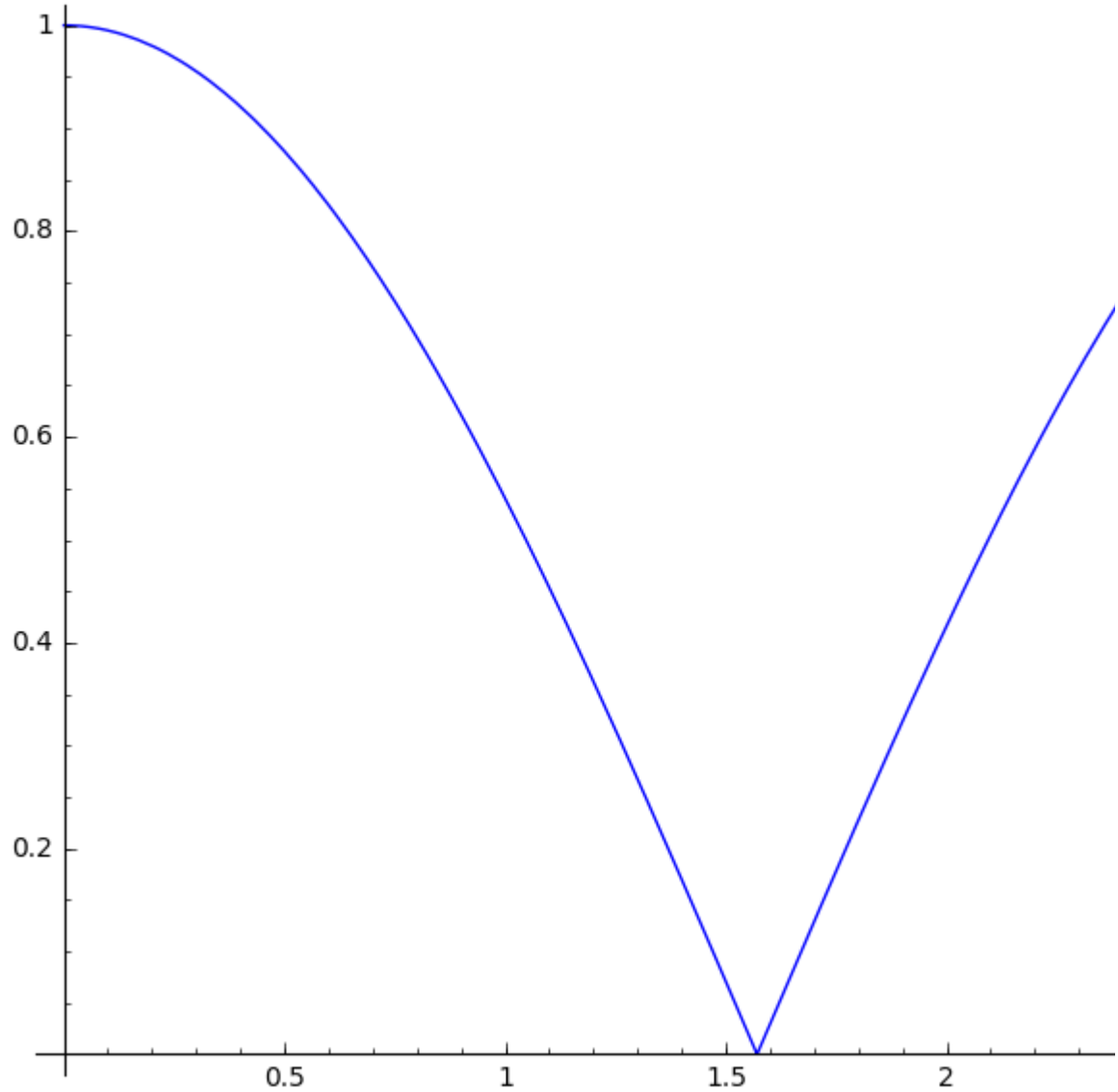
Einige nicht-elementare Funktionen können dennoch mit heuristischen Methoden berechnet werden.

```
integrate(exp(-x^2),x)
```
```
    1/2*sqrt(pi)*erf(x)
```

Allerdings begibt man sich in Teufels Küche: Während der Risch-Algorithmus bewiesenermaßen korrekt ist, können heuristische Methoden alle möglichen Fehler liefern. Darüberhinaus sind gewisse Fragen formal *nicht entscheidbar*, d.h., es gibt keinen Algorithmus, der alle Probleme lösen kann, wenn neben elementaren Funktionen auch z.B. die Betragsfunktion erlaubt wird (Satz von Richardson, 1968).

Wir betrachten das folgende harmlose, aber falsche Integral.

```
integrate(abs(cos(x)),x,0,pi)
```
```
    -1
```
```
plot(abs(cos(x)),x,0,pi)
```



Variablen können mit **reset** zurückgesetzt werden.

```
reset('y')
y
```

```
Traceback (click to the left of this block for traceback)
...
NameError: name 'y' is not defined
```

Grenzwerte

Unendlich wird mit der Variable `oo` simuliert.

```
var('n')
limit(1/n, n=oo)
```

```
0
```

```
limit((1+x/n)^n, n=oo)
```

```
e^x
```

Beim Berechnen von Grenzwerten ist ebenfalls höchste Vorsicht geboten. Der folgende Limes verlangt eigentlich die Regel von De l'Hospital, das wird aber nicht erkannt.

```
var('w')
g = w/(sqrt(1+w)*sin(x)^2 + sqrt(1-w)*cos(x)^2-1)
g.show()
```

$$\frac{w}{\sqrt{-w+1}\cos\left(x\right)^2 + \sqrt{w+1}\sin\left(x\right)^2 - 1}$$

```
limit(g, w=0)
```

```
0
```

Das Problem besteht darin, daß die trigonometrischen Rechenregeln nicht automatisch angewandt werden.

```
h = sin(x)^2+cos(x)^2 -1
h.show()
```

$$\cos\left(x\right)^2 + \sin\left(x\right)^2 - 1$$

```
h.is_zero()
```

```
True
```

Oft hängt die Integrierbarkeit von Zusatzannahmen ab.

```
var('a')
integral(exp(-a*x), (x, 1, oo))
```

```
Traceback (click to the left of this block for traceback)
...
Is a positive, negative or zero?
```

```
assume(a>0)
integral(exp(-a*x), (x, 1, oo))
```

```
e^(-a)/a
```

```
assume(a<0)
```

```
Traceback (click to the left of this block for traceback)
...
ValueError: Assumption is inconsistent
```

```
forget(a>0)
assume(a<0)
integral(exp(-a*x), (x, 1, oo))
```

```
Traceback (click to the left of this block for traceback)
...
```

```
ValueError: Integral is divergent.
```

Funktionsgraphen

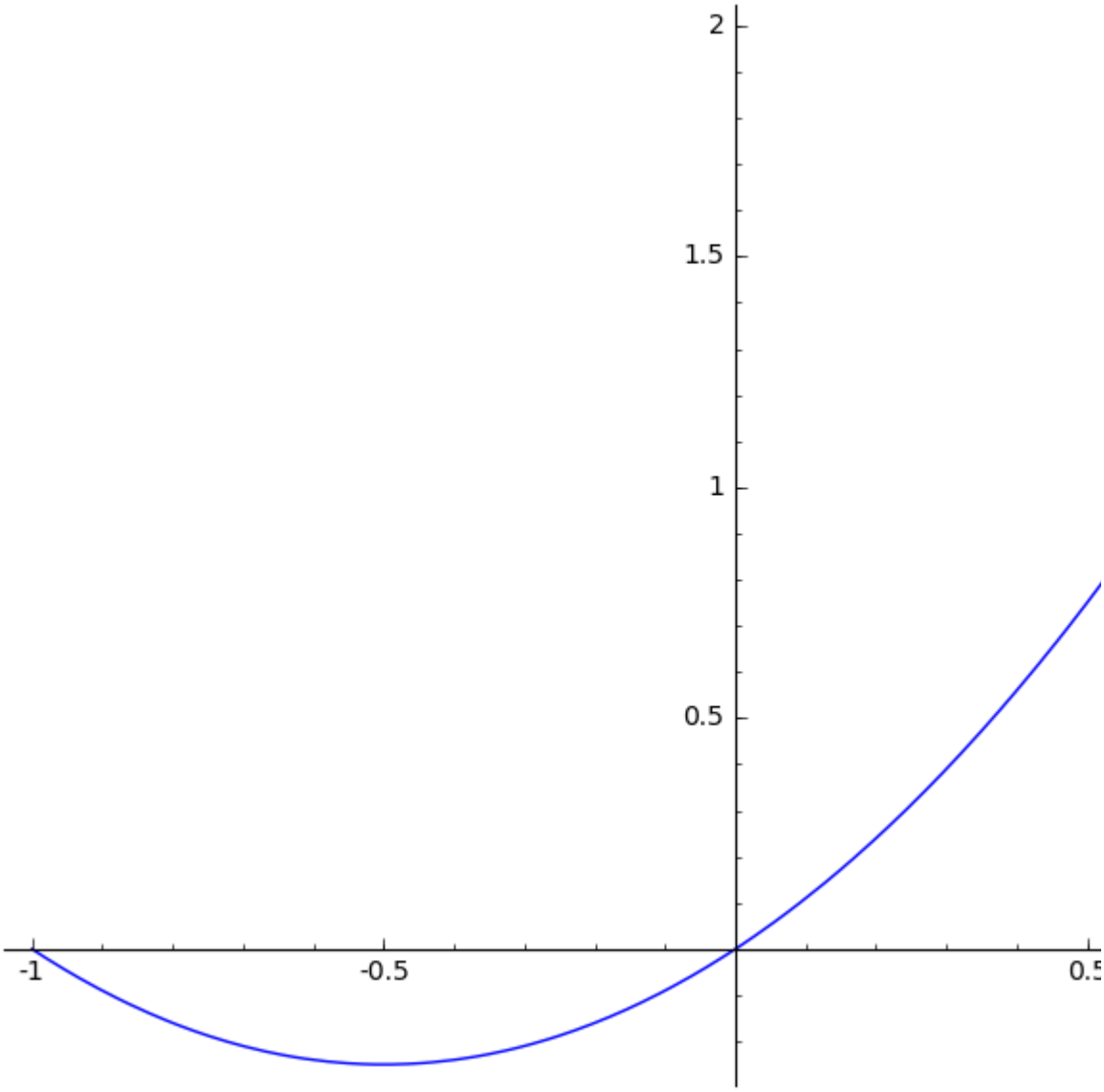
```
f(x) = x^2+x
f
```

```
x |--> x^2 + x
```

```
diff(f)
```

```
x |--> 2*x + 1
```

```
plot(f,-1,1)
```




```
g(x,y) = y^2+sin(x)
g
```

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

```
plot3d(g,(-1,1),(-1,1))
```

Lösung von Gleichungen

Polynomgleichungen können bis zum Grad 4 explizit gelöst werden.

```
var('a,b,c')
solve(a*x^2+b*x+c==0,x)
```

```
[x == -1/2*(b + sqrt(b^2 - 4*a*c))/a, x == -1/2*(b - sqrt(b^2
4*a*c))/a]
```

```
solve(x^2-2 == 0, x)
```

```
[x == -sqrt(2), x == sqrt(2)]
```

Allerdings sind die sogenannten *Formeln von Cardano* recht unhandlich:

```
solve(x^3-x+2==0, x)
```

```
[x == -1/2*(1/9*sqrt(26)*sqrt(3) - 1)^(1/3)*(I*sqrt(3) + 1) -
1/6*(-I*sqrt(3) + 1)/(1/9*sqrt(26)*sqrt(3) - 1)^(1/3), x ==
-1/2*(1/9*sqrt(26)*sqrt(3) - 1)^(1/3)*(-I*sqrt(3) + 1) -
1/6*(I*sqrt(3) + 1)/(1/9*sqrt(26)*sqrt(3) - 1)^(1/3), x ==
(1/9*sqrt(26)*sqrt(3) - 1)^(1/3) + 1/3/(1/9*sqrt(26)*sqrt(3)
```

```
    1)^(1/3)]
```

```
show(_[0])
```

$$x = -\frac{1}{2}\left(\frac{1}{9}\sqrt{26}\sqrt{3}-1\right)^{\frac{1}{3}}(i\sqrt{3}+1) - \frac{-i\sqrt{3}+1}{6\left(\frac{1}{9}\sqrt{26}\sqrt{3}-1\right)^{\frac{1}{3}}}$$

Für Polynome von höherem Grad gibt es keine Lösungsformeln.

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

```
    [0 == x^5 - x + 2]
```

Gleichungen mit transzendenten Funktionen können nur in wenigen Spezialfällen explizit gelöst werden.

```
solve(sin(x)==0, x)
```

```
    [x == 0]
```

Wenn alle Lösungen gewünscht sind, muß das explizit angefordert werden.

```
solve(sin(x)==0, x, to_poly_solve='force')
```

```
    [x == pi*z30]
```

`z30` ist eine automatisch erzeugte formale Variable. Das Ergebnis ist zu interpretieren als Menge aller ganzzahligen Vielfachen von Pi.

Einfache Ungleichungen können ebenfalls gelöst werden.

```
solve(1/(x-1)<8, x)
```

```
    [[x < 1], [x > (9/8)]]
```

```
solve(abs(x)<1, x)
```

```
    #0: solve_rat_ineq(ineq=abs(_SAGE_VAR_x) < 1)
    [[-1 < x, x < 1]]
```

Der Algorithmus stößt aber rasch an Grenzen.

```
solve(abs(cos(x))<1/2, x)
```

```
    #0: solve_rat_ineq(ineq=abs(cos(_SAGE_VAR_x)) < 1/2)
    [[4*cos(x) + 2 > 0, -4*cos(x) + 2 > 0]]
```

Polynome

Wenn man von vornherein weiß, daß man es nur mit Polynomen zu tun hat, ist es effizienter, in einem Polynomring zu arbeiten. Wir betrachten den Ring der Polynome in der einer Variable über dem Körper der rationalen Zahlen

```
QQ
```

```
Rational Field
```

```
Px = QQ[x]
Px
```

```
Univariate Polynomial Ring in x over Rational Field
```

Allerdings weiß das Symbol x nichts von diesem Ring

```
parent(x)
```

```
Symbolic Ring
```

Dazu muß es explizit umgewandelt werden.

```
x = Px(x)
parent(x)
```

```
Univariate Polynomial Ring in x over Rational Field
```

Achtung, die Variable x (im Sinn der Informatik) und das Symbol x sind nicht das gleiche. Im vorherigen Ausdruck ist `x` nach wie vor Element des `Symbolic Ring`

```
h
```

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

```
parent(h)
```

```
Symbolic Ring
```

Gegebenenfalls wird x automatisch in den `Symbolic Ring` zurückgeholt.

```
cos(x)
```

```
cos(x)
```

```
parent(_)
```

```
Symbolic Ring
```

Der *Zeiger* `x` bleibt davon aber unberührt.

```
parent(x)
```

```
Univariate Polynomial Ring in x over Rational Field
```

Man kann die obigen Operationen (Erzeugung des Polynomrings und Zuweisung der Variablen) auch in einem Schritt durchführen

```
Py.<y> = QQ[]
Py
```

```
Univariate Polynomial Ring in y over Rational Field
```

```
parent(y)
```

```
Univariate Polynomial Ring in y over Rational Field
```

Das gleiche in mehreren Variablen

```
P2.<u,v> = QQ[]
P2
```

```
Multivariate Polynomial Ring in u, v over Rational Field
```

```
parent(u)
```

```
Multivariate Polynomial Ring in u, v over Rational Field
```

```
parent(v)
```

```
Multivariate Polynomial Ring in u, v over Rational Field
```

Dabei sind die Ringe strikt getrennt.

```
u+y
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: unsupported operand parent(s) for +: 'Multivariate
Polynomial Ring in u, v over Rational Field' and 'Univariate
Polynomial Ring in y over Rational Field'
```

Im Ring der Polynome sind immer alle Nullstellen berechenbar.

```
p= x^2-3
p
```

```
x^2 - 3
```

```
parent(p)
```

```
Univariate Polynomial Ring in x over Rational Field
```

Allerdings nur im zugrundeliegenden Ring.

```
p.roots()
```

```
[]
```

Wenn in anderen Ringen gesucht werden soll, muß dies explizit angegeben werden.

```
p.roots(ring=RR)
```

```
[(-1.73205080756888, 1), (1.73205080756888, 1)]
```

Eine Zahl heißt **algebraisch** wenn sie Nullstelle eines Polynoms mit rationalen Koeffizienten ist. Die algebraischen Zahlen bilden einen Körper und können beliebig genau approximiert werden, das wird von sage durch das Fragezeichen am Ende der Dezimalentwicklung angedeutet.

```
p.roots(ring=QQbar)
```

```
[(-1.732050807568878?, 1), (1.732050807568878?, 1)]
```

Das Polynom kann auch in den `Symbolic Ring` eingebettet werden.

```
pe = SR(p)
pe
```

```
x^2 - 3
```

```
parent(pe)
```

```
Symbolic Ring
```

jetzt können die dortigen Methoden angewendet werden.

```
solve(pe==0,x)
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: x is not a valid variable.
```

```
parent(x)
```

```
Univariate Polynomial Ring in x over Rational Field
```

```
restore('x')
```

```
solve(pe==0,x)
```

```
[x == -sqrt(3), x == sqrt(3)]
```

Nachtrag zu solve

Die Lösungsmenge wird als Liste zurückgegeben.

```
sol=solve(x^2-3,x)
sol
```

```
[x == -sqrt(3), x == sqrt(3)]
```

Die Lösungen können daruas auf verschiedene Arten extrahiert werden.

```
sol[0]
```

```
x == -sqrt(3)
```

Durch Substitution

```
sin(x).subs(sol[0])
```

```
-sin(sqrt(3))
```

dabei werden alle Vorkommen der Variablen ersetzt.

```
(sin(x)*cos(x)).subs(sol[1])
```

```
cos(sqrt(3))*sin(sqrt(3))
```

Oder mit der Methode `rhs` (=right hand side):

```
sol[0].rhs()
```

```
    -sqrt(3)
```

```
sin(sol[0].rhs())
```

```
    -sin(sqrt(3))
```

```
sin(x).subs(x=sol[0].rhs())
```

```
    -sin(sqrt(3))
```