Sage-Glossar

Thomas Maxara, Stand Februar 2013

Version 1.2

Änderungsdatum: 02.04.2016

Inhaltsverzeichnis

A

Ableitungen
Mit dem diff-Befehl kann sage ganz leicht jede Funktion ableiten.

```
f1(a,x) = diff(f(a,x),x)
```

Von der Syntax her ist noch zu sagen, dass man einfach nur eine neue Funtkion (hier f1) als Ableitung definiert und nach dem Komma angibt, nach welcher Variable (hier x) abgeleitet wird.

Absolutwert
Mit dem abs(variable)-Befehl kann man den Absolutwert einer Variablen wiedergeben.

```
abs(-1.51)
1.51
abs(-10)
10
abs(2)
2
```

Annahmen machen
Mit dem Befehl assume() kann man Annahmen machen.

```
assume(n<0)
```

Um diese Annahme zu vergessen und eine andere zu machen, kann man den forget()-Befehl nutzen. Beispiel aus der sage Dokumentation:

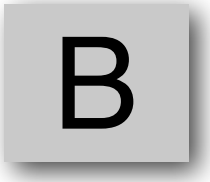
```
assume(x > 0)
bool(sqrt(x^2) == x)
True
forget()
bool(sqrt(x^2) == x)
False
```

Um zu sehen, welche Annahmen bisher gemacht wurden, kann man den Befehl assumptions() benutzen.

assumptions
Mit dem Befehl assumptions() kann man alle vorher gemachten Annahmen löschen. Beispiel aus der sage Dokumentation:

```
assume(x^2+y^2 > 0)
assumptions() [x^2 + y^2 > 0]
```

Dies ist nützlich, um z. B. vor dem Vergessen von Annahmen zu überprüfen, welche es gibt, oder z. B. nach dem Vergessen alle übrigen Annahmen ausgeben zu lassen.



Binomialverteilung
Die Binomialverteilung lässt sich in sage, wie prinzipiell jede Verteilung, die mittels einer Formel angegeben werden kann, in Sage wie folgt definieren:

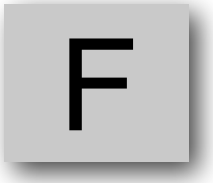
```
i,n,m,k = var('i,n,m,k')
bv(n,p,k) = binomial(n,k)*p^k*(1-p)^(n-k)
```

Kumulierte Binomialverteilung
Zur kumulierten Binomialverteilung gelangt man mittels der Befehle bv (es sei denn, man nennt die Verteilung anders) und sum.

```
sum(bv(20,0.8,i),i,0,14)
```

Dieser Befehl berechnet folgendes:

$$\sum_{k=0}^{14} \binom{20}{k} \cdot 0,8^k \cdot 0,2^{20-k}$$



Faktorisieren
sage kann Terme faktorisieren.
Erinnern Sie sich noch an die Polynomdivision? Bei der Polynomdivision ist das Ziel, ein Polynom in seine Linearfaktoren zu zerlegen. Dies gelingt zwar nicht immer vollständig, aber sage kann auch dies.

```
factor(3*x^4 - 6*x^3 - 15*x^2 + 18*x)
3*(x - 3)*(x - 1)*(x + 2)*x
```

forget
Mit dem Befehl forget() kann man vorher gemachte Annahmen "vergessen". Wenn man die Klammern leer lässt, werden alle Annahmen vergessen. Um etwas bestimmtes zu vergessen, muss man das zu vergessene in die Klammern schreiben. Bei mehreren zu vergessenden Annahmen trennt man diese durch ein Komma. Beispiel aus der sage Dokumentation:

```
x,y,z = var('x,y,z') assume(x>0, y>0, z == 1) forget(x>0, z==1)
assumptions() [y > 0]
```

Wenn man nicht genau weiß, welche Annahmen es gibt, kann man alle Annahmen mit dem Befehl assumptions() ausgeben.

Funktion zeichnen
Durch den Befehl

```
plot(f(x),-357,1567)
```

(Funktions)Gleichungen lösen
Der Befehl solve(Funktion==Zahl,Variabel) setzt Gleichungen einer bestimmten Zahl gleich und löst sie nach einer bestimmten Variablen (algebraisch) auf.

```
f(x) = 3+2*x
solve(f(x) == 2,x)
```

```
[x == (-1/2)]
```

Funktionsscharen zeichnen

```
var('a')
f(a,t) = t^2+a
schar = [ f(a,t)  for a in srange(1,4,0.25) ]
grafik = Graphics()
for element in schar: grafik += plot(element,[-3,4])
grafik.show()
```

```
schar = [ f(a,t)  for a in srange(1,4,0.25) ]
```

erzeugt eine Liste von Funktionstermen, wobei der Parameter a die Liste der Zahlen von 1 bis 4 mit Schrittweite 0,25 durchläuft.

```
grafik = Graphics()
```

erzeugt ein (noch leeres) Grafik-Objekt

```
for element in schar: grafik += plot(element,[-3,4])
```

fügt dem Grafik-Objekt die Schaubilder der einzelnen Funktionen hinzu

```
grafik.show()
```

zeigt das so erzeugte Grafik-Objekt an.

Abschnittsweise definierte Funktionen

Teil-Terme definieren

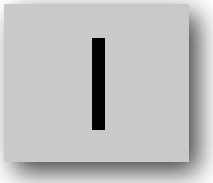
Mit Pieceweise diese Terme auf den Teilintervallen zu einer Funktion zusammensetzen.

```
f = Piecewise([[(a,b),f2],[(b,c),f1],[(c,d),f2]])
```

Pieceweis verlangt als Eingabeparameter eine Liste der Definitionen für die Abschnitte. Jeder Abschnitt wird durch eine Liste aus zwei Angaben festgelegt, die erste Angabe gibt die Intervallgrenzen als Wertepaar an, die zweite den Funktionsterm, der auf diesem Intervall gültig sein soll.

Der Plot-Befehl übernimmt automatisch den in Piecewise festgelegten Definitionsbereich .

```
f1(x) = -2*x^2+7
f2(x) = x^2+4
a = -3; b = -1; c = 1; d = 3
f3 = Piecewise([[(a,b),f2],[(b,c),f1],[(c,d),f2]])
show(f3)
plot(f3)
```



Integral berechnen
Um das Integral der Funktion f von a bis b zu berechnen:

```
integrate(f(x),x,a,b)
numerical_integral(f,a,b)
```

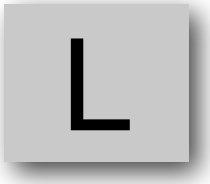
Interaktive Steuerelemente (Schieberegler)
Mit dem Befehl @interact gefolgt von def name(k=slider(1,10,1,2)) erzeugt man einen Schieberegler, den man sowohl beim Plotten von Funktionen als auch in jedem Rechenausdruck (Funktion) verwenden kann.

```
@interact
def name(k = slider(1,10,1,2)):
    grafik = plot(sin(x*k))
    grafik.show()
```

Der Eintrag @interact in der Eingabezelle sorgt dafür, dass die folgende Python-Funktion eine interaktive Ausgabe erzeugt. Der Name der Funktion ist ohne Bedeutung, ein Unterstrich täte es auch. In der Parameterliste dieser Funktion werden die Kontrollelemente aufgeführt.
Hier ein Schieberegler (slider), der wiederum durch Angabe von linker (hier: 1) und rechter Grenze (hier: 10), Startwert (hier: 1) und Schrittweite (hier: 2) festgelegt wurde.
Der mit dem Schieberegler erzeugte Wert wird der Variablen k zugewiesen und in der Funktion als Frequenz der Sinusfunktion verwendet.

```
@interact
def name(k = slider(1,19,1,15)):
    print 'Fehler 1. Art =', sum(bv(20,0.6,i),i,k,20)
```

An dieser Stelle variiert der Schieberegler die untere Grenze k einer Summe über (eine zuvor definierte!) Binomialverteilungsfunktion bv.

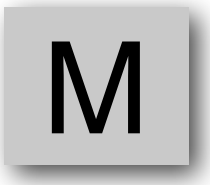


Liste von ganzen Zahlen
Der Befehl range(Zahl) liefert eine Liste der ganzen Zahlen von 0 bis Zahl-1.

```
a = range(7); a
[0, 1, 2, 3, 4, 5, 6]
```

Liste von Zahlen
Der Befehl srange(a,b,c) liefert eine Liste der Zahlen von a bis b-c in der Schrittweite c.

```
a = srange(3,9,0.25); a
[3.00000000000000, 3.25000000000000, 3.50000000000000,
3.75000000000000, 4.00000000000000, 4.25000000000000,
4.50000000000000, 4.75000000000000, 5.00000000000000,
5.25000000000000, 5.50000000000000, 5.75000000000000,
6.00000000000000, 6.25000000000000, 6.50000000000000,
6.75000000000000, 7.00000000000000, 7.25000000000000,
7.50000000000000, 7.75000000000000, 8.00000000000000,
8.25000000000000, 8.50000000000000, 8.75000000000000]
```



Matrizen
Matrizen lassen sich in Sage folgendermaßen definieren:

```
A = matrix(3,4,[1,3,2,4,-2,5,4,2,-3,-6,2,1])
```

matrix(3,4) erzeugt eine 3x3 Matrix, inklusive des Lösungsvektor; deshalb (3,4). Danach werden in eckigen Klammern alle Einträge, inklusive der Komponenten des Lösungsvektors der Reihe nach angegeben. Die ersten vier Einträge bedeuten also – sofern man die Matrix als zugehörige Matrix eines LGS interpretiert:

$$x_1 + 3x_2 + 2x_3 = 4$$

Der Befehl

```
A.rref()
```

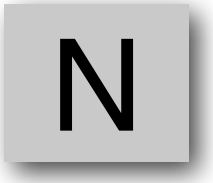
liefert dann die Normalen(einheits)form der Matrix und der Lösungsvektor des zugehörigen LGS lässt sich ablesen.

Abbildungsmatrizen
Eine Drehung um den Ursprung mit dem Winkel a wird mit folgender Matrix erreicht:

```
A = matrix(2,2,[cos(a),-sin(a),sin(a),cos(a)])
```

Eine Spiegelung an einer (allg.) Urpsrungsgeraden mit dem Richtungsvektor (u,w) durch:

```
A = 1/(u^2+w^2)*matrix(2,2,[u^2-w^2,2*u*w,2*u*w,w^2-u^2])
```



Nullstellen näherungsweise bestimmen
Mit dem Befehl find_root() kann man die Nullstellen eine Funktion näherungsweise bestimmten, wenn die Zahlen von solve() Befehl nicht handhabbar sind.

```
f(x) = x^2
n1 = find_root(f(x),-0.1,0.1);n1
0.0
```

Der Vorteil dieses Befehls ist, dass man die Nullstelle gleich benennen kann (siehe n1=). Nachteilig ist, dass man den ungefähren Bereich, in dem die Nullstelle liegt erst schätzen/ am Graph Bestimmen muss.

Numerische Zahlangaben (Dezimalzahl)vs. „exakte“ Zahlangaben (Bruchzahl)
Sage ist relativ „anspruchsvoll“, was die Eingabe von Zahlen angeht. Sobald eine Dezimalzahl verwendet wird, rechnet Sage numerisch. Die Angabe einer Zahl mit Hilfe des Bruchstriches bewirkt eine exakte Berechnung.

Dies hat Auswirkungen bei der Lösung von LGS, aber auch beim Berechnen von kumulierten Binomialverteilungen.

Numerische Approximation
Mit der Funktion n(Zahl) - oder der Methode Zahl.n() - können Zahlen z.B. ein Rechenergebnis approximiert, also gerundet, ausgeben werden.

```
n(pi)
3.14159265358979

2/142+5.n()
5.01408450704225
```

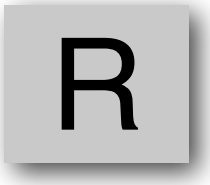
Die Funktion und die Methode nehmen auch die optionalen Argumente "digits=" und "prec=" an, welche die gewünschte Anzahl von Dezimalstellen bzw. die gewünschte Anzahl von Bit an Genauigkeit (Standardwert 53 Bit) sind.

```
n(2/3,digits = 3)
0.667

pi.n(prec = 100)
3.1415926535897932384626433833
```

Normalverteilung
Die standardisierte Normalverteilung muss in Sage als Funktion definiert werden. Dies ist durch folgenden Ausdruck möglich:

```
nv(x) = 1/(sqrt(2*pi))*e^(-1/2*x^2)
```



Regressionen
Regressionen lassen sich in Sage wie folgt durchführen:

```
x = [1,2.5,4,6,6.7,8];
```

```
y = [3,2.5,4.5,4,6,6.5]
R = zip(x,y)
```

Zunächst werden in Sage Variablen x und y definiert, in die Werte hineingeschrieben werden. Der Befehl zip fasst diese beiden Variablen zu einer einzigen (einem 2-dimensionalen array) zusammen.

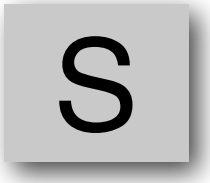
```
a,b = var('a,b')
model(x) = a*x+b
find_fit(R,model)
```

In der Funktion model wählt man das Regressionsmodell, in diesem Fall eine lineare Funktion. Der Befehl find_fit liefert dann Werte für die Variablen nach der „Methode der kleinsten Quadrate“.

```
[a == 0.53089887591466689, b == 1.9214419489329133]
```

Mit dem Befehl list_plot lassen sich dann die Punkte (evtl. zusammen mit der Regressionsfunktion) in ein Koordinatensystem zeichnen:

```
list_plot(R,pointsize=30)
```



show

Mit dem Befehl show() kann man mehrere Plots die in Variablen gespeichert sind in einem Koordinatensystem ausgeben lassen. Beispiel:

```
show(p+p1+p2)
```

oder

```
(p+p1+p2).show
```

Zusätzlich kann man die Funktion (show()) und die Methode (().show) durch Argumente erweitern, mit denen man die Skalierung, Darstellung usw. des Koordinatensystems bestimmen kann:

Breite und Höhe des Koordinatensystems:

```
figsize = [b,h]
```

Achsenbeschriftung:

```
axes_labels = ['x', 'y']
```

Achseneinteilung der x-Achse, z.B. :

```
ticks = x
```

Formatierung der Achseneinteilung, z.B. :

```
tick_formatter = x
```

Schriftgröße aller Beschriftungen:

```
fontsize = x
```

Auflösung in Punktedichte (Standard ist 100):

```
dpi = x
```

Koordinatensystem einrahmen:

```
frame = true
```

Achsen ausblenden:

```
axes = false
```

Positionierung der Argumente bei der show-Funktion:

```
show(p+p1,Argument1,Argument2,[...])
```

Positionierung der Argumente bei der show-Methode:

```
(p+p1).show(Argument1,Argument2,[...])
```

Summe (von Zahlen)

In sage lassen sich Summen mit dem Befehl sum(Ausdruck,Argument,Startwert,Endwert) leicht berechnen.

```
sum(i,i,0,9)
45
```

Zuvor muss jedoch die Variable i (als Variable) definiert worden sein.

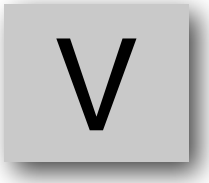
```
sum(i^2,i,0,10)
385
```

Und sage kann auch mit Variablen summieren:

```
sum(i^2,i,0,n-1)
```

...

Dann fehlt noch der Befehl factor und die Formel ist faktorisiert.



Variable

Der Befehl

```
x,y,z = var ('x,y,z')
```

Bestimmt x,y und z als Variable.
Diese kann in einer Funktion verwendet werden oder ihr kann ein Wert zugewiesen werden.

Variable näherungsweise bestimmen

Wenn beim Lösen einer Gleichung nach einer Variable keine brauchbaren Ergebnisse erscheinen, kann man den Befehl .roots(RR) verwenden.

```
g1 = integrate(f(x)) == 27/4/2
```

Mann bestimmt zu erst die Gleichung. Dann nennt man den Namen der Gleichung mit .roots(ring=RR) dahinter.

```
g1.roots(ring = RR)
[(1.84281729560283,1),(3.74244686576301,1)]
```

Siehe da, Sage spuckt recht genaue Ergebnisse aus, die man auch verwenden kann.

(Werte)tabellen

Wertetabellen können in sage mit Hilfe des folgenden Befehls ausgegeben werden:

```
var('i')
f(x) = x^2
for i in srange(1,4,0.5): (i,f(i))
(1.00000000000000, 1)
(1.50000000000000, 2.25000000000000)
(2.00000000000000, 4.00000000000000)
(2.50000000000000, 6.25000000000000)
```

```
(3.00000000000000, 9.00000000000000)
(3.50000000000000, 12.2500000000000)
```

Winkel

Winkel werden in Sage im Bogenmaß aus- und angegeben. Möchte man dies in einem worksheet ändern, so muss man die Funktionen sin und cos (und evtl. tan) umdefinieren. Dies funktioniert wiefolgt:

```
def sin(w):
    return round(math.sin(w*pi/180),5)
def cos(w):
    return round(math.cos(w*pi/180),5)
```

Für die Umkehrfunktionen von Sinus und Kosinus definiert man sich am besten zwei eigene - neue - Funktionen

```
arcsin_g(w) = arcsin(w)/pi*180
arccos_g(w) = arccos(w)/pi*180
```

Wurzelziehen

Mit dem Befehl sqrt(Zahl) kann man die Quadratwurzel einer Zahl ziehen.

```
sqrt(16)
4
```

Durch das Argument "prec=" kann man zusätzlich die die gewünschte Anzahl von Bit an Genauigkeit angeben.

```
sqrt(125,prec = 60)
11.180339887498948
```

Sage Kurzreferenz

(beruht auf Sage Quick Reference von William Stein und anderen) GNU Free Document License

Arbeiten im Notebook

Eingaben auswerten	Klick auf **evaluate** oder **<shift-enter>**
Doku zum Befehl *befehl*	*befehl***? <tab>**
Automatische Befehlsergänzung z.B.: Befehle, die mit *fa* beginnen	*fa* **<tab>**
Methoden eines Objekts *obj* anzeigen	*obj***. <tab>**
Neue Eingabe-Zelle einfügen	Klick auf die blaue Linie
Eingabe-Zelle löschen	Inhalt löschen, dann **<backspace>**
typografisch schöne Ausgabe	show(Ausdruck)

Wichtige Funktionen, Konstanten und Operatoren

einige Konstanten	$\pi = pi$	e = e	$\infty = oo$
numerische Angabe z.B. π (mit k Ziffern)		pi.n(digits=k)	
einige Funktionen	sin cos tan log ln exp (u.a.)		

ab	a*b	$\frac{a}{b}$	a/b	a^b	a^b	$\log_b(x)$	log(x,b)
$\sqrt{x}$	sqrt(x)	$\sqrt[n]{x}$	x^(1/n)	$\lvert x \rvert$	abs(x)	n!	factorial(n)

Summen $\sum_{i=1}^{5} f(i)$	sum(f(i) for i in [1..5])	$\binom{n}{k}$	binomial(n,k)

Variablen und Funktionen (symbolische*)

Variablen definieren (z. B.: a , t, y)	a, t, y = var(' a t y')
Funktionen definieren, z.B.: $f(x) = x^2$	f(x) = x^2

*) Die Z*uweisung* **a= 5** erzeugt ein Zahl-Objekt, mit dem gerechnet werden kann (a+a ergibt hier 10);
a = var('a') erzeugt eine symbolische Variable, die keinen festen Wert hat (a + a ergibt hier 2a)
Die Variable **x** ist als symbolische Variable vordefiniert.

Bezeichner für Grundmengen

Ganze Zahlen	ZZ	Reelle Zahlen (Decimals)	RR
Rationale Zahlen	QQ	Komplexe Zahlen	CC

Operationen mit Termen

In Faktoren zerlegen	factor(term)	term.factor()
Ausmultiplizieren	expand(term)	term.expand()
Vereinfachen (Es gibt weitere Varianten von *simplify.)*	term.simplify() term.simplify_full()	

Gleichungen lösen

Gleichung (**doppeltes** Gleichheitszeichen !)	f(x) == g(x)
Gleichung lösen	solve (f(x) == g(x), x)
Gleichungssystem lösen (Liste von Gleichungen)	gl = [f(x,y) == 0, g(x,y)==0] solve(gl, x, y)
Nullstelle (numerisch) im Intervall $[a\,;b]$	find_root(f(x), a, b)

Funktionen plotten

Funktion f im Intervall $[a\,;b]$	plot(f, xmin=a, xmax=b)
Ein Grafikobjekt erzeugen und anzeigen. (Zusätzliche Optionen bei *show)*	P = plot(f, xmin=a, xmax=b) P.show(ymin=c, ymax=d)

Analysis

Grenzwert $\lim_{x\to 0} f(x)$	$\lim_{x\to\infty} f(x)$	limit(f(x),x=0)	limit(f(x),x=oo)
Ableitung	n-te Ableitung	diff(f(x),x)	diff(f,x,n)
Integral $\int f(x)\,dx$	$\int_a^b f(x)\,dx$	integral(f(x),x)	integral(f(x),x,a,b)

Sonstiges

Listen (z.B der Zahlen 0,1, 2, 3, 4)		[0,1,2,3,4] oder [0..4] oder range(5)	
Wertepaare, Tupel	n-Tupel	(1,4)	(1, -2, 4 , 5)
Mengen		set([1,2,3,4])	
Listen schnell erzeugen, z.B. Wertepaare		L = [(x,f(x)) for x in [-2..2]]	
Eine Liste L durchlaufen (und z.B ausgeben)		`for k in L:` `  print k`	

Grafik (zweidimensional)

Streckenzug (durch die Punkte der Liste)	line([(x_1,y_1),...,(x_n,y_n)], *options*)
Polygonzug (zum Anfangspunkt geschlossen)	polygon([(x_1,y_1),...,(x_n,y_n)], *options*)
Kreis um Mittelpunkt (x,y) mit Radius r	circle((x,y), r, *options*.)
Textausgabe an der Position (x,y)	text('mein text', (x,y), *options*)
Punkt (a,b)	point((a,b), *options*)
Eine Liste von Punkten erzeugen und als Grafikobjekt ausgeben	L = [(k, 2*k) for k in [1..4]) points(L, *options*)
Eine Funktion im Intervall [a,b] plotten	plot(f(x),a,b, *options*)
Eine parametrische Kurve plotten	parametric_plot((t, t^2), (t, -4, 4))
Grafikobjekte anzeigen	show(P, *options*) P.show(*options*)
Mehrere Grafik-Objekte kombinieren	C= circle((1,1), 2) P = point((1,1)) show(C+P)

Grafik-Optionen

Größe der Grafik: *figsize*	figsize = [w,h] figsize = [4,2]
Seitenverhältnis der Grafik: Option *aspect_ratio*	aspect_ratio = 1
Farboption *rgbcolor* Rot-Grün-Blau-Mischung oder Farbnamen angeben auch Farboption *color*	rgbcolor=(r,g,b) Werte für r,g,b in einer Skala von 0 bis 1 rgbcolor = 'green' color = 'red'
Weitere Optionen (je nach Grafikobjekt)	alpha, thickness, linestyle, fill, fillcolor, pointsize, opacity, ...
Optionen der Grafik-Objekte nachschlagen	z.B.: plot? **<tab>**
Werden Grafikobjekte mit *show* dargestellt, gibt es weitere Optionen, zum Beispiel	ymin=-4, ymax=4 axes = False frame = True

Vektoren und Matrizen

Vektor $\vec{v}=\begin{pmatrix}2\\3\end{pmatrix}$	v = vector([2,3]) *Hinweis: view(v) zeigt den Zeilenvektor*	
Matrix $M=\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	M = matrix([[1,2],[3,4]]) M = matrix(2,2, [1,2,3,4]) M = matrix(QQ,2,2, [1,2,3,4])	
Inverse Matrix	M^-1	M.inverse()
Transponierte Matrix	M.transpose()	
Lösen $M\,\vec{x}=\vec{v}$ (LGS mit Koeffizientenmatrix M und 'rechter Seite' v.)	x = M.solve_right(v)	
Skalarprodukt $s=\vec{a}\,\vec{b}$	s = a*b	
Vektorprodukt $\vec{c}=\vec{a}\,x\,\vec{b}$	c = a.cross_product(b)	

Grafik dreidimensional

Streckenzug (durch die Punkte der Liste)	line3d([(x_1,y_1,z_1),...,(x_n,y_n,z_n)], *options*)
Würfel mit dem Mittelpunkt (x,y,z) und der Kantenlänge a	cube((x,y,z),a, *options*)
Kugel um den Mittelpunkt (x,y,z) mit Radius r	sphere((x,y,z), r, *options*)
Textausgabe an der Position (x,y)	text3d('mein text', (x,y,z), .*options*)
Punkt (x1,x2,x3)	point3d((x1,x2,x3), *options*)
Eine Funktion f(x,y) plotten	plot3d(f(x,y), (x,-2,2), (y,-2,2))
Eine parametrische Kurve für $t\in[a\,;b]$ plotten	parametric_plot3d((f(t),g(t), h(t), (t,a,b)
Eine parametrische (auch gekrümmte) Fläche plotten	parametric_plot(vector([r*r,s,r]),(r,-4,4),(s,0,5))
Pfeile	arrow3d((0,0,0), (1,1,1), color='green')

Animierte Grafiken erzeugen

Zuerst eine Liste (L) von Grafikobjekten erzeugen. In eine animierte Grafik umwandeln und anzeigen mit Verzögerung	L = [plot(x^k) for k in [0..8]] F = animate(L, options) F.show(delay=50)

Norbert Domes: Sage Kurzreferenz, Stand März 2011