Sage Tutorium 2

Eine Besonderheit von Sage sind die Schnittstellen zu anderen Computeralgebra Systemen. Dabei werden sowohl freie System (wie Maxima, Axiom, Gap, Pari, Singular, Octave) als auch kommerzielle Systeme (wie Mathematica, Maple, Mupad, Magma, Matlab) unterstützt.

Eine solche Schnittstelle ermöglicht den Datenaustausch zwischen den verschiedenen Systemen. Und erlaubt es Funktionen anderer Computeralgebra-Systeme direkt von Sage heraus zu benützen.

Wir beschränken uns auf die Schnittstelle zu Maxima. Maxima ist standardmässig bei Sage installiert, und kann somit auch in jeder Sage Installation genützt werden.

Die Maxima-Schnittstelle

Es gibt mehrere Möglichkeiten wie Maxima von Sage aus benützt werden kann.

1. Eine Maxima Zelle

Wenn eine Zelle mit "%maxima" beginnt, wird der Code in der Zelle von Maxima berechnet.

```
%maxima
1+1
```

```
2
```

Achtung: Maxima hat eine von Sage verschiedene Syntax.

Zum Beispiel die wichtigen mathematischen Konstanten beginnen in Maxima alle mit einem %-Zeichen:

Konstante		Sage	Maxima
Kreiszahl:	π	pi	%pi
Eulersche Zahl:	e	e	%e
Komplexe Einheit:	i	I	%i

```
%maxima
%e^(%i * %pi)
```

```
-1
```

Das gleiche direkt in Sage:

```
e^(I * pi)
```

```
-1
```

Es gibt einige Funktionen die Maxima versteht, Sage aber noch nicht.

Zum Beispiel kennt Sage derzeit noch keine symbolische Faktorielle (das wird aber in einer der nächsten Versionen behoben.)

```
var('n')
factorial(n)/factorial(n + 2)
```

```
Traceback (click to the left for traceback)
...
```

```
TypeError: unable to convert x (=n) to an integer
```

Wir können aber Maxima verwenden um das gewünschte zu Berechnen:

```
%maxima
a: n!/(n+2)!
```

```
n!/(n+2)!
```

Variablen Zuweisungen werden in Maxima mit einem ":" vorgenommen. (In Sage mit "=").

Die Maxima Funktion **minfactorial** kann Ausdrücke die Faktorielle enthalten vereinfachen:

```
%maxima
minfactorial(a)
```

```
1/((n+1)*(n+2))
```

Achtung: Die Variable a ist nur in Maxima definiert. In Sage können wir nicht auf a zugreifen.

```
a
```

```
Traceback (click to the left for traceback)
...
NameError: name 'a' is not defined
```

2. Erzeugen von Variablen vom Typ Maxima

Die Funktion maxima() nimmt einen String als Argument, evaluiert diesen String mit Maxima, und gibt das Ergebnis als Variable zurück.

```
b = maxima('n!/(n+2)!')
b
```

```
n!/(n+2)!
```

```
type(b); parent(b)
```

```
<class 'sage.interfaces.maxima.MaximaElement'>
Maxima
```

Wir können uns b in mathematischer Notation ansehen.

```
show(b)
```

$$\frac{n!}{(n+2)!}$$

Und wir können Maxima Funktionen auf b anwenden.

```
b2 = b.minfactorial()
show(b2)
```

$$\frac{1}{(n+1)\,(n+2)}$$

Dokumentation zu Maxima Befehlen erhalten wir wie gewohnt.

```
# Tabulator Taste um alle Funktionen fuer Maxima Elemente zu sehen.
b.
```

```
b.minfactorial?
```

Elemente vom Typ Maxima können auch wieder normale Symbolische Ausdrücke konvertiert werden.

```
bsage = b2.sage()
show(bsage)
```

$$\frac{1}{(n+1)\,(n+2)}$$

```
type(bsage); parent(bsage)
```

```
<class 'sage.calculus.calculus.SymbolicArithmetic'>
Symbolic Ring
```

Variablen die in einer Maxima Zelle definiert wurden, können auf die gleiche weise in Sage-Ausdrücke konvertiert werden.

```
%maxima
c: sin(x)/sqrt(1-x^2)
```

```
sin(x)/sqrt(1-x^2)
```

```
csage = maxima('c').sage()
show(csage)
```

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

Alle symbolischen Ausdrücke in Sage können auch automatisch in Maxima Ausdrücke konvertiert werden. Damit lässt sich sehr komfortabel mit der Maxima Schnittstelle arbeiten.

```
z = 2 + 3*I
```

Wir können z mit der Maxima Funktion **polarform** in Polarkoordinaten umwandeln.

```
z_polar = maxima.polarform(z).sage()
show(z_polar)
```

$$\sqrt{13}e^{i\,\tan^{-1}\left(\frac{3}{2}\right)}$$

Mit dem Ergebnis kann ganz normal weitergerechnet werden.

```
show(z_polar^3)
```

$$13\sqrt{13}e^{3i\,\tan^{-1}\left(\frac{3}{2}\right)}$$

```
show((z_polar^3).imag())
```

$$13^{\frac{3}{2}}\sin\left(3\tan^{-1}\left(\frac{3}{2}\right)\right)$$

```
(z_polar^3).imag().trig_simplify()
```

```
9
```

Maxima Erweiterungspakete

Maxima besitzt viele Erweiterungspakete, die zusäzliche Funktionen verfügbar machen.

Erweiterungspakete werden mit dem Befehl **maxima.load()** geladen.

Beispiel: Vereinfachen von verschachtelten Wurzeln

Wir wollen den Ausdruck

$$\sqrt{11+6\sqrt{2}}+\sqrt{11-6\sqrt{2}}$$

vereinfachen.

```
w = sqrt(11+6*sqrt(2)) + sqrt(11 - 6*sqrt(2))
show(w)
```

$$\sqrt{6\sqrt{2}+11}+\sqrt{11-6\sqrt{2}}$$

Die normalen Simplifiktionsfunktionen von Sage liefern leider kein Ergebnis:

```
show(w.simplify_full())
show(w.factor())
show(w.expand())
```

$$\sqrt{6\sqrt{2}+11}+\sqrt{11-6\sqrt{2}}$$

$$\sqrt{6\sqrt{2}+11}+\sqrt{11-6\sqrt{2}}$$

$$\sqrt{6\sqrt{2}+11}+\sqrt{11-6\sqrt{2}}$$

Das Maxima-Paket **sqdnst** bietet aber eine Funktion die verschachtelte Wurzelausdrücke vereinfachen kann.

```
maxima.load("sqdnst")
```

```
"/local/data/huss/software/sage/local/share/maxima/5.16.2/share/sim
lification/sqdnst.mac"
```

```
maxima.sqrtdenest?
```

```
maxima.sqrtdenest(w).sage()
```

```
6
```

Wir können dieses Ergebnis numerisch überprüfen:

```
RR(w)
```

```
6.00000000000000
```

```

```