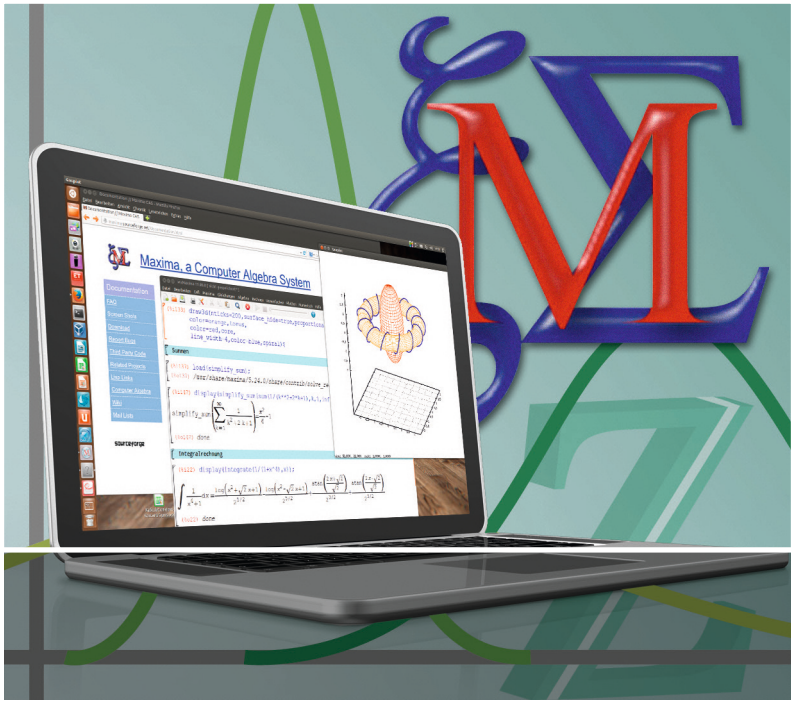
Wilhelm Haager

Computeralgebra mit Maxima

Grundlagen der Anwendung und Programmierung



2., aktualisierte Auflage

HANSER

Haager

Computeralgebra mit Maxima

Wilhelm Haager

Computeralgebra mit Maxima

Grundlagen der Anwendung und Programmierung

2., aktualisierte Auflage

HANSER

Autor:

Dipl.-Ing. Dr. Wilhelm Haager
HTL St. Pölten

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Internet: www.hanser-fachbuch.de

Lektorat: Frank Katzenmayer
Herstellung: Anne Kurth
Covergestaltung: Max Kostopoulos
Coverkonzept: Marc Müller-Bremer, www.rebranding.de, München
Satz: Wilhelm Haager
Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen
Printed in Germany

Print-ISBN 978-3-446-44868-1
E-Book-ISBN 978-3-446-46095-9

Vorwort

Durch die allgegenwärtige Verfügbarkeit hoher Rechenleistung in Form von PCs, Laptops und sogar Mobiltelefonen gewinnt der Einsatz von Computeralgebra zunehmend an Bedeutung, nicht nur für Wissenschaftler und Ingenieure, sondern für jeden, der sich in irgendeiner Form mit Rechnen und mathematischen Formeln beschäftigen muss.

Vor 45 Jahren brachte der Einzug des Taschenrechners auf den Schreibtischen von Wissenschaftlern, Ingenieuren, Studenten und Schülern Befreiung von *rechnerischer* Mühsal und damit eine wesentliche Erhöhung der Produktivität; Rechenschieber und Logarithmentafeln konnten auf der Müllhalde (oder im Museum) entsorgt werden. Die Anwendung von Computeralgebra befreit nun von *algorithmischer* Mühsal. Nicht nur das Berechnen von Werten, sondern auch das *Beschreiten* von Lösungswegen kann nun dem Rechner überlassen werden; üblicherweise ist der Rechner dabei wesentlich schneller und weniger fehleranfällig.

Glaubt man sich nun aber aller mathematischen Sorgen entledigt zu haben, so wird man auch bald die Grenzen der Computeralgebra erkennen; Computeralgebra kann menschliches Denken zwar bei Routineaufgaben ersetzen, nicht aber in Bereichen, die Kreativität erfordern: Das Finden von Ansätzen und *Formulieren* von Lösungswegen wird vermutlich immer menschlichen Denkens bedürfen.

Außerdem sollte man beim Einsatz von Computeralgebra folgenden Irrtümern nicht erliegen:

1. Dem Glauben, dass mit Computeralgebra *alles* berechenbar ist. Abgesehen davon, dass sehr viele Dinge prinzipiell nicht analytisch berechnet werden können, sind Berechnungsergebnisse mitunter so komplex, dass sie für die Praxis unbrauchbar sind. Wenn sich eine Lösungsformel über viele Seiten erstreckt und sich in ihr keine Zusammenhänge und Abhängigkeiten mehr erkennen lassen, so hat sie gegenüber einer numerisch (und vielleicht bequemer) gefundenen Lösung wahrscheinlich keinen Vorteil.
2. Dem Glauben des Lernenden, dass Computeralgebra eine rechnerische Routine, die nur durch Training erworben werden kann, überflüssig macht. Computeralgebra kann notwendiges Training zwar begleiten (etwa durch eine rasche Kontrolle von Ergebnissen), aber niemals ersetzen. Wer nicht in der Lage ist, mathematische Ausdrücke mit Papier und Bleistift zu manipulieren, wird weder in der Lage sein, kreative Ansätze und Lösungswege zu finden, noch Computeralgebra erfolgreich einzusetzen.

Das vorliegende Buch kann und will das mehr als tausendseitige Handbuch [Max], das eine vollständige Beschreibung aller Funktionen von Maxima enthält, nicht ersetzen. Zum *Lernen* sind Handbücher in ihrer üblichen, rein deduktiven Struktur aber nicht gut geeignet. Mit diesem Buch soll der Einstieg in Maxima erleichtert werden, indem es die Grundlagen thematisch geordnet, aber doch möglichst aufbauend, vermittelt. Nicht Vollständigkeit ist das Ziel, sondern eine übersichtliche Darstellung jener Prinzipien und Funktionen, die am ehesten benötigt werden.

Ich danke meinen Kollegen und Freunden an der Abteilung Elektrotechnik der HTL St. Pölten für wertvolle Diskussionen, deren Ergebnisse im Buch ihren Niederschlag fanden. Ganz be-

sonderer Dank gilt dabei Dr. Peter Zaniat. Er hat mich nicht nur wiederholt angehalten, das Buchprojekt konsequent und zügig weiterzuverfolgen, er hat auch mit der Akribie eines professionellen Lektors auf viele Tippfehler und Holprigkeiten im Lesefluss hingewiesen.

Ich bedanke mich bei den Damen und Herren des Carl Hanser Verlages für die perfekte Zusammenarbeit; insbesondere bei Herrn Dr. Martin Feuchte für die Bereitschaft, das Buch in das Verlagsprogramm aufzunehmen, bei den Lektorinnen Frau Mirja Werner M. A. und Frau Franziska Jacob M. A. sowie bei der Herstellerin Frau Dipl.-Ing. Franziska Kaufmann.

Möge das vorliegende Buch dazu beitragen, Maxima nicht nur einem breiten Kreis von Mathematikern und Technikern, Lehrenden und Lernenden bekannt zu machen, sondern auch Freude an dessen Verwendung zu wecken. Gelingt dies, so hat es seinen Zweck erfüllt.

Vorwort zur zweiten Auflage

Programmiersprachen sind langlebig; ein Buch über eine Programmiersprache – als solche kann auch *Maxima als Sprache* angesehen werden – bleibt daher über Jahre aktuell. Die *Bedienbarkeit* von Programmen hingegen ist ständigen Weiterentwicklungen unterworfen und wird üblicherweise von Version zu Version besser und bequemer; das betrifft auch *Maxima als Programm*.

Die größten Änderungen seit dem Entstehen der ersten Auflage gibt es bei der Benutzerumgebung *wxMaxima*, im Zusammenspiel mit dem Grafikprogramm *Gnuplot*, sowie bei der Unterstützung von Unicode. Obwohl sich Maxima als Sprache im Kern nicht verändert, wird doch der Funktionsumfang durch von der Benutzer- und Entwicklergemeinde geschriebenen Zusatzpaketen ständig erweitert. Auch dem wurde in der zweiten Auflage Rechnung getragen.

Möge das Buch auch weiterhin interessante „Maximale“ Erkenntnisse liefern.

Wieselburg und Innergschlöß, im August 2019 Wilhelm Haager

Inhalt

1 Einführung

The purpose of computing is insight, not numbers.

R. W. HAMMING

Computeralgebra-Programme gibt es seit den Sechzigerjahren des vorigen Jahrhunderts, ihr Einsatz blieb aber lange Zeit im Wesentlichen auf den akademischen Bereich beschränkt. Die billige Verfügbarkeit von PCs brachte zwar die Möglichkeit, Computeralgebra auch in Schulen und Unternehmen einzusetzen, die hohen Kosten kommerzieller Computeralgebra-Programme hemmten aber eine weite Verbreitung.

Mit dem Programm *Maxima* steht eine leistungsfähige, kostenlose Computeralgebra für jedermann zur Verfügung, die den Vergleich mit ihren kommerziellen Schwestern nicht zu scheuen braucht. Auf Grund der Leistungsfähigkeit und der freien Verfügbarkeit gibt es eigentlich keinen Grund, Maxima *nicht* zu verwenden. Mit einem breiten und alltäglich gewordenen Einsatz wird die Beherrschung von Computeralgebra ebenso zu einer selbstverständlichen Kulturtechnik werden, wie es heute schon die Bedienung eines Taschenrechners oder eines Textverarbeitungsprogrammes ist.

Das erste Kapitel des Buches erläutert die Benutzeroberfläche *wxMaxima* und vermittelt die wichtigsten Grundlagen in einem Ausmaß, dass nach dessen Durcharbeiten ein selbständiges und produktives Arbeiten mit dem Programm möglich sein sollte. Die weiteren Kapitel sind nach einzelnen Aspekten von Maxima oder mathematischen Themen systematisch geordnet, aber doch möglichst aufbauend. Sie können je nach Bedarf auch einzeln durchgearbeitet oder als Referenz konsultiert werden. Bezüge auf vorige Kapitel sind in einem aufbauenden Buch natürlich vorhanden, Bezüge auf spätere Kapitel gibt es nur wenige.

Jedes dieser Kapitel enthält die wichtigsten, jeweils thematisch zusammengehörigen Maxima-Befehle in einer tabellarischen Übersicht sowie Erläuterungen und kommentierte Beispiele in Form von zusammenhängenden Arbeitssitzungen. Diese Beispielsitzungen erstrecken sich manchmal auch aufbauend über mehrere Abschnitte. Für das Verständnis vieler Beispiele ist es hilfreich, den Verlauf einer Arbeitssitzung von Beginn an zu kennen, also gegebenenfalls auch unmittelbar vorangegangene Abschnitte zu konsultieren.

1.1 Grundlegendes

Maxima ist ein Open-Source-Abkömmling von *Macsyma*, einem der ersten Computeralgebra-Systeme. Macsyma wurde in den Jahren 1968–1982 am MIT 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 von Macsyma unter der *GNU General Public Licence* (GPL) veröffentlicht und somit jedermann kostenlos verfügbar gemacht. Diese Version wird nun unter dem Namen *Maxima* von einer unabhängigen Gruppe von Anwendern und Entwicklern gepflegt und – insbesondere im Bereich der Benutzerschnittstellen [Vod] und Grafikmöglichkeiten [Rod] – intensiv weiterentwickelt.

1.1.1 Motivation

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, ein wesentliches Argument im Ausbildungsbereich: Für Schulen fallen keine Lizenzgebühren an, die Schüler bekommen ein Werkzeug, das sie auch nach der Ausbildung kostenlos 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: Es unterstützt prozedurales Programmieren, funktionales Programmieren und (in beschränktem Ausmaß) auch regelbasiertes Programmieren. Die Funktionalität ist damit beliebig erweiterbar.

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.

1.1.2 Installation, Bestandteile

Maxima ist für alle gängigen Betriebssysteme verfügbar (Windows, Linux, MacOS, Android); Informationen, Dokumentationen und Links zu den Downloadseiten finden sich auf der Homepage (`http://wxmaxima.sourceforge.net`). Die Installation ist selbsterklärend, bei Problemen mit der Installation oder Ausführung bietet die Homepage Hilfestellung.

Maxima ist prinzipiell kommandozeilenorientiert, es stehen aber eine Reihe grafischer Benutzeroberflächen zur Verfügung, die sich vor allem in den Möglichkeiten der Darstellung von Grafiken unterscheiden. Dieses Buch stützt sich auf die Benutzeroberfläche *wxMaxima*, die bei der Installation von Maxima automatisch als Standardinterface mitinstalliert wird. Zur Erzeugung von Grafiken verwendet Maxima das Programm *Gnuplot*, das ebenfalls automatisch mitinstalliert wird (Bild 1.1).

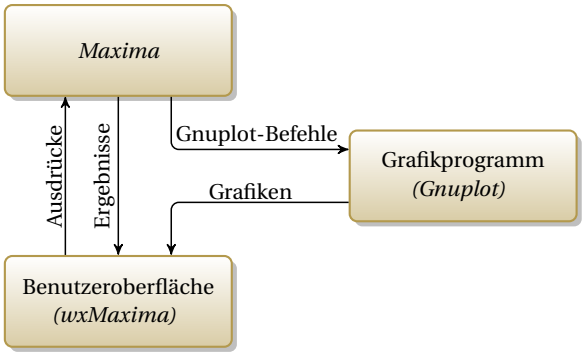
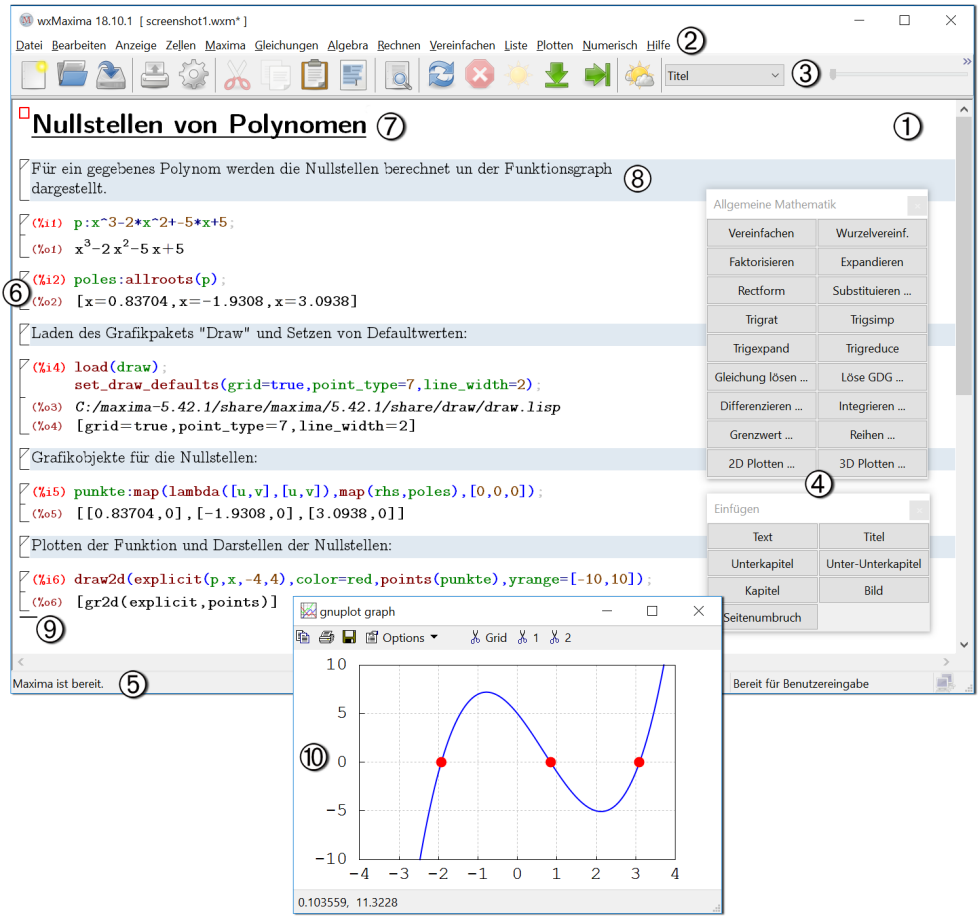


Bild 1.1 Bestandteile einer Maxima-Installation

1.1.3 Links

- `http://maxima.sourceforge.net`
  Homepage mit vielen Informationen, Dokumentationen und Zusatzpaketen
- `http://wxmaxima-developers.github.io`
  Homepage des Benutzer-Interfaces *wxMaxima* mit User-Forum
- `http://www.austromath.at/daten/maxima/`
  Maxima Online-Kurs von Walter Wegscheider
- `http://www.weilharter.info/`
  Homepage von Johann Weilharter mit vielen Maxima-Materialien
- `http://www.csulb.edu/~woollett/`
  Homepage von Ted Woollett mit ausführlichen Tutorials
- `http://maxima.cesga.es`
  Maxima Online-Webinterface

1.2 Benutzeroberfläche wxMaxima



① … Arbeitsfenster
② … Menüleiste
③ … Werkzeugleiste
④ … Befehlsschaltflächen
⑤ … Statusfeld
⑥ … Zellenklammer
⑦ … Titelzelle
⑧ … Textzelle
⑨ … Einfügecursor
⑩ … Gnuplot-Fenster

Bild 1.2 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 Mausklicker) den Aufruf der wichtigsten Befehle über Menüs und Befehlsschaltflächen.

1.2.1 Aufbau

Arbeitsfenster: Hier erfolgt die Eingabe sowohl von mathematischen Ausdrücken als auch von gewöhnlichem Text zur Gliederung und Dokumentation von Berechnungen. Mit der Eingabetaste *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 öffnen ein entsprechendes Kontextmenü.

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

Durch Anklicken aufgerufene Maxima-Befehle beziehen sich dabei immer auf den unmittelbar zuvor eingegebenen oder berechneten Ausdruck, der mit % bezeichnet wird.

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, Kapitelzelle, Abschnittszelle, etc.: Damit kann ein Arbeitsblatt mit Überschriften unterschiedlicher Größen optisch strukturiert werden. Die Kapitel, Abschnitte etc. werden automatisch nummeriert. Erzeugt werden sie durch Menübefehle [*Zellen*] oder die Tastenkürzel *Strg*+2, *Strg*+3 etc.

Textzelle: Dient zur Kommentierung einer Berechnung. Erzeugt wird eine Textzelle über einen Menübefehl oder mit dem Tastenkürzel *Strg*+1. Zusätzlich zu normalem Text ist die Eingabe von griechischen Buchstaben und mathematischen Sonderzeichen möglich; näheres hierzu findet sich im wxMaxima-Manual, aufzurufen über das Menü [*Hilfe*].

Einfügecursor: Kennzeichnet jene Stelle auf dem Arbeitsblatt durch einen blinkenden Balken, an der eine neue Eingabe erfolgt (Text oder zu berechnender Ausdruck). 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 *Strg*+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.

1.2.2 Abspeichern und Laden

Über das Menü [*Datei*][*Speichern unter ...*] kann eine wxMaxima-Arbeitssitzung auf zwei Arten abgepeichert werden:

wxm-File: Es enthält alle eingegebenen Ausdrücke sowie die Textzellen und Überschriften als zusätzliche Kommentare. Diese Kommentare werden zwar von *Maxima* bei einem späteren Einlesen ignoriert, von *wxMaxima* aber entsprechend interpretiert. Die Struktur des Arbeitsblattes mit Ausdrücken, Texten und Überschriften bleibt somit beim Abspeichern und Wiedereinlesen erhalten.

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 Inhalte (nicht nur die Eingaben, sondern auch die Berechnungsergebnisse) 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.

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* – bei einem wxmx-File auch die zuvor abgespeicherten Berechnungsergebnisse – auf dem Arbeitsblatt dargestellt, eine (neuerliche) Berechnung wird erst mit der Eingabe von *Strg*+`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.

1.2.3 Export

Über das Menü [*Datei*][*Exportieren ...*] kann eine wxMaxima-Arbeitssitzung in verschiedenen Formaten exportiert werden:

mac-File: Es enthält alle Eingaben einer Arbeitssizung, die beim Wiedereinlesen über das Menü [*Datei*][*Paket laden ...*] oder mit dem Befehl `load` eingelesen und sofort abgearbeitet werden (ohne Ausgabe von Ergebnissen).

Üblicherweise enthält ein Mac-File, ein sogenanntes *Paket* in erster Linie Funktionsdefinitionen und Variablenzuweisungen, aber kaum ausführbaren Code (siehe Abschnitt 7.5).

HTML: Beim Export einer Arbeitssitzung in HTML werden die Ausgaben von Maxima in der Auszeichnungssprache *MathML* ausgegeben, die Grafiken als png-Files in einem Verzeichnis mit dem Namen *name*`_img` abgespeichert.

LaTeX: Der Export einer Arbeitssitzung in LaTeX ist zwar möglich, funktioniert aber leider noch nicht einwandfrei. In den meisten Fällen ist ein händisches Nacheditieren des erzeugten tex-Files notwendig.

Sehr gut funktioniert hingegen die Umwandlung einzelner berechneter Ausdrücke in LaTeXdurch Markieren und das Menü [*Bearbeiten*][*Als LaTeX kopieren*]. Eine Umwandlung eines Ausdrucks in das TEX-Format ist auch mit den Maxima-Befehlen `tex` und `Tex1` möglich (siehe Abschnitt 7.2).

1.2.4 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 wxMaxima-Befehle können daher – alternativ zur Auswahl aus Menüs – als Tastenkürzel eingegeben werden, Tabelle 1.1 enhält eine Auswahl davon.

Tabelle 1.1 wxMaxima Tastenkürzel

Windows-spezifische Tastenkürzel:

Strg+O	Öffnen eines wxm-Files (und Beginn einer neuen Arbeitssitzung)
Strg+P	Drucken des Arbeitsblattes
Strg+Q	Beenden von wxMaxima
Strg+C	Kopieren des markierten Bereichs in die Zwischenablage
Strg+X	Ausschneiden des markierten Bereichs in die Zwischenablage
Strg+V	Einfügen des Inhalts der Zwischenablage
Strg+Z	Rückgängig machen

wxMaxima-spezifische Tastenkürzel:

Strg+R	Neuberechnung aller *sichtbaren* Zellen
Strg+*Shift*+R	Neuberechnung des gesamten Arbeitsblatts
Strg+L	Laden eines Pakets (mac-File)
Strg+B	Ausführen eines Maxima-Programms (mac-File)
Strg+G	Unterbrechen einer Berechnung
Strg+1	Erstellen einer neuen Textzelle
Strg+2	Erstellen einer Titelzelle
Strg+3	Erstellen einer Abschnittszelle
Strg+4	Erstellen einer Unterabschnittszelle
Strg+M	Mehrere Zellen vereinigen
Strg+D	Aufspalten in mehrere Zellen
Strg+K	Auto-Vervollständigen von Eingaben
Alt+I, *Strg*++	Vergrößern der Darstellung
Alt+O, *Strg*+−	Verkleinern der Darstellung

1.3 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 einem 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.

1.3.1 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
```

Eckige Klammern fassen mehrere Ausdrücke zu einer *Liste* zusammen.

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

Tabelle 1.2 Maxima als Taschenrechner

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 unterscheidet zwischen Ganzzahlen und Gleitkommazahlen.

```
(%i4) [4/6, 4/6.0];
(%o4) [2/3,0.6666666666666666 ]
```

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) [2/3,0.66667]
```

Darstellung von rationalen und irrationalen Zahlen

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

Umwandlung in die Gleitkommadarstellung mit der Funktion `float`

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

Umwandlung in die Gleitkommadarstellung 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),0.58779]
```

Abschneiden und Runden von Gleitkommazahlen

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

1.3.2 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.

Tabelle 1.3 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*

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]
```

Symbolischer Ausdruck

```
(%i21) z:1/(x+1)+1/(y+2);
```

$$(\%o21)\quad \frac{1}{y+2}+\frac{1}{x+1}$$

Auf gleichen Nenner bringen

```
(%i22) xthru(z);
```

$$(\%o22)\quad \frac{y+x+3}{(x+1)\,(y+2)}$$

„Universal“-Vereinfachungsbefehl

```
(%i23) ratsimp(z);
```

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

Expandieren

```
(%i24) expand(z);
```

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

Kombination mehrerer Umwandlungsbefehle

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

$$(\%o25)\quad \frac{y}{x\,y+y+2\,x+2}+\frac{x}{x\,y+y+2\,x+2}+\frac{3}{x\,y+y+2\,x+2}$$

Getrenntes Expandieren von Zähler und Nenner

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

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

Zähler und Nenner eines Ausdrucks

```
(%i27) [num(ratsimp(z)),denom(ratsimp(z))];
(%o27) [y+x+3,(x+1)y+2x+2]
```

In wxMaxima sind auch griechische Buchstaben als Variablennamen sowie Variablen mit tiefgestelltem Index möglich.

Griechische Buchstaben und Variablen und Variablen mit Index

```
(%i28) [alpha,beta,omega_m,t_ein];
(%o28) [α,β,ω_m,t_ein]
```

1.3.3 Logische Ausdrücke

Manche Ausdrücke, wie zum Beispiel Vergleichsausdrücke mit den Operatoren = (gleich), # (ungleich), < (kleiner), > (größer) etc. oder logische Ausdrücke 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.

Tabelle 1.4 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* denselben Wert ergeben
`is`(*expr*)	Auswerten des Vergleichsausdrucks *expr* zu `true`, `false` oder `unknown`

Belegen einer Variablen

```
(%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
```

1.3.4 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.

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.

Tabelle 1.5 Listen

$[x_1, x_2, \ldots x_n]$	Zusammenfassung der Ausdrücke x_i 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.

`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, %pi/12, %pi/6, %pi/4]
```

Anzahl der Listenelemente

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

Zugriff auf ein Listenelement

```
(%i37) liste[2];
(%o37) %pi/12
```

Kein Listenelement, sondern eine indizierte Variable

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

Arithmetische Operationen von Listen erfolgen elementweise, …

```
(%i39) liste+[1,2,3,4];
```
(%o39) $[1,\frac{\pi}{12}+2,\frac{\pi}{6}+3,\frac{\pi}{4}+4]$

… ebenso Operationen mit einem Skalar.

```
(%i40) liste+k;
```
(%o40) $[k,k+\frac{\pi}{12},k+\frac{\pi}{6},k+\frac{\pi}{4}]$

Viele Funktionsaufrufe werden auf die Listenelemente „abgebildet“, …

```
(%i41) float(liste);
```
(%o41) [0.0,0.2618,0.5236,0.7854]

… nicht aber undefinierte Funktionen, …

```
(%i42) f(liste);
```
(%o42) $f\left([0,\frac{\pi}{12},\frac{\pi}{6},\frac{\pi}{4}]\right)$

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

```
(%i43) map(f,liste);
```
(%o43) $[f(0),f\left(\frac{\pi}{12}\right),f\left(\frac{\pi}{6}\right),f\left(\frac{\pi}{4}\right)]$

Erzeugen einer Liste mit einem Zählindex

```
(%i44) makelist(1/(s+k),k,0,2,0.4);
```
(%o44) $[\frac{1}{s},\frac{1}{s+0.4},\frac{1}{s+0.8},\frac{1}{s+1.2},\frac{1}{s+1.6},\frac{1}{s+2.0}]$

Erzeugen einer Liste mit den Elementen einer anderen Liste

```
(%i45) create_list(k**2+1,k,[aa,bb,cc]);
```
(%o45) $[aa^2+1,bb^2+1,cc^2+1]$

1.3.5 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`)`.

Tabelle 1.6 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

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 Polarkoordinatendarstellung umgewandelt werden.

Imaginäre Einheit

```
(%i46) sqrt(-1);
(%o46) %i
```

Betrag eines komplexen Ausdrucks

```
(%i47) cabs(a+%i*b);
(%o47) sqrt(b^2+a^2)
```

Argument eines komplexen Ausdrucks mit der Funktion `atan2`

```
(%i48) carg(a+%i*b);
(%o48) atan2(b,a)
```

Einschränkung des Wertebereichs für Variablen

```
(%i49) assume(a>0,b>0);
(%o49) [a>0,b>0]
```

Argument bei bekanntem Quadranten

```
(%i50) carg(a+%i*b);
(%o50) atan(b/a)
```

Für bestimmte Werte werden symbolische Ergebnisse gefunden.

```
(%i51) carg(1+%i*sqrt(3));
(%o51) %pi/3
```

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

```
(%i52) %e^(%i*%pi)+1;
(%o52) 0
```

Ein komplexer Ausdruck

```
(%i53) z:(1+%i)/(5-2*%i);
(%o53) (%i+1)/(5-2*%i)
```

Berechnung des Realteils

```
(%i54) realpart(z);
(%o54) 3/29
```

Berechnung des Imaginärteils

```
(%i55) imagpart(z);
(%o55) 7/29
```

Darstellung in kartesischen Koordinaten (Real- und Imaginärteil)

```
(%i56) rectform(z),numer;
(%o56) 0.24138*%i+0.10345
```

Darstellung in Polarkoordinaten (Betrag und Argument)

```
(%i57) polarform(z),numer;
(%o57) 0.26261*%e^(1.1659*%i)
```

1.3.6 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

```
(%i58) sum(1/2^n,n,1,10);
(%o58) 1023/1024
```

Tabelle 1.7 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`).

Summe mit symbolischen Summanden

```
(%i59) sum(x[k],k,1,5);
(%o59) x5+x4+x3+x2+x1
```

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

```
(%i60) sum(x[k],k,1,N);
                 N
                ====
                \
(%o60)           >    x
                /     k
                ====
                k = 1
```

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

```
(%i61) sum(1/n^2,n,1,inf),simpsum=true;
                  2
                 %pi
(%o61)           ---
                  6
```

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

```
(%i62) limit(sin(x)/x,x,0);
(%o62) 1
```

Definition der eulerschen Zahl *e*

```
(%i63) limit((1+1/n)^n,n,inf);
(%o63) %e
```

1.3.7 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

```
(%i64) z:(x-1)/(x+5);
                 x - 1
(%o64)           -----
                 x + 5
```

Erste Ableitung

```
(%i65) diff(z,x);
                  1          x - 1
(%o65)          -----  -  ----------
                x + 5             2
                          (x + 5)
```

Dritte Ableitung

```
(%i66) diff(z,x,3);
                   6          6 (x - 1)
(%o66)          ------  -  ------------
                     3               4
                (x + 5)    (x + 5)
```

Tabelle 1.8 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

Unbestimmtes Integral

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

Bestimmtes Integral zwischen den Grenzen 0 und 10

```
(%i68) integrate(z,x,0,10);
(%o68) -6*log(15)+6*log(5)+10
```

Ausgabe als Gleitkommazahl

```
(%i69) integrate(z,x,0,10),numer;
(%o69) 3.4083
```

Für dieses Integral gibt es keine geschlossene Lösung.

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

$$\text{(\%o70)} \quad \int_0^{\pi} \frac{\sin(x)}{x^2+x+1}\,dx$$

Numerische Integration nach dem Romberg-Verfahren

```
(%i71) romberg(sin(x)/(x**2+x+1),x,0,%pi);
(%o71) 0.50388
```

Numerische Integration mit einer Quadpack-Funktion

```
(%i72) quad_qags(sin(x)/(x**2+x+1),x,0,%pi);
(%o72) [0.50388,1.2231*10^-9,21,0]
```

1.3.8 Einschränken des Bereichs 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.

Tabelle 1.9 Einschränken des Bereichs 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

Bei ungebundenen Symbolen ist die Bestimmung des Maximums nicht möglich.

```
(%i73) max(3,5,a,b);
(%o73) max(5,a,b)
```

Einschränkung des Bereichs für zwei Variablen

```
(%i74) assume(a>10,b<a);
(%o74) [a>10,a>b]
```

Damit wird die Ermittlung des Maximums möglich.

```
(%i75) max(3,5,a,b);
(%o75) a
```

Aufhebung der Bereichseinschränkung für eine Variable

```
(%i76) forget(a>b);
(%o76) [a>b]
```

Die Ermittlung des Maximums ist damit nicht mehr möglich.

```
(%i77) max(3,5,a,b);
(%o77) max(a,b)
```

Integral, dessen Berechnung vom Wertebereich einer Variablen abhängt.

```
(%i78) integrate(sin(t)*exp(-s*t),t,0,inf);
       Is s positive, negative or zero? p;
(%o78)                    1
                        ------
                         2
                        s  + 1
```

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

```
(%i79) assume(s>0);
(%o79) [s>0]
```

... so erfolgt die Berechnung ohne unterbrechende Nachfrage.

```
(%i80) integrate(sin(t)*exp(-s*t),t,0,inf);
(%o80)                    1
                        ------
                         2
                        s  + 1
```

Liste mit allen getroffenen Annahmen über den Bereich von Variablen

```
(%i81) facts();
(%o81) [a>0,b>0,a>10,s>0]
```

1.3.9 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 Näherungslösung 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.

Tabelle 1.10 Gleichungen

`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(`*eqn*`,` *x*`,` x_1`,` x_2`)`	Numerische Lösung der Gelichung *eqn* in der Variablen *x* innerhalb des Intervalls $x_1 \ldots x_2$

Eine quadratische Gleichung liefert zwei Lösungen.

```
(%i82) res1:solve((x+5)/(x^2-a)=2,x);
```

$$(\%o82)\quad \left[x=-\frac{\sqrt{16\,a+41}-1}{4},x=\frac{\sqrt{16\,a+41}+1}{4}\right]$$

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

```
(%i83) ev(x,res1);
```

$$(\%o83)\quad -\frac{\sqrt{16\,a+41}-1}{4}$$

Zwei lineare Gleichungen in zwei Variablen

```
(%i85) g1:2*x+y=3;
       g2:2*x-4*y=5;
```

$$(\%o84)\quad y+2\,x=3$$

$$(\%o85)\quad 2\,x-4\,y=5$$

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

```
(%i86) res2:solve([g1,g2],[x,y]);
```

$$(\%o86)\quad \left[\left[x=\frac{17}{10},y=-\frac{2}{5}\right]\right]$$

Zugriff auf eine Lösungsvariable mit der Anweisung `ev`

```
(%i87) ev(x,res2);
```

$$(\%o87)\quad \frac{17}{10}$$

Quadratische Gleichung in zwei Variablen

```
(%i88) g1:2*x^2+x-y^2=1;
```

$$(\%o88)\quad -y^2+2\,x^2+x=1$$

Auch für nichtlineare Gleichungssysteme können Lösungen gefunden werden.

```
(%i89) res3:solve([g1,g2],[x,y]);
```

$$(\%o89)\quad \left[\left[x=-\frac{4\sqrt{23}+9}{14},y=-\frac{\sqrt{23}+11}{7}\right],\left[x=\frac{4\sqrt{23}-9}{14},y=\frac{\sqrt{23}-11}{7}\right]\right]$$

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

```
(%i90) res4:solve(x^5-x^3+2*x^2+5*x-1=0,x);
(%o90) [0=x^5-x^3+2 x^2+5 x-1]
```

Numerische Berechnung der Nullstellen eines Polynoms

```
(%i91) res5:allroots(x^5-x^3+2*x^2+5*x-1);
(%o91) [x=0.18724,x=0.71869 %i-1.2334,x=-
       0.71869 %i-1.2334,x=1.1496 %i+1.1398,x=1.1398
       -1.1496 %i]
```

Näherungslösung einer transzendenten Gleichung innerhalb eines Intervalls

```
(%i92) res6:find_root(exp(-x)=x,x,0,10);
(%o92) 0.56714
```

1.3.10 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.

Tabelle 1.11 Benutzerdefinierte Funktionen

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

Definition einer Funktion mit dem formalen Parameter *t*

```
(%i93) f(t):=sin(t)*exp(-5*t);
(%o93) f(t):=sin(t) exp((-5) t)
```

Verwendung der Funktion mit unterschiedlichen aktuellen Parametern

```
(%i94) [diff(f(x),x),integrate(f(omega),omega)];
(%o94) [%e^-5 x cos(x)-5 %e^-5 x sin(x),
       %e^-5 omega (-5 sin(omega)-cos(omega))
       ----------------------------------------]
                        26
```

Definition einer Funktion in drei Variablen

```
(%i95) f(x,y,z):=(2*x**2-y)/(z+1)^2;
                          2 x^2-y
(%o95) f(x,y,z):=-------------------
                        (z+1)^2
```

Ermittlung eines Funktionswertes

```
(%i96) f(2,2,3);
(%o96) 3/8
```

Auch das Rechnen mit *symbolischen* Funktionen ist möglich.

```
(%i97) diff(g(x)*sin(x),x);
                                d
(%o97) sin(x) (-------- g(x)) + g(x) cos(x)
                d x
```