HTBLuVA St. Pölten
Elektrotechnik

3100 St. Pölten | Waldstrasse 3 | 02742-75051-300
et@htlstp.ac.at | http://et.htlstp.ac.at
Höhere Abteilung | Fachschule | Abendschule

Erste Schritte mit MAXIMA

Wilhelm Haager
HTL St. Pölten, Abteilung Elektrotechnik
wilhelm.haager@htlstp.ac.at

3. November 2013

Inhaltsverzeichnis

1 Grundlegendes **1**

2 Benutzeroberfläche wxMaxima **3**
2.1 Aufbau 4
2.2 Abspeichern und Laden 5
2.3 Tastenkürzel 5
2.4 Export 6

3 Erste Schritte **7**
3.1 Maxima als Taschenrechner 7
3.2 Maxima als symbolischer Rechner 9
3.3 Logische Ausdrücke 11
3.4 Listen 11
3.5 Komplexe Zahlen 13
3.6 Summen und Grenzwerte 14
3.7 Differenzial- und Integralrechnung 15
3.8 Einschränken des Bereiches von Variablen 16
3.9 Gleichungen 17
3.10 Benutzerdefinierte Funktionen 19
3.11 Vektoren und Matrizen 19
3.12 Zeichen und Zeichenketten 21
3.13 Grafiken 22
3.14 Programmkontrollstrukturen 24

1 Grundlegendes

Maxima ist ein Open-Source-Abkömmling von *Macsyma*, einem der ersten Computeralgebra-Systeme. Macsyma wurde in den Jahren 1968–1982 am M.I.T. in Boston im Auftrag des US-Energieministeriums (*Department of Energy, DoE*) entwickelt, mitfinanziert von der DARPA (*Defence Advanced Research Projects Agency*).

1998 wurde eine Version *Macsyma* unter der *GNU General Public Licence* (GPL) unter dem Namen *Maxima* veröffentlicht und somit jedermann kostenlos verfügbar gemacht. Es wird seither von einer unabhängigen Gruppe von Anwendern und Entwicklern gepflegt und – insbesondere im Bereich der Benutzerschnittstellen und Grafikmöglichkeiten – intensiv weiterentwickelt.

Maxima weist eine Reihe von Eigenschaften auf, die es gegenüber anderen Rechenprogrammen auszeichnet:

Maxima ist plattformunabhängig: Es ist für alle gängigen Betriebssysteme verfügbar, neuerdings auch für Android.

Maxima ist ein echtes Computeralgebrasystem: Variablen können nicht nur mit numerischen Werten, sondern auch mit symbolischen Ausdrücken belegt werden.

Berechnungsabläufe können direkt auf der Ebene des interaktiven Arbeitens programmiert werden, nicht nur innerhalb von Funktionsdefinitionen.

Maxima ist frei verfügbar und somit kostenlos; ein wesentliches Argument im Ausbildungsbereich: für Schulen fallen keine Lizenzgebühren an, die Schüler bekommen ein Werkzeug, das sie auch nach der Ausbildung zur Verfügung haben werden – eine Voraussetzung für echte Nachhaltigkeit.

Maxima ist einfach bedienbar: Mathematische Ausdrücke werden in reinem ASCII-Code in der für Programmiersprachen üblichen Notation mit der Tastatur eingegeben; fingerverrenkende Tastenkombinationen und Mausklick-Akrobatik sind nicht erforderlich.

Maxima trennt klar zwischen Rechnung und Dokumentation: Berechnungen können mit Text und Grafiken dokumentiert werden. Für die Nachvollziehbarkeit von Rechengängen ist es dabei vorteilhaft, klar zu erkennen, welche mathematischen Ausdrücke Teil der Berechnung und welche Ausdrücke Teil der erläuternden Dokumentation sind.

Für eine typografisch korrekte Dokumentation können Ausdrücke im TEX-Format ausgegeben werden.

Maxima-Files sind versionskompatibel: Alle Datenfiles (Programmpakete und abgespeicherte Maxima-Sitzungen) sind reine Textdateien oder zip-komprimierte Dateien davon. Dabei ist sowohl Abwärts- als auch Aufwärtskompatibilität zwischen unterschiedlichen Versionen von Maxima gewährleistet.

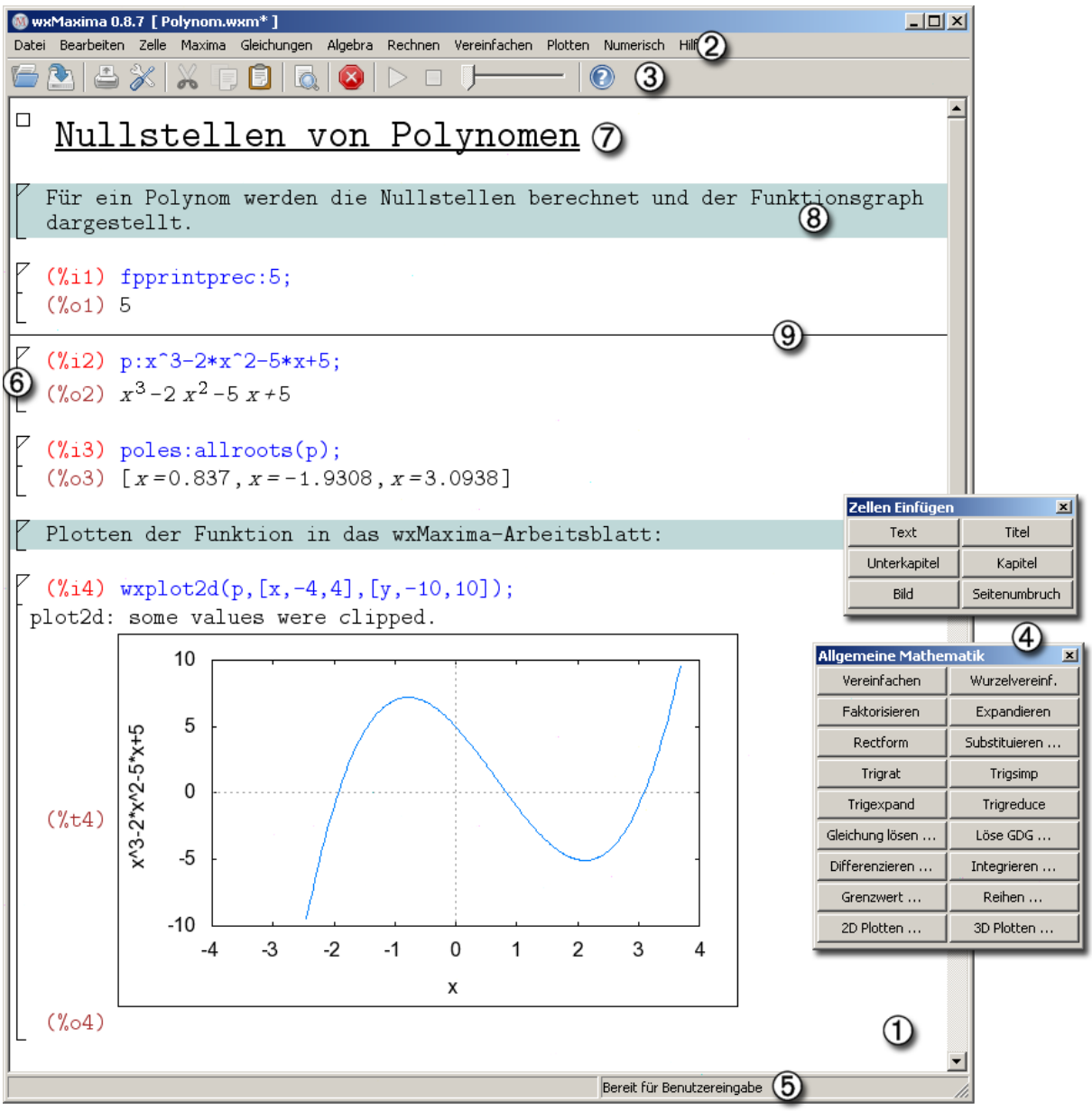
Maxima ist eine umfangreiche Programmiersprache und somit beliebig erweiterbar. Es unterstützt prozedurales Programmieren, funktionales Programmieren und (in beschränkem Ausmaß) auch regelbasiertes Programmieren.

Maxima hat eine sehr engagierte Entwickler- und Anwendergemeinde: Anfragen im Userforum werden in der Regel innerhalb weniger Stunden beantwortet, gemeldete Bugs (die gibt es bei *jedem* Programm) innerhalb weniger Tage behoben.

Links

- `http://maxima.sourceforge.net`
  Homepage mit vielen Informationen, Dokumentationen, Zusatz-Paketen und Download-Links
- `http://wxmaxima.sourceforge.net`
  Homepage des Benutzer-Interfaces *wxMaxima* mit User-Forum
- `http://www.austromath.at/daten/maxima/`
  Maxima Online-Kurs von Walter Wegscheider
- `http://www.computermathematik.info/`
  Homepage von Johann Weilharter mit vielen Maxima-Materialien
- `http://www.csulb.edu/ woollett/`
  Homepage von Ted Woollett mit ausführlichen Tutorials
- `http://crategus.users.sourceforge.net/maxima.html`
  Deutschsprachiges Maxima-Handbuch von Dieter Kaiser („Crategus“):
- `http://maxima.cesga.es`, `http://maxima-online.org`
  Maxima Online-Webinterfaces
- `http://www.math.utexas.edu/pipermail/maxima/`
  Maxima Mailingliste

2 Benutzeroberfläche wxMaxima



① . . . Arbeitsfenster
② . . . Menüleiste
③ . . . Werkzeugleiste
④ . . . Befehlsschaltflächen
⑤ . . . Statusfeld
⑥ . . . Zellenklammer
⑦ . . . Titelzelle
⑧ . . . Textzelle
⑨ . . . Einfügecursor

Abbildung 2.1: wxMaxima-Benutzeroberfläche

wxMaxima ist eine von mehreren grafischen Benutzeroberflächen für Maxima. Es ermöglicht die grafische Ausgabe von Formeln, die direkte Ausgabe von Grafiken in das wxMaxima-Arbeitsfenster, das Abspeichern von Arbeitssitzungen, sowie (für notorische Mausklickser) den Aufruf der wichtigsten Befehle über Menüs und Befehlsschaltflächen.

Bei der Installation von Maxima wird wxMaxima automatisch als Standard-Interface mitinstalliert.

2.1 Aufbau

Arbeitsfenster: Hier erfolgt die Eingabe, sowohl von mathematischen Ausdrücken, als auch von gewöhnlichem Text zur Gliederung und Dokumentation von Berechnungen. Mit der Enter-Taste <*enter*> erfolgt ein Zeilenumbruch, das Abschließen der Eingabe erfolgt mit der Tastenkombination <*shift*><*enter*>. Damit wird auch ein eingegebener mathematischer Ausdruck zur Auswertung an Maxima übergeben.

Die Ausgabe des Ergebnisses erfolgt ebenfalls auf dem Arbeitsblatt, unmittelbar unter der Eingabe. Werden den Namen der Grafikroutinen die Buchstaben „`wx`“ vorangestellt, so erfolgt die Ausgabe der Grafiken ebenfalls auf dem Arbeitsblatt. Eine so erstellte Grafik kann in die Zwischenablage kopiert oder als png-File abgespeichert werden; Anklicken und nachfolgender Rechtsklick öffnet ein entsprechendes Kontextmenü.

Menüleiste: Enthält neben den üblichen Menüpunkten zum Laden, Speichern und Konfigurieren wichtige Maxima-Befehle zum Aufruf durch Anklicken.

Werkzeugleiste: Zusätzlich zu einigen als eigene Schaltflächen herausgeführten Menübefehlen finden sich hier Bedienelemente zum Abspielen animierter Grafiken.

Befehlsschaltflächen: Damit können ebenfalls wichtige Befehle durch Anklicken mit der Maus aufgerufen werden; sie können weggeschaltet werden.

Statusfeld: Informiert den Benutzer über den aktuellen Status: entweder die Bereitschaft, Benutzereingaben entgegenzunehmen, eine Berechnung durchzuführen, oder eine Ausgabe zu formatieren (um sie entsprechend darzustellen).

Zellenklammer: Eingaben und zugehörige Ausgaben sind zu *Zellen* zusammengefasst, die das Arbeitsblatt strukturieren und durch eine Zellenklammer am linken Rand des Arbeitsblattes gekennzeichnet sind. Durch Anklicken der Zellenklammer kann eine Zelle zwecks Löschens, Verschiebens oder Kopierens markiert werden; Rechtsklick öffnet ein Kontextmenü. Anklicken der dreieckigen Marke am oberen Rand der Zellenklammer blendet die Ausgabe weg.

Titelzelle, Abschnittszelle: Damit kann ein Arbeitsblatt mit Überschriften unterschiedlicher Größen optisch strukturiert werden. Erzeugt werden sie durch Menübefehle [*Bearbeiten*][*Cell*] oder die Tastenkürzel <*ctrl*><2> bzw. <*ctrl*><3>.

Textzelle: Dient zur Kommentierung einer Berechnung. Erzeugt wird eine Textzelle über einen Menübefehl oder mit dem Tastenkürzel <*ctrl*><1>.

Einfügecursor: Kennzeichnet den Zwischenraum zweier Zellen, in dem eine neue Eingabe erfolgt (Text oder zu berechnender Ausdruck), durch eine waagrechte Linie. Prinzipiell können Ausdrücke an beliebigen Stellen im Arbeitsblatt eingefügt werden, die Auswertung erfolgt jeweils sofort nach Abschließen der Eingabe (mit <*shift*><*enter*>). Die Berechnungsreihenfolge entspricht dabei der Reihenfolge der Eingabe – unabhängig von der Anordnung auf dem Arbeitsblatt.

Mit dem Tastenkürzel <*ctrl*>R kann ein ganzes Arbeitsblatt in einem Zug (von oben nach unten) neu durchgerechnet werden. Die Berechnungsreihenfolge entspricht hier logischerweise aber der *Anordnung auf dem Arbeitsblatt*, die nicht mit der Reihenfolge der ursprünglichen Eingabe übereinstimmen muss.

2.2 Abspeichern und Laden

Über das Menü [*Datei*][*Sichern als*] kann eine wxMaxima-Arbeitssitzung auf drei Arten abgepeichert werden:

wxm-File: Es enthält alle eingegebenen Ausdrücke und Texte mit zusätzlichen Kommentaren. Diese Kommentare werden zwar von *Maxima* ignoriert, legen aber die Struktur des Arbeitsblattes in *wxMaxima* fest.

Ein wxm-File ist ein normales ASCII-File, kann daher mit einem Texteditor geöffnet und verändert werden. Die Struktur der Kommentare und Leerzeichen muss dabei aber erhalten bleiben, damit das File bei neuerlichem Einlesen in wxMaxima fehlerfrei erkannt wird.

wxmx-File: Es enthält alle Ausdrücke und Texte einer wxMaxima-Arbeitssitzung, sowie Formatierungsanweisungen zum Wiedereinlesen in einem binären XML-Format. Zuätzlich kann es auch in das Arbeitsblatt eingefügte Bilder zur Dokumentation von Berechnungen enthalten.

mac-File: Es enthält die Maxima-Eingaben im ASCII-Format (also editierbar) ohne irgendwelche Formatierungsanweisungen. Texte sind zwar als Kommentare abgespeichert, werden aber beim Wiedereinlesen ignoriert.

wxm-Files und wxmx-Files werden über das Menü [*Datei*][*Öffnen*] in wxMaxima eingelesen. Dabei wird eine neue Arbeitssitzung begonnen. Vorherige Eingaben gehen – wenn sie nicht zuvor abgespeichert wurden – verloren; die Nummerierung der eingegebenen Ausdrücke beginnt wieder bei 1. Beim Einlesen werden zunächst nur die *Eingaben* auf dem Arbeitsblatt dargestellt, die Berechnung wird erst mit der Eingabe von <*ctrl*>R gestartet. Die Reihenfolge der Berechnungen erfolgt dabei nach der Position auf dem Arbeitsblatt, von oben nach unten und *nicht* in der Reihenfolge der ursprünglichen Eingaben.

mac-Files werden über das Menü [*Datei*][*Batch-Datei laden*] oder mit dem Maxima-Befehl `batch` eingelesen. Dabei werden alle im File enthaltenen Anweisungen als *Programm* ausgeführt.

2.3 Tastenkürzel

Für einen geübten Benutzer sind Tastatureingaben zur Programmbedienung meist bequemer und jedenfalls effizienter als die Auswahl von Menüpunkten und das Drücken von Schaltflächen mit der Maus (Tastatureingabe kann blind erfolgen – Mausbedienung bedarf immer der visuellen Rückkopplung vom Bildschirm zum Auge). Viele Befehle können daher – alternativ zur Auswahl aus Menüs – als Tastenkürzel eingegeben werden.

<*ctrl*>L ruft den Maxima-Befehl `load` auf, ein mac-File wird eingelesen und ausgeführt. Dabei werden *keine* Ausgaben für die einzelnen im File enthaltenen Ausdrücke (oder Befehle) erzeugt. Diese Art der Ausführung ist in erster Linie für *Pakete* sinnvoll, die vorwiegend Funktionsdefinitionen enthalten.

Windows-spezifische Tastenkürzel:

<*ctrl*>O	Öffnen eines wxm-Files (und Beginn einer neuen Arbeitssitzung)
<*ctrl*>P	Drucken des Arbeitsblattes
<*ctrl*>Q	Beenden von wxMaxima
<*ctrl*>C	Kopieren des markierten Bereiches in die Zwischenablage
<*ctrl*>X	Ausschneiden des markierten Bereiches in die Zwischenablage
<*ctrl*>V	Einfügen des Inhalts der Zwischenablage

xwMaxima-spezifische Tastenkürzel:

<*ctrl*>R	Neuberechnung des gesamten Arbeitsblattes
<*ctrl*>L	Laden eines Pakets (mac-File)
<*ctrl*>B	Ausführen eines Maxima-Programms (mac-File)
<*ctrl*>G	Unterbrechen einer Berechnung
<*ctrl*>1	Erstellen einer neuen Textzelle
<*ctrl*>2	Erstellen einer Titelzelle
<*ctrl*>3	Erstellen einer Abschnittszelle
<*ctrl*>4	Erstellen einer Unterabschnittszelle
<*alt*>I	Vergrößern der Darstellung
<*alt*>O	Verkleinern der Darstellung

wxMaxima Tastenkürzel

<*ctrl*>B ruft den Maxima-Befehl `batch` auf, der ebenfalls ein mac-File einliest und ausführt. Dabei werden Ausgaben für alle im File enthaltenen Ausdrücke (oder Befehle) erzeugt, genau so, als wären die Ausdrücke einzeln mit der Tastatur eingegeben. Diese Art der Ausführung ist in erster Linie für *Programme* sinnvoll, die (im Gegensatz von Funktionsdefinitionen) unmittelbar auszuführenden Code enthalten.

2.4 Export

Eine wxMaxima-Arbeitssitzung kann in ein HTML-File und in ein LaTeX-File exportiert werden.

Beim Export in HTML werden die Ausgaben von Maxima – sowohl die Texte und Formeln als auch die Grafiken – als png-Files in einem Verzeichnis mit dem Namen *name*`_img` abgespeichert.

Der Export in LaTeX funktioniert (noch) nicht ganz fehlerfrei.

3 Erste Schritte

Maxima arbeitet wie ein Interpreter, ein eingegebener Ausdruck oder eine Anweisung wird sofort evaluiert bzw. ausgeführt. In wxMaxima bewirkt die <*enter*>-Taste einen *Zeilenumbruch* bei der Eingabe; die *Auswertung* erfolgt durch Eingabe von <*shift*><*enter*>.

In Maxima gibt es keinen formalen Unterschied zwischen Daten und Programmcode, es gibt daher auch keinen Unterschied zwischen einem *Ausdruck* und einer *Anweisung*. Daher sind auch die Begriffe aus*werten* (evaluieren) und aus*führen* gleichwertig. Wie und in welchem Ausmaß diese Auswertung erfolgt, kann in vielerlei Hinsicht durch entsprechende Anweisungen und das Setzen globaler Variablen gesteuert werden.

Jede Anweisung wird mit einen Strichpunkt oder dem Dollarzeichen abgeschlossen. Bei Angabe des Strichpunktes wird das Ergebnis der Auswertung ausgegeben, bei Angabe des Dollarzeichens erfolgt keine Ausgabe (Speicherung aber schon).

Bei Verwendung der Benutzeroberfläche wxMaxima müssen die Anweisungen *nicht* mit einem Strichpunkt abgeschlossen werden; er wird am Ende jeder Eingabe automatisch hinzugefügt.

Alle Eingaben und Ergebnisse werden von Maxima mit Marken versehen, die mit fortlaufenden Nummern *xx* gekennzeichnet sind: `%i`*xx* („input“) bzw. `%o`*xx* („output“). Mit diesen Nummern kann man sich nachträglich auf die entsprechenden Ausdrücke beziehen. Manche Ausgaben (z. B. Grafiken) werden mit Zwischenmarken der Form `%t`*xx* versehen.

3.1 Maxima als Taschenrechner

Mathematische Ausdrücke können in gewohnter Weise mit den gebräuchlichen Operatoren für die Grundrechnungsarten (`+`, `-`, `*`, `/`) und das Potenzieren (`^` oder `**`), sowie mit der gewohnten Notation für mathematische Standardfunktionen (`sin`, `cos`, `sqrt`, `log`,...) formuliert werden.

Mit *runden* Klammern kann die Berechnungsreihenfolge beeinflusst werden; die Argumente von Funktionen werden – gegebenenfalls durch Beistriche getrennt – ebenfalls in *runde* Klammern eingeschlossen. *Eckige* Klammern fassen mehrere Ausdrücke – durch Beistriche getrennt – zu einer *Liste* zusammen.

Maxima unterscheidet zwischen Ganzzahlen und Gleitkommazahlen. Rationale Zahlen werden als Brüche dargestellt, Ausdrücke, die irrationale Zahlen ergeben, bleiben unausgewertet, das heißt in symbolischer Form bestehen. Die Umwandlung in Gleitkommazahlen kann mit der Umwandlungsfunktion `float` oder dem Auswertebefehl `ev` mit dem Zusatz `numer` erfolgen.

Eine einfache Rechnung

```
(%i1) 12*5-30/4+2^3;
(%o1)                              121
                                   ---
                                    2
```

Die Reihenfolge der Operatoren kann mit *runden* Klammern beeinflusst werden:

```
(%i2) (3+5)/(2*(5-2)-2);
(%o2)                               2
```

a+*b*, *a*-*b*, *a***b*, *a*/*b*	Addition, Subtraktion, Multiplikation, Division
a^*b* oder *a****b*	Potenz a^b
`sqrt(`*x*`)`, `exp(`*x*`)`, `log(`*x*`)`	Wurzel, Exponentialfunktion, natürlicher Logarithmus
`sin(`*x*`)`, `cos(`*x*`)`, `tan(`*x*`)`	Winkelfunktionen
`asin(`*x*`)`, `acos(`*x*`)`, `atan(`*x*`)`	Arkusfunktionen
`float(`*x*`)`	Umwandlung aller Zahlen im Ausdruck *x* in Gleitkommazahlen
`floor(`*x*`)`, `round(`*x*`)`	Abschneiden und Runden
`%pi`, `%e`, `%i`	Kreiszahl π, Eulersche Zahl und imaginäre Einheit
Systemvariablen:	
`fpprintprec`	Anzahl der signifikanten Stellen bei der Ausgabe von Gleitkommazahlen
`float`	Umwandlung aller Zahlen in Gleitkommazahlen (Default: `false`)
`numer`	Wie `float`, veranlasst zusätzlich mathematische Funktionen zur Auswertung in Gleitkommadarstellung (Default: `false`)

Maxima als Taschenrechner

Eckige Klammern fassen mehrere Audrücke zu einer *Liste* zusammen:

```
(%i3) [2+5,2*5,2/5,2^5];
(%o3) [7,10,2/5,32]
```

Maxima unterscheidet zwischen Ganzzahlen und Gleitkommazahlen.

```
(%i4) [4/6, 4/6.0];
(%o4) [2/3,0.66666666666667]
```

Mit der Systemvariablen `fpprintprec` wird die Anzahl der angezeigten Stellen von Gleitkommazahlen festgelegt.

```
(%i5) fpprintprec:5;
(%o5) 5
```

Ausgabe mit begrenzter Stellenzahl

```
(%i6) [4/6, 4/6.0];
(%o6) [2/3,0.667]
```

Darstellung von rationalen und irrationalen Zahlen

```
(%i7) [%pi,sqrt(2),8/3];
(%o7) [π,√2,8/3]
```

Umwandlung in die Gleitkomma-Darstellung mit der Funktion `float`

```
(%i8) float([%pi,sqrt(2),8/3]);
(%o8) [3.1416,1.4142,2.6667]
```

Umwandlung in die Gleitkomma-Darstellung mit dem Auswertebefehl `ev`

```
(%i9) ev([%pi,sqrt(2),8/3],numer);
(%o9) [3.1416,1.4142,2.6667]
```

Vereinfachte Anwendung des Befehls `ev`

```
(%i10) [%pi,sqrt(2),8/3],numer;
(%o10) [3.1416,1.4142,2.6667]
```

Gibt es kein symbolisches Ergebnis, so bleiben mathematische Funktionen unausgewertet.

```
(%i11) [sin(%pi/4),sin(%pi/5),float(sin(%pi/5))];
(%o11) [1/sqrt(2), sin(%pi/5), 0.588]
```

Abschneiden und Runden von Gleitkommazahlen

```
(%i12) [floor(1234.56),round(1234.56)];
(%o12) [1234, 1235]
```

3.2 Maxima als symbolischer Rechner

$a : b$	Zuweisung; der Wert von b wird dem Symbol a zugewiesen.
`values`	Liste aller mit einem Wert belegten Benutzervariablen
`kill(`x_1`,` x_2`,...)`	Löschen der Variablen x_1, x_2, ...
`ev(`*expr*`,` *opts*`)`	Auswerten des Ausdrucks *expr* mit den optionalen Parametern *opts*
expr`,` *opts*	Verkürzte Form des Auswertebefehls
	Optionen: *var*=*val* ... Einsetzen eines Wertes `eval` ... nochmaliges Auswerten `infeval` ... wiederholtes Auswerten `numer` ... Gleitkommadarstellung
`xthru(`*expr*`)`	Auf gleichen Nenner bringen
`ratsimp(`*expr*`)`	Vereinfachen und auf gleichen Nenner bringen
`expand(`*expr*`)`	Expandieren (Ausmultiplizieren)
`map(expand,`*expr*`)`	Zähler und Nenner getrennt expandieren
`num(`*expr*`)`, `denom(`*expr*`)`	Zähler bzw. Nenner von *expr*

Maxima als symbolischer Rechner

Die Manipulation von symbolischen Ausdrücken ist eine wesentliche Eigenschaft von Computeralgebra-Programmen, die sie von „gewöhnlicher“ Mathematik-Software unterscheidet.

Der Zuweisungsoperator ist in Maxima der Doppelpunkt „:“; damit können beliebige Ausdrücke Symbolen zugewiesen werden. Diese Zuweisung kann auch innerhalb einer Liste erfolgen; mit einem einzigen Ausdruck können somit beliebig viele Zuweisungen erfolgen. Es ist sinnvoll, *jedem* Ausdruck, der später noch benötigt wird, ein (möglichst „sprechendes“) Symbol zuzuweisen. Aus diesem Grund werden die meisten eingegebenen Ausdrücke Zuweisungen enthalten.

Beachte: Die versehentliche Verwendung des (gewohnten) Gleichheitsoperators „=“ für die Zuweisung ist ein häufiger Fehler, der bei ungeübten Benutzern leicht zu Verwirrung führt.

Die Manipulation und *Auswertung* von Ausdrücken kann in vielfältiger Weise mit verschiedensten Befehlen, insbesondere mit dem sehr universellen Befehl `ev` gesteuert werden.

Zuweisung zweier Zahlen zu Symbolen

```
(%i13) [a:5,b:12];
(%o13) [5, 12]
```

Zuweisung eines symbolischen Ausdrucks

```
(%i14) x1:a+b+c+d;
(%o14) d+c+17
```

Werden Symbole *nachträglich* mit Werten belegt, ...

```
(%i15) [c:25,d:13];
(%o15) [25,13]
```

... so bleiben sie in anderen Ausdrücken unausgewertet, ...

```
(%i16) x1;
(%o16) d+c+17
```

... deren Auswertung kann aber mit dem Befehl `ev` („evaluate“) erzwungen werden:

```
(%i17) ev(x1);
(%o17) 55
```

Liste mit allen Variablen, die mit einem Wert belegt sind.

```
(%i18) values;
(%o18) [a,b,x1,c,d]
```

Die Belegung von Symbolen kann mit `kill` wieder aufgehoben werden.

```
(%i19) kill(a,b,c,d);
(%o19) done
```

Liste mit allen belegten Variablen

```
(%i20) values;
(%o20) [x1]
(%i21) ev(x1);
(%o21) d+c+17
```

Symbolischer Ausdruck

```
(%i22) z:1/(x+1)+1/(y+2);
```

$$(\%o22)\quad \frac{1}{y+2}+\frac{1}{x+1}$$

Auf gleichen Nenner bringen

```
(%i23) xthru(z);
```

$$(\%o23)\quad \frac{y+x+3}{(x+1)(y+2)}$$

„Universal“-Vereinfachungsbefehl

```
(%i24) ratsimp(z);
```

$$(\%o24)\quad \frac{y+x+3}{(x+1)\,y+2\,x+2}$$

Expandieren

```
(%i25) expand(z);
```

$$(\%o25)\quad \frac{1}{y+2}+\frac{1}{x+1}$$

Kombination mehrerer Umwandlungsbefehle

```
(%i26) expand(ratsimp(z));
```

$$(\%o26)\quad \frac{y}{x\,y+y+2\,x+2}+\frac{x}{x\,y+y+2\,x+2}+\frac{3}{x\,y+y+2\,x+2}$$

Getrenntes Expandieren von Zähler und Nenner

```
(%i27) map(expand,ratsimp(z));
```

$$(\%o27)\quad \frac{y+x+3}{x\,y+y+2\,x+2}$$

Zähler und Nenner eines Ausdrucks

```
(%i28) [num(ratsimp(z)),denom(ratsimp(z))];
(%o28) [y+x+3,(x+1)y+2 x+2]
```

3.3 Logische Ausdrücke

`true`, `false`	Logische Konstanten *wahr* und *falsch*
`=`, `#`, `<`, `<=`, `>`, `>=`	Vergleichsoperatoren *gleich*, *ungleich*, *kleiner*, *kleiner gleich*, *größer*, *größer gleich*
`and`, `or`, `not`	Logische Operatoren *und*, *oder*, *nicht*
`equal(`*a*`,` *b*`)`	Überprüfung, ob die beiden Ausdrücke *a* und *b* den selben Wert ergeben
`is(`*expr*`)`	Auswerten des Vergleichsausdrucks *expr* zu `true`, `false` oder `unknown`

Logische Ausdrücke

Manche Ausdrücke, wie zum Beispiel Vergleichsausdrücke mit den Operatoren = (gleich), # (ungleich), < (kleiner), > (größer) etc. oder logische Ausrücke mit den Operatoren `not`, `and`, `or` liefern ein logisches Ergebnis, das einen der beiden Werte *false* („falsch“) oder *true* („wahr“) annehmen kann. Zusätzlich gibt es einen dritten logischen Wert, `unknown`, den ein Ausdruck dann liefert, wenn ein Ergebnis keinen eindeutigen Wert (*true* oder *false*) ergibt.

Logische Ausdrücke werden so weit wie möglich ausgewertet, Vergleichsausdrücke im Allgemeinen nicht, außer in Bedingungen und als Teil von logischen Ausdrücken. Ihre Auswertung kann mit der Funktion `is(`*Ausdruck*`)` veranlasst werden.

Belegen einer Variable

```
(%i29) u:5;
(%o29) 5
```

Vergleichsausdrücke bleiben unausgewertet.

```
(%i30) [u=4,u#4,u<4,u>4];
(%o30) [5=4,5#4,5<4,5>4]
```

Mit der Funktion `is` wird eine Auswertung erzwungen.

```
(%i31) [is(u=4),is(u#4),is(u<4),is(u>4)];
(%o31) [false,true,false,true]
```

Logische Ausdrücke werden ausgewertet.

```
(%i32) u<4 and true;
(%o32) false
```

Logischer Ausdruck mit Konstanten

```
(%i33) true or false;
(%o33) true
```

Ein Symbol in einem logischen Ausdruck wird selbst als logischer Ausdruck angesehen.

```
(%i34) xxx and true;
(%o34) xxx
```

3.4 Listen

Mit eckigen Klammern werden mehrere Ausdrücke zu einer *Liste* zusammengefasst; der Zugriff auf ein Listenelement erfolgt über einen ganzzahligen Index in eckigen Klammern, der bei 1 beginnt. Dieser Zugriff kann sowohl *lesend* (beispielsweise auf der rechten Seite einer Zuweisung) als auch *schreibend* (auf der linken Seite einer Zuweisung) erfolgen.

$[x_1, x_2, \ldots x_n]$	Zusammenfassung der Ausdrücke x_1, x_2 ... x_n zu einer Liste
$w[n]$	n-tes Element der Liste w (Referenz)
first(w), second(w), ... tenth(w)	erstes, zweites, ... zehntes Element der Liste w (Wert)
last(w)	letztes Element der Liste w (Wert)
map(f, w)	„Abbilden“ der Funktion f auf jedes Element der Liste w
apply(op, w)	Anwenden des Operators op auf die Elemente der Liste w
makelist($expr$, k, k_1, k_2, d_k)	Erzeugen einer Liste aus dem Ausdruck *expr*, der die Variable k enthält, die alle Werte von k_1 bis k_2 im Abstand von d_k durchläuft (d_k und k_1 sind optional).
create_list($expr$, k, w)	Erzeugen einer Liste aus dem Ausdruck *expr*, der die Variable k enthält, die alle Werte der Liste w annimmt.

Listen

Wird ein Index auf ein Symbol angewendet, das keine Liste ist, so entsteht eine sogenannte *Arrayvariable*, die in weiten Bereichen wie eine *indizierte* Variable verwendet werden kann.

Manche Funktionen werden, wenn sie eine Liste als Parameter enthalten, automatisch auf jedes Listenelement angewendet; man spricht hier von *Abbilden* (engl. „map“) einer Funktion auf eine Liste, was ein Grundprinzip der *funktionalen Programmierung* ist. Mit der Funktion `map` kann das Abbilden einer Funktion auf eine Liste erzwungen werden, wenn dies nicht automatisch geschieht.

`apply` wendet einen Operator auf alle Listenelemente an; das heißt, alle Listenelemente werden mit diesem Operator zu einem einzigen Ausdruck verknüpft.

Aufbau einer Liste

```
(%i35) liste:[0,%pi/12,%pi/6,%pi/4];
(%o35) [0, π/12, π/6, π/4]
```

Anzahl der Listenelemente

```
(%i36) length(liste);
(%o36) 4
```

Zugriff auf ein Listenelement

```
(%i37) liste[2];
(%o37) π/12
```

Kein Listenelement, sondern eine indizierte Variable

```
(%i38) x[2];
(%o38) x2
```

Manche Funktionen werden auf jedes Listenelement angewendet.

```
(%i39) floor(6*liste);
(%o39) [0, 1, 3, 4]
```

Manche Funktionen übernehmen eine Liste als einen einzigen Parameter.

```
(%i40) round(6*liste);
(%o40) round([0, π/2, π, 3 π/2])
```

Undefinierte Funktionen werden nicht auf jedes Listenelement angewendet, ...

```
(%i41) f(liste);
(%o41) f([0,π/12,π/6,π/4])
```

... was aber mit der Funktion `map` erzwungen werden kann.

```
(%i42) map(f,liste);
(%o42) [f(0),f(π/12),f(π/6),f(π/4)]
```

Erzeugen einer Liste mit einem Zählindex

```
(%i43) makelist(1/(s+k),k,0,2,0.4);
(%o43) [1/s,1/(s+0.4),1/(s+0.8),1/(s+1.2),1/(s+1.6),1/(s+2.0)]
```

Erzeugen einer Liste mit den Elementen einer anderen Liste

```
(%i44) create_list(k**2+1,k,[aa,bb,cc]);
(%o44) [aa^2+1,bb^2+1,cc^2+1]
```

3.5 Komplexe Zahlen

`%i`	Imaginäre Einheit
`realpart(z)`	Realteil des komplexen Ausdrucks z
`imagpart(z)`	Imaginärteil von z
`cabs(z)`	Betrag von z
`carg(z)`	Argument von z
`rectform(z)`	Umwandlung von z in die kartesische Koordinatendarstellung
`polarform(z)`	Umwandlung von z in die Exponentialform

Komplexe Zahlen

Maxima beherrscht das Rechnen mit komplexen Ausdrücken, die imaginäre Einheit wird als `%i` dargestellt. Ungebundene Variablen – das sind Variablen, denen kein Wert zugewiesen wurde – werden bei Berechnungen als reell angenommen.

Die Berechnung von Betrag und Argument erfolgt mit den Funktionen `cabs` bzw. `carg`. Ist der Quadrant eines komplexen Ausdrucks nicht festgelegt, so erfolgt die Angabe des Arguments mit der Funktion `atan2(y,x)`, sonst mit dem gewöhnlichen Arcustangens `atan(y/x)`.

Die Berechnung von Real- und Imaginärteil erfolgt mit den Funktionen `realpart` bzw. `imagpart`. Beliebige komplexe Ausdrücke können sowohl in die kartesische Koordinatendarstellung als auch in die Pokarkoordinatendarstellung umgewandelt werden.

Imaginäre Einheit

```
(%i45) sqrt(-1);
(%o45) %i
```

Betrag eines komplexen Ausdrucks

```
(%i46) cabs(a+%i*b);
(%o46) sqrt(b^2+a^2)
```

Argument eines komplexen Ausdrucks mit der Funktion `atan2`

```
(%i47) carg(a+%i*b);
(%o47) atan2(b,a)
```

Einschränkung des Wertebereiches für Variablen

```
(%i48) assume(a>0,b>0);
(%o48) [a>0,b>0]
```

Argument bei bekanntem Quadranten

```
(%i49) carg(a+%i*b);
(%o49) atan(b/a)
```

Für bestimmte Werte werden symbolische Ergebnisse gefunden.

```
(%i50) carg(1+%i*sqrt(3));
(%o50) π/3
```

Eulersche Formel, sie vereinigt die mathematischen Konstanten e, i und π in einem Ausdruck.

```
(%i51) %e^(%i*%pi)+1;
(%o51) 0
```

Ein komplexer Ausdruck

```
(%i52) z:(1+%i)/(5-2*%i);
(%o52) (%i+1)/(5-2 %i)
```

Berechnung des Realteils

```
(%i53) realpart(z);
(%o53) 3/29
```

Berechnung des Imaginärteils

```
(%i54) imagpart(z);
(%o54) 7/29
```

Darstellung in kartesischen Koordinaten (Real- und Imaginärteil)

```
(%i55) rectform(z),numer;
(%o55) 0.241 %i+0.103
```

Darstellung in Polarkoordinaten (Betrag und Argument)

```
(%i56) polarform(z),numer;
(%o56) 0.263 %e^(1.1659 %i)
```

3.6 Summen und Grenzwerte

`inf`, `minf`	Symbole für *unendlich* und *negativ unendlich*
`sum(`*expr*`,`*i*`,`i_0`,`i_1`)`	Summe $\sum_{i=i_0}^{i_1} expr$
`limit(`*expr*`,`*i*`,`i_0`)`	Grenzwert $\lim_{i\to i_0} expr$
`simpsum`	Systemvariable, veranlasst die Berechnung von Summen mit symbolischen Indexgrenzen (Default: `false`).

Summen und Grenzwerte

Maxima beherrscht die Berechnung von Summenausdrücken, die so weit wie möglich ausgewertet werden, sowie die Berechnung von Grenzwerten. Die Zahlen „unendlich“ (∞) und „minus unendlich“ ($-\infty$) werden mit den Ausdrücken `inf` bzw. `minf` angegeben.

Summe, die ein berechenbares Ergebnis liefert

```
(%i57) sum(1/2^n,n,1,10);
(%o57) 1023/1024
```

Summe mit symbolischen Summanden

```
(%i58) sum(x[k],k,1,5);
(%o58) x5+x4+x3+x2+x1
```

Ist die Anzahl der Summanden nicht exakt (als Ganzzahl) festgelegt, so bleibt der Summenausdruck unausgewertet.

```
(%i59) sum(x[k],k,1,N);
(%o59) sum(x[k],k,1,N)
```

Wird `simpsum` auf `true` gesetzt, werden auch Summen mit symbolischen Indexgrenzen berechnet.

```
(%i60) sum(1/n^2,n,1,inf),simpsum=true;
(%o60) %pi^2/6
```

Grenzwert für eine unbestimmte Form („0/0“)

```
(%i61) limit(sin(x)/x,x,0);
(%o61) 1
```

Definition der Eulerschen Zahl *e*

```
(%i62) limit((1+1/n)^n,n,inf);
(%o62) %e
```

3.7 Differenzial- und Integralrechnung

`diff(`*expr*`,` *x*`[,` *n*`])`	n-te Ableitung des Ausdrucks *expr* nach der Variablen *x*; die Angabe von *n* ist optional, Default: 1.
`integrate(`*expr*`,` *x*`)`	Unbestimmtes Integral des Ausdrucks *expr* mit der Integrationsvariablen *x*
`integrate(`*expr*`,` *x*`,` x_0`,` x_1`)`	Bestimmtes Integral des Ausdrucks *expr* mit der Integrationsvariablen *x* zwischen den Grenzen x_0 und x_1
`romberg(`*expr*`,` *x*`,` x_0`,` x_1`)`	Numerische Integration von *expr* mit dem Romberg-Verfahren
`quad_qags(`*expr*`,` *x*`,` x_0`,` x_1`)`	Numerische Berechnung des bestimmten Integrals *expr* mit der Integrationsvariablen *x* zwischen den Grenzen x_0 und x_1

Differenzial- und Integralrechnung

Mit der Funktion `diff` können beliebige Ausdrücke nach einer Variablen differenziert werden, als zusätzlicher Parameter kann die Ordnung der Ableitung angegeben werden (Default: 1. Ableitung).

Mit der Funktion `integrate` können unbestimmte und bestimmte Integrale berechnet werden. Ist die Berechnung eines Integrals nicht möglich, so bleibt das Integral unausgewertet. In diesem Fall kann aber auf numerische Integrationsmethoden, zum Beispiel auf die Funktionen `romberg` oder `quad_qags` zurückgegriffen werden.

Definition eines von x abhängigen Ausdrucks

```
(%i63) z:(x-1)/(x+5);
```

$$(\%o63)\quad \frac{x-1}{x+5}$$

Erste Ableitung

```
(%i64) diff(z,x);
```

$$(\%o64)\quad \frac{1}{x+5}-\frac{x-1}{(x+5)^2}$$

Dritte Ableitung

```
(%i65) diff(z,x,3);
```

$$(\%o65)\quad \frac{6}{(x+5)^3}-\frac{6\,(x-1)}{(x+5)^4}$$

Unbestimmtes Integral

```
(%i66) integrate(z,x);
(%o66) x-6 log(x+5)
```

Bestimmtes Integral zwischen den Grenzen 0 und 10

```
(%i67) integrate(z,x,0,10);
(%o67) -6 log(15)+6 log(5)+10
```

Ausgabe als Gleitkommazahl

```
(%i68) integrate(z,x,0,10),numer;
(%o68) 3.4083
```

Für dieses Integral gibt es keine geschlossene Lösung.

```
(%i69) integrate(sin(x)/(x**2+x+1),x,0,%pi);
```

$$(\%o69)\quad \int_0^{\pi}\frac{\sin(x)}{x^2+x+1}\,\mathrm{d}\,x$$

Numerische Integration nach dem Romberg-Verfahren

```
(%i70) romberg(sin(x)/(x**2+x+1),x,0,%pi);
(%o70) 0.504
```

Numerische Integration mit einer Quadpack-Funktion

```
(%i71) quad_qags(sin(x)/(x**2+x+1),x,0,%pi);
(%o71) [0.504,1.22306 10^-9,21,0]
```

3.8 Einschränken des Bereiches von Variablen

`assume(`vgl_1`,`vgl_2`,...)`	Treffen von Annahmen für den Bereich von Variablen in Form von Vergleichsausdrücken vgl_1, vgl_2, ...
`forget(`vgl_1`,`vgl_2`,...)`	Aufheben von Annahmen, die mit `assume` getroffen wurden
`facts()`	Liste mit allen getroffenen Annahmen über den Bereich von Variablen

Einschränken des Bereiches von Variablen

Mit der Anweisung `assume` kann der Wertebereich von Variablen eingeschränkt werden. Mitunter kann so eine weitergehende Auswertung von Ausdrücken erreicht werden, beispielsweise bei der Ermittlung des Maximums mehrerer Werte oder bei der Auswertung von Ausdrücken, die einen logischen Wert ergeben. Mit der Anweisung `forget` kann diese Einschränkung wieder aufgehoben werden.

Manchmal hängen Rechenergebnisse nicht nur *numerisch*, sondern auch *strukturell* von Werten von Variablen ab. In diesem Fall müssen bei der Berechnung Fallunterscheidungen getroffen werden. Maxima fragt dann den Benutzer nach anzunehmenden Wertebereichen der entsprechenden Variablen. Wird der anzunehmende Wertebereich im vorhinein entsprechend festgelegt, so kann die Berechnung ohne nachfragende Unterbrechung erfolgen.

Bei ungebundenen Symbolen ist die Bestimmung des Maximums nicht möglich.

```
(%i72) max(3,5,a,b);
(%o72) max(5, a, b)
```

Einschränkung des Bereiches für zwei Variable:

```
(%i73) assume(a>10,b<a);
(%o73) [a > 10, a > b]
```

Damit wird die Ermittlung des Maximums möglich.

```
(%i74) max(3,5,a,b);
(%o74) a
```

Aufhebung der Bereichseinschränkung für eine Variable:

```
(%i75) forget(a>b);
(%o75) [a > b]
```

Die Ermittlung des Maximums ist damit nicht mehr möglich.

```
(%i76) max(3,5,a,b);
(%o76) max(a, b)
```

Integral, dessen Berechnung vom Wertebereich einer Variablen abhängt.

```
(%i77) integrate(sin(t)*exp(-s*t),t,0,inf);
Is  s  positive, negative, or zero? p;
(%o77)  1/(s^2+1)
```

Wird der Wertebereich dieser Variablen im vorhinein festgelegt, ...

```
(%i78) assume(s>0);
(%o78) [s > 0]
```

... so erfolgt die Berechnung ohne unterbrechende Nachfrage.

```
(%i79) integrate(sin(t)*exp(-s*t),t,0,inf);
(%o79)  1/(s^2+1)
```

Liste mit allen getroffenen Annahmen über den Bereich von Variablen

```
(%i80) facts();
(%o80) [a > 0, b > 0, a > 10, s > 0]
```

3.9 Gleichungen

`solve` ist der grundlegende Befehl zur Lösung von Gleichungen und Gleichungssystemen. Die Lösungen werden in einer Liste mit Ausdrücken der Form *Variable=Wert* zurückgegeben. Zum bequemen Zugriff auf die Lösungen wird diese Liste zweckmäßigerweise einer Variablen zugewiesen, der Befehl `solve` also als rechter Teil einer Zuweisung aufgerufen.

Bei Gleichungs*systemen* ist diese Liste zweifach geschachtelt, jeder Satz von Lösungen ist in einer Unterliste zusammengefasst.

Für Polynome höherer als vierter Ordnung ist eine geschlossene Lösung im Allgemeinen nicht möglich. Mit der Funktion `allroots` werden alle (komplexen) Nullstellen eines Polynoms beliebiger Ordnung *numerisch* gefunden.

Transzendente Gleichungen besitzen im Allgemeinen keine geschlossene Lösung. Eine numerische Näherung für eine Nullstelle einer transzendenten Funktion innerhalb eines vorgegebenen Intervalls kann mit der Funktion `find_root` gefunden werden. Dabei sind die Intervallgrenzen, innerhalb derer sich die zu berechnende Nullstelle befindet, anzugeben.

`solve(`*eqn*`,` *var*`)`	Lösen der algebraischen Gleichung *eqn* nach der Variablen *var*
`solve(`*eqns*`,` *vars*`)`	Lösen einer Liste *eqns* von algebraischen Gleichungen nach den Variablen in der Liste *vars*
`allroots(`*p*`)`	Numerische Berechnung *aller* Nullstellen des Polynoms *p*
`find_root(`*expr*`,` x_1`,` x_2`)`	Numerische Ermittlung einer Nullstelle des Ausdrucks *expr* innerhalb des Intervalls $x_1 \ldots x_2$

Gleichungen

Eine quadratische Gleichung liefert zwei Lösungen.

```
(%i81) res1:solve((x+5)/(x^2-a)=2,x);
```
$$(\%o81)\quad \left[x=-\frac{\sqrt{16\,a+41}-1}{4},\ x=\frac{\sqrt{16\,a+41}+1}{4}\right]$$

Um eine Lösung als *Wert* zu erhalten, ist die entsprechende Variable mit dem Befehl `ev` und der Angabe der Lösung auszuwerten.

```
(%i82) ev(x,res1);
```
$$(\%o82)\quad -\frac{\sqrt{16\,a+41}-1}{4}$$

Zwei lineare Gleichungen in zwei Variablen

```
(%i83) g1:2*x+y=3;
       g2:2*x-4*y=5;
```
$$(\%o83)\quad y+2\,x=3$$
$$(\%o84)\quad 2\,x-4\,y=5$$

Die Lösung für beide Variablen wird in einer doppelt geschachtelten Liste zurückgeliefert.

```
(%i85) res2:solve([g1,g2],[x,y]);
```
$$(\%o85)\quad \left[\left[x=\frac{17}{10},\ y=-\frac{2}{5}\right]\right]$$

Zugriff auf eine Lösungsvariable mit der Anweisung `ev`:

```
(%i86) ev(x,res2);
```
$$(\%o86)\quad \frac{17}{10}$$

Quadratische Gleichung in zwei Variablen:

```
(%i87) g1:2*x^2+x-y^2=1;
```
$$(\%o87)\quad -y^2+2\,x^2+x=1$$

Auch für nichtlineare Gleichungssysteme können Lösungen gefunden werden.

```
(%i88) res3:solve([g1,g2],[x,y]);
```
$$(\%o88)\quad \left[\left[x=-\frac{4\sqrt{23}+9}{14},\ y=-\frac{\sqrt{23}+11}{7}\right],\left[x=\frac{4\sqrt{23}-9}{14},\ y=\frac{\sqrt{23}-11}{7}\right]\right]$$

Für ein Polynom 5. Ordnung kann `solve` keine Nullstellen ermitteln.

```
(%i89) res4:solve(x^5-x^3+2*x^2+5*x-1=0,x);
(%o89) [0=x^5-x^3+2*x^2+5*x-1]
```

Numerische Berechnung der Nullstellen eines Polynoms:

```
(%i90) res5:allroots(x^5-x^3+2*x^2+5*x-1);
(%o90) [x=0.187, x=0.719 %i-1.2334, x=-0.719 %i-1.2334, x=1.1496 %i+1.1398, x=1.1398-1.1496 %i]
```

Berechnung einer Nullstelle einer transzendenten Funktion innerhalb eines Intervalls:

```
(%i91) res6:find_root(exp(-x)=x,x,0,10);
(%o91) 0.567
```

3.10 Benutzerdefinierte Funktionen

f(x_1, x_2, ...):=*expr*	Definition einer Funktion f mit den formalen Parametern x_1, x_2, ... und dem Funktionskörper *expr*

Benutzerdefinierte Funktionen

Mit dem Operator „:=“ können eigene Funktionen definiert werden, die formalen Parameter sind hinter dem Funktionsnamen in runden Klammern (durch Beistriche getrennt) anzugeben. Beim Aufruf werden diese formalen Parameter durch die aktuellen Parameter, die Zahlen oder beliebige Ausdrücke sein können, ersetzt. Auch wenn eine Funktion ausnahmsweise überhaupt keine Parameter hat, so sind die Klammern (wie in der Programmiersprache C, als Kennzeichen einer Funktion) trotzdem anzugeben.

Wird ein ungebundenes Symbol, das heißt ein Symbol, dem noch nichts zugewiesen wurde, wie eine Funktion (also mit Parametern) verwendet, so wird es als *symbolische Funktion* angesehen. Mit symbolischen Funktionen kann genauso gerechnet werden, wie mit symbolischen Werten.

Definition einer Funktion mit dem formalen Parameter *t*

```
(%i92) f(t):=sin(t)*exp(-5*t);
```

$$(\%o92)\quad \mathrm{f}(t):=\sin(t)\,\exp((-5)\,t)$$

Verwendung der Funktion mit unterschiedlichen aktuellen Parametern

```
(%i93) [diff(f(x),x),integrate(f(omega),omega)];
```

$$(\%o93)\quad \left[\%e^{-5\,x}\cos(x)-5\,\%e^{-5\,x}\sin(x),\ \frac{\%e^{-5\,\omega}\left(-5\sin(\omega)-\cos(\omega)\right)}{26}\right]$$

Definition einer Funktion in drei Variablen

```
(%i94) f(x,y,z):=(2*x**2-y)/(z+1)^2;
```

$$(\%o94)\quad \mathrm{f}(x,y,z):=\frac{2\,x^2-y}{(z+1)^2}$$

Ermittlung eines Funktionswertes

```
(%i95) f(2,2,3);
```

$$(\%o95)\quad \frac{3}{8}$$

Auch das Rechnen mit *symbolischen* Funktionen ist möglich.

```
(%i96) diff(g(x)*sin(x),x);
```

$$(\%o96)\quad \sin(x)\left(\frac{\mathrm{d}}{\mathrm{d}\,x}\,\mathrm{g}(x)\right)+\mathrm{g}(x)\cos(x)$$

3.11 Vektoren und Matrizen

`matrix(`z_1`,` z_2`,` ... z_n`)`	Matrix, deren Zeilen aus den Listen $z_1, \ldots z_n$ gebildet werden.
A`[`i`,` j`]` *oder* A`[`i`][`j`]`	Element in der i-ten Zeile und der j-ten Spalte der Matrix A
A`.`B, A`^^`n	Matrixmultiplikation zweier Matrizen A, B und n-te Potenz der Matrix A
`invert(`A`)`	Inversion der Matrix A
`transpose(`A`)`	Transponierte von A
`args(`A`)`	Umwandlung der Matrix A in eine geschachtelte Liste
`apply(matrix,`*liste*`)`	Umwandlung der geschachtelten Liste *liste* in eine Matrix

Vektoren und Matrizen

Matrizen werden in Maxima nicht durch doppelt geschachtelte Listen dargestellt, sondern durch eine eigene Datenstruktur, die mit der Funktion `matrix` gebildet wird. Ein Vektor ist ein eindimensionaler Sonderfall einer Matrix (als Zeile oder Spalte). Die Standard-Rechenoperationen wirken elementweise, für die Matrixmultiplikation und das Potenzieren gibt es eigene Operatoren.

Erzeugung einer 3×3-Matrix

```
(%i97) m:matrix([1,2,0],[3,0,-5],[1,1,0]);
```

$$(\%o97)\begin{bmatrix} 1 & 2 & 0 \\ 3 & 0 & -5 \\ 1 & 1 & 0 \end{bmatrix}$$

Skalare Multiplikation (elementweise) und Matrixmultiplikation

```
(%i98) [m*m,m.m];
```

$$(\%o98)\left[\begin{bmatrix} 1 & 4 & 0 \\ 9 & 0 & 25 \\ 1 & 1 & 0 \end{bmatrix},\begin{bmatrix} 7 & 2 & -10 \\ -2 & 1 & 0 \\ 4 & 2 & -5 \end{bmatrix}\right]$$

Inversion einer Matrix

```
(%i99) invert(m);
```

$$(\%o99)\begin{bmatrix} -1 & 0 & 2 \\ 1 & 0 & -1 \\ -\frac{3}{5} & -\frac{1}{5} & \frac{6}{5} \end{bmatrix}$$

Der Zugriff auf ein Matrixelement erfolgt mit Zeilen- und Spaltenindex, jeweils in einer eckigen Klammer.

```
(%i100) m[2][3];
(%o100) -5
```

Spaltenvektor als Sonderfall einer Matrix

```
(%i101) v:matrix([xx],[yy],[zz]);
```

$$(\%o101)\quad \begin{bmatrix} xx \\ yy \\ zz \end{bmatrix}$$

Multiplikation einer Matrix mit einem Vektor

```
(%i102) m.v;
```

$$(\%o102)\quad \begin{bmatrix} 2\,yy + xx \\ 3\,xx - 5\,zz \\ yy + xx \end{bmatrix}$$

Umwandlung einer Matrix in eine geschachtelte Liste

```
(%i103) args(m);
(%o103) [[1,2,0],[3,0,-5],[1,1,0]]
```

Umwandlung einer geschachtelten Liste in eine Matrix

```
(%i104) apply(matrix,[[1,2],[3,4]]);
```

$$(\%o104)\quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

3.12 Zeichen und Zeichenketten

`cint(`*c*`)`	Ermittlung des (ganzzahligen) Codewertes 0...255 des Zeichens *c*
`ascii(`*n*`)`	Ermittlung des Zeichens aus dem ganzzahligen Codewert *n*
`slength(`*s*`)`	Anzahl der Zeichen der Zeichenkette *s*
`string(`*expr*`)`	Umwandlung des Ausdrucks *expr* in eine Zeichenkette
`eval_string(`*s*`)`	Umwandlung der Zeichenkette *s* in einen Ausdruck mit nachfolgender Auswertung
`charlist(`*s*`)`	Liste aus allen Zeichen der Zeichenkette *s*
`split(`*s*`[,` *delim*`[,` *mult*`]])`	Aufspalten der Zeichenkette *s* in eine Liste von Teilen
`sconcat(`$expr_1$`,` $expr_2$`, ...)`	Verkettung der Ausdrücke $expr_1$, $expr_2$, ... zu einer Zeichenkette

Zeichen und Zeichenketten

Eine Zeichenkette (oder ein Zeichen) wird bei der Eingabe durch Einschließen in doppelte Anführungszeichen gekennzeichnet; bei der Ausgabe erscheinen die Anführungszeichen aber standardmäßig nicht.

Einzelne Zeichen können in den ASCII-Code umgewandelt werden und umgekehrt, einige Sonderzeichen besitzen symbolische Namen.

Ein *Text* kann mit dem Befehl `eval_string` in einen *Ausdruck* umgewandelt und ausgewertet werden. Mit dem Befehl `string` wird ein beliebiger Ausdruck in eine Zeichenkette umgewandelt.

Mit dem Befehl `charlist` wird aus einer Zeichenkette eine Liste aus den einzelnen *Zeichen* erzeugt. Mit dem Befehl `split` wird eine Zeichenkette nach gewissen Regeln, beispielsweise bei den Leerzeichen, aufgespalten und die Elemente in einer Liste zurückgegeben. Das Verketten mehrerer Zeichenketten erfolgt mit dem Befehl `sconcat`.

Umwandlung eines Zeichens in den ASCII-Code und umgekehrt

```
(%i105) [cint("A"),ascii(66)];
(%o105) [65,B]
```

Einige Sonderzeichen und Steuerzeichen haben symbolische Namen.

```
(%i106) [cint(newline),cint(space)];
(%o106) [10,32]
```

Zeichenkette mit „mathematischer Bedeutung“

```
(%i107) text:"sin(%pi/6)";
(%o107) sin(%pi/6)
```

Ermittlung der Anzahl der Zeichen

```
(%i108) slength(text);
(%o108) 10
```

Auswertung der Zeichenkette als mathematischer Ausdruck

```
(%i109) eval_string(text);
(%o109) 1/2
```

Erzeugung einer Liste mit den einzelnen Zeichen

```
(%i110) charlist(text);
(%o110) [s,i,n,(,%,p,i,/,6,)]
```

Zeichenkette mit Zahlenwerten, durch Leerzeichen getrennt

```
(%i111) werte:"10 12.5 25 33.33";
(%o111) 10 12.5 25 33.33
```

Liste mit den Zahlenwerten als Zeichenketten

```
(%i112) liste:split(werte);
(%o112) [10,12.5,25,33.33]
```

Die einzelnen Elemente sind keine Zahlen, sondern Texte; daher kann nicht mit ihnen gerechnet werden.

```
(%i113) liste[2]*3;
(%o113) 3 12.5
```

Umwandlung in einen Ausdruck bewirkt Rechenfähigkeit.

```
(%i114) eval_string(liste[2])*3;
(%o114) 37.5
```

Definition zweier Zeichenketten:

```
(%i115) [vn:"Rudi",zn:"Huber"];
(%o115) [Rudi,Huber]
```

Zusammenketten mehrerer Zeichenketten

```
(%i116) sconcat(vn,space,zn);
(%o116) Rudi Huber
```

3.13 Grafiken

`load(draw);`	Laden des Grafikpakets *Draw*
`(wx)draw2d(`*opts*`, ..., `*obj*`, ...)`	2D-Grafik aus Grafikobjekten *obj* mit Optionen *opts*.
`(wx)draw3d(`*opts*`, ..., `*obj*`, ...)`	3D-Grafik aus Grafikobjekten *obj* mit Optionen *opts*.
`with_slider_draw(`t`,`w`,`*opts*`, ..., `*obj*`, ...)`	Animation mit dem Parameter t aus einer Liste w von Werten mit Grafikobjekten *obj* und Optionen *opts*.
Wichtige Grafikobjekte:	
`explicit(`$f(x)$`,`x`,`x_1`,`x_2`)`	Funktion $f(x)$ zwischen den Werten x_1 und x_2
`parametric(`$x(t)$`,`$y(t)$`,`t`,`t_1`,`t_2`)`	Kurve $x(t), y(t)$ in Parameterdarstellung mit dem Parameter t zwischen den Werten t_1 und t_2
`implicit(`*equation*`,`x`,`x_1`,`x_2`,`y`,`y_1`,`y_2`)`	Implizite Kurve *equation* in den Variablen x und y innerhalb der Bereiche $x_1 \ldots x_2$ und $y_1 \ldots y_2$
`points(`*xwerte*`,`*ywerte*`)`	Punkte; *xwerte* und *ywerte* sind Listen mit den x- bzw. y-Werten; bei 3D-Grafiken gibt es eine dritte Liste mit den z-Werten.
`points(`p_1`,`p_2`,...)`	Punkte; jeder Punkt p_i ist eine Liste aus zwei oder drei Koordinatenwerten.
Wichtige Optionen:	
`dimensions=[`*width*`,`*height*`]`	Abmessungen der Grafik in Bildpunkten (bei Pixelgrafiken)
`color=`*name*	Zeichenfarbe für Linien
`x(yz)range=[`*min*`,`*max*`]`	Darstellungsbereich auf der x(yz)-Achse
`grid=true/false`	Zeichnen von Gitterlinien in der xy-Ebene
`line_width=`*width*	Linienbreite
`point_size=`*size*	Größe von Punkten bei Punkt-Plots
`point_type=`*n*	Punkt-Typ, mögliche Werte -1,0,1,2,... 13.
`points_joined=true/false`	Angabe, ob Punkte durch Linienzüge verbunden werden (default: `false`)

Grafiken

Zur Darstellung von Grafiken verwendet Maxima das Programm *Gnuplot*, das bei der Erstellung der Grafik aufgerufen wird. Bei Verwendung der Benutzerschnittstelle wxMaxima können die Grafiken direkt in das Arbeitsfenster gezeichnet werden, was durch Voranstellen der Buchstaben „`wx`“ an die Namen der Plotroutinen erreicht wird.

Zwei verschiedene Gnuplot-Interfaces stehen zur Verfügung: die Standardfunktionen von Maxima, sowie die Funktionen des Zusatzpaketes *Draw* mit dem Wortstamm `draw` in den Funktions-

namen. Das Zusatzpaket *Draw* ist vor der erstmaligen Verwendung in einer Arbeitssitzung mit dem Befehl `load` zu laden.

wxMaxima ermöglicht das Erstellen einfacher Animationen; nach Anklicken der Grafik im Arbeitsblatt kann die Animation mit den entsprechenden Schaltflächen und einem Schieber in der Werkzeugleiste oder dem Scrollrad der Maus gesteuert werden.

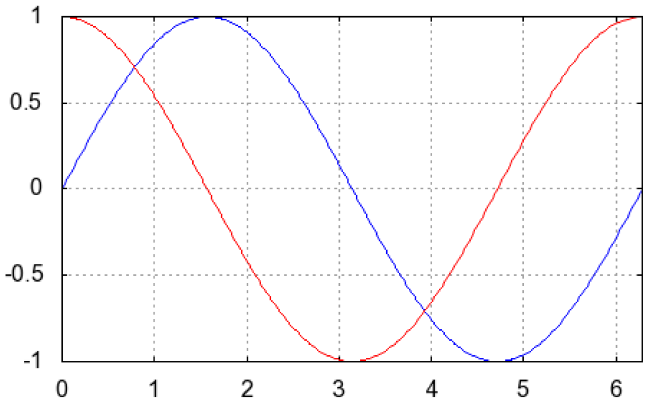
Laden des Grafikpakets *Draw*

```
(%i117) load(draw);
(%o117)
C:/Programme/Maxima-5.27.0/share/maxima/5.27.0/ ...
```

Plotten zweier Funktionen

```
(%i118) wxdraw2d(color=blue,explicit(sin(t),t,0,2*%pi),
        grid=true,color=red,explicit(cos(t),t,0,2*%pi))$
```

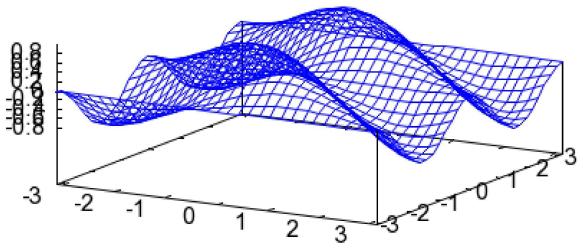
(%t118)



Dreidimensionale Darstellung einer Funktion in zwei Variablen

```
(%i119) wxdraw3d(explicit(sin(x+y)*sin(y),
            x,-%pi,%pi,y,-%pi,%pi))$
```

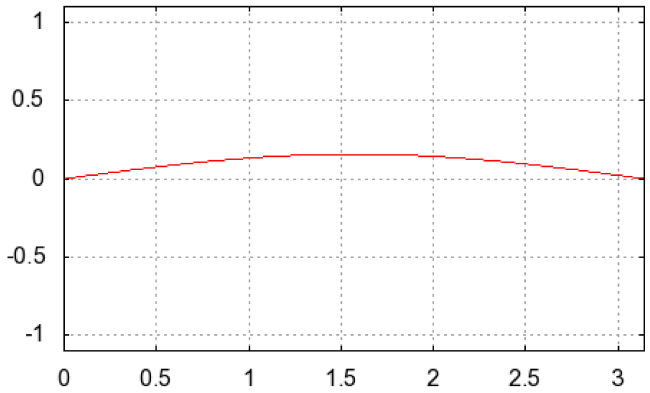
(%t119)



Animation

```
(%i120) with_slider_draw(t,makelist(t*%pi/20,t,20),
        color=red,yrange=[-1.1,1.1],grid=true,
        explicit(sin(x)*sin(t),x,0,%pi))$
```

(%t120)



3.14 Programmkontrollstrukturen

Umfangreiche Programmkontrollstrukturen machen Maxima zu einer vollständigen, sehr vielseitigen Programmiersprache.

$(x_1, x_2, \ldots x_n)$	Zusammenfassen der Ausdrücke x_1, x_2, ... x_n zu einer Sequenz
`block([`$v_1, v_2, \ldots v_n$`],`$x_1, x_2, \ldots x_n$`)`	Zusammenfassen der Ausdrücke x_1, x_2, ... x_n zu einem Block mit den lokalen Variablen v_1, v_2, ... v_n
`return(`x`)`	Verlassen des Blocks und Rückgabe von x als Ergebnis
`if` *bed* `then` $expr_1$ `else` $expr_2$	Bedingte Auswertung des Ausdrucks $expr_1$ oder $expr_2$ in Abhängigkeit der Bedingung *bed*; der else-Teil kann auch entfallen.
`for` z`:`z_0 `step` δz `thru` z_1 `do` *expr*	Wiederholte Ausführung von *expr*, wobei die Steuervariable z die Werte von z_0 bis z_1 mit der Schrittweite δz durchläuft.
`for` z `:`z_0 `step` δz `while` *bed* `do` *expr*	Wiederholte Ausführung von *expr* solange die Bedingung *bed* erfüllt ist.
`while` *bed* `do` *expr*	Verkürzte Form der Schleife

Programmkontrollstrukturen

Mit einer *Sequenz* oder einem *Block* können mehrere Ausdrücke – durch Beistriche getrennt – zusammengefasst werden; sie wirken nach außen wie ein einziger Ausdruck. Rückgabewert ist jeweils das Ergebnis des letzten Ausdrucks.

Innerhalb eines Blocks ist die Deklaration von lokalen Variablen möglich, außerdem kann ein

Block mit der Anweisung `return` mit Angabe eines Rückgabewertes an jeder Stelle vorzeitig verlassen werden.

Mit `if...then...else` können *Bedingungen* formuliert werden, und zwar sowohl im Stil bedingter *Anweisungen* (mit mehreren Ausdrücken, die bedingt ausgewertet werden), als auch im Stil bedingter *Ausdrücke* (mit einer Bedingung innerhalb eines einzigen Ausdrucks).

Mit `for...do` können auf unterschiedliche Weise *Schleifen* zur wiederholten Auswertung von Ausdrücken formuliert werden, was dem *prozeduralen* Programmierstil entspricht. Schleifen werden in Maxima aber selten benötigt; sie können durch Verwendung anderer Kostrukte, die dem *funktionalen* Programmierstil entsprechen, ersetzt werden (siehe Abschnitt **??**).

Zusammenfassung mehrerer Ausdrücke zu einer *Sequenz*; das Ergebnis ist der Wert des letzten Ausdrucks.

```
(%i121) (a:3,b:7,c:a*b);
(%o121) 21
```

Ein Block mit zwei lokalen Variablen; das Ergebnis ist der Wert des letzten Ausdrucks.

```
(%i122) block([a,b],a:5,b:12,c:a*b);
(%o122) 60
```

Zusammenfassung mehrerer Ausdrücke zu einer *Liste*; beachte: die lokalen Variablen des Blocks wirken *nicht* nach außen.

```
(%i123) [a,b,c];
(%o123) [3,7,60]
```

Definition einer Funktion mit einem bedingten Ausdruck

```
(%i124) f(t):=if t<0 then 0 else t;
(%o124) f(t):=if t<0 then 0 else t
```

Berechnung von Funktionswerten

```
(%i125) [f(-1),f(-0.5),f(0),f(0.5),f(1)];
(%o125) [0,0,0,0.5,1]
```

Wiederholtes Verändern einer Variablen x in einer Schleife

```
(%i126) for n:1 thru 3 do x:1+1/x;
(%o126) done
```

Endwert von x

```
(%i127) x;
```

$$(\%o127)\quad \cfrac{1}{\cfrac{1}{\cfrac{1}{x}+1}+1}+1$$