Kapitel 14

Computeralgebra mit Maple

14.1 Einführung

Gute Einführungen in Maple sind André Heck, *Introduction to Maple*, Springer Verlag sowie Michael Kofler, Maple V Release 4, Einführung und Leitfaden für den Praktiker, Addison Wesley. Weiterhin gibt es einige brauchbare Kurzeinführungen, die auf dem Internet verfügbar sind. Als Startpunkt der Suche empfiehlt sich `www.maplesoft.com`, ein Rechner, der von den Maple-Entwicklern unterhalten wird.

Maple ist eines der am weitesten entwickelten Computeralgebra-Programme. Es bietet einen weiten Vorrat an Symbolmanipulationsroutinen, numerischen Algorithmen, flexiblen Daten- und Kontrollstrukturen für die Programmierung, grafische Ausgabemöglichkeiten und die Möglichkeit der Protokollierung der Sitzungen in sogenannten Arbeitsblättern *(worksheets).*

Ganz ähnliche Funktionalität wie Maple bieten Mathematica und Macsyma, andere weitverbreitete Computeralgebra-Programme, die nicht auf bestimmte Problemklassen spezialisiert sind. Es sollte noch REDUCE erwähnt werden, das weiterhin gepflegt wird und als LISP Quellcode verfügbar ist. REDUCE ist wie Macsyma ein "Veteran" und stammt bereits aus den 60'er Jahren. Die auf LISP beruhenden Programme haben den Ruf, mit Systemresourcen nicht sehr sparsam umzugehen, was insbesondere bei umfangreichen symbolischen Operationen Probleme machen kann.

Die "modernen" Systeme wie Mathematica und Maple setzen auf einem algorithmischen Kern aus C Routinen auf, in dem die Grundfunktionalität implementiert ist. Diese Systeme sind durch benutzerdefinierte Routinen erweiterbar, die entweder zur Laufzeit interpretiert werden oder in einem kompakten, schnell lesbaren internen Zwischencode vorliegen.

Der Hauptvorteil der Computeralgebra liegt darin, daß bei symbolischer Manipulation keine Information verlorengeht. Falls Arithmetik erforderlich ist, so wird diese mit beliebiger, bzw. bei numerischen Rechnungen mit einer vorher festgelegten (fast) beliebigen Genaugkeit ausgeführt. Der Hauptnachteil besteht in dem gegenüber rein numerischen, hardwareorientierten Ansätzen sehr großen Bedarf an Speicherplatz und Rechenzeit.

Der folgende Text beabsichtigt nur, in die Grundideen von Maple einzuführen, die häufig auch Entsprechungen in den Konzepten der anderen Programme haben, die sich dann ebenfalls leicht erlernen lassen sollten. Komplexere Aufgabenstellungen sollten mit einem der eingangs erwähnten Büchern und der im Programmpaket enthaltenen interaktiven Hilfefunktionalität erschließbar werden.

14.2 Eingabe

Maple arbeitet zeilenorientiert. Es gibt daher sowohl eine kommandoorientierte Version, die sich durch `maple` aufrufen läßt, als auch eine Version mit grafischer Benutzeroberfläche, die unter UNIX als `xmaple` startet. Deren wesentlicher Vorteil ist eine bessere Editierbarkeit der Kommandos und natürlich die viel besseren Möglichkeiten, Grafiken darzustellen. Der rein algorithmische Kern ist bei beiden Versionen gleich. Die aktuelle Maple Version ist V, Release 5. Die folgenden Beispiele sind mit Release 3 und 4 gerechnet worden.

Die Eingabesyntax ist ähnelt denen vieler prozeduraler Programmiersprachen mit den üblichen Operatoren, Assoziativitäts-, Kommutativitäts und Präzedenzregeln.

Im interaktiven Betrieb wertet Maple alle Eingaben hinter dem Prompt `>` aus, sobald sie durch ein Semikolon ; oder einen Doppelpunkt : beendet werden. Der Doppelpunkt unterdrückt dabei die Ausgabe des berechneten Ergebnisses. Maple führt Rechnungen soweit möglich, symbolisch aus, und geht erst dann zu numerischen Rechnungen über, wenn (i) es mit `evalf` (**eval**uate by **f**loating point computation) dazu aufgefordert wird, oder (ii) der zu berechnende Ausdruck eine bereits numerisch ermittelte Zahl enthält, wozu ein Punkt in der Zahldarstellung ausreicht. Kommentare werden durch `#` eingeleitet und enden am Zeilenende.

```
> restart;
>  3 /7 + 5/8;
```

$$\frac{59}{56}$$

```
> # mit  "  bezieht man sich auf das jeweils letzte Ergebnis,
> # zwei "" bezeichnen das vorletzte Ergebnis, usw.
> evalf(");
```

$$1.053571429$$

```
> evalf("",25);
```

$$1.053571428571428571428571$$

14.3 Hilfe, Dokumentation online

Eines der wichtigsten Hilfsmittel zum Umgang mit Maple ist die interaktiv verfügbare Hilfe. Man erreicht sie im Kommandozeilenmodus durch Voranstellen eines Fragezeichens vor die Funktion, für die Hilfe benötigt wird, z.B. `?exp`. Ein Index läßt sich über `?index` erreichen.

In Fenstersystemen gibt es zusätzlich leistungsfähige Suchmechanismen, die in der Regel erlauben, die benötigte Funktionalität in kurzer Zeit aufzufinden (*Topic Search, Full Text Search*).

14.4 Variablen, Zuweisung und Auswertung

Jedes sinnvolle Objekt kann in Maple mit einen Namen verbunden werden Die Repräsentation eines solchen Objektes im Speicher beinhaltet den Typ, Speicherbereich für die interne Darstellung der Zeichenkette sowie einen Zeiger, der auf einen eventuellen "Wert" zeigt. In Abb. (14.1) ist als Beipiel die interne Darstellung eines polynomialen Ausdrucks gezeigt. Ein bisher unbekannter Name in einem Ausdruck dient als Variable in dem Sinne, daß Maple versucht, Operationen so durchzuführen, daß für alle möglichen Einsetzungen korrekte Ergebnisse resultieren.

Dieser Wert kann jetzt ein beliebiger anderer Name, eine Zahl, eine Funktionsvorschrift oder ein anderer Maple-Datentyp sein. Ein Wert wird in einer Zuweisung festgelegt, die entweder durch den Operator `:=` oder funktional durch `assign()` durchgeführt wird. Zur internen Repräsentation wird ein Objekt angelegt, daß der linken Seite entspricht und dann im Namensobjekt ein Zeiger auf diese Struktur eingetragen.

Achtung: Das Symbol `=` alleine trennt linke und rechte Seite einer mathematisch aufzufassenden *Gleichung*, die selbst wieder ein Objekt darstellt, das zugewiesen werden kann. Der auf der linken Seite einer Zuweisung stehende Ausdruck wird dabei von Maple vor der Zuweisung zunächst soweit möglich ausgewertet. Um eine Auswertung zu verhindern, benutzt den linksgerichteten Akzent `'` oder `evaln()`; um eine Auswertung zu erzwingen, stellt Maple `eval()` bereit.

Bei der Auswertung eines Ausdruckes folgt Maple dabei soweit wie möglich den Zeigern, die die verschiedenen Objekte miteinander verknüpfen. Falls sich dabei schleifenförmige

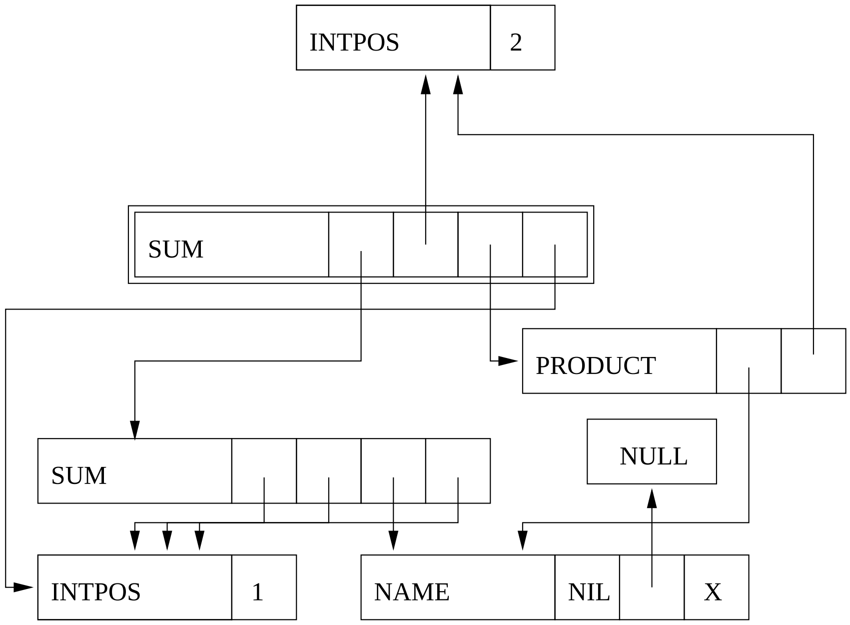


Abbildung 14.1: Interne Repräsentation des Ausdrucks $2(1 + x) + x^2$ durch einen gerichteten, nichtzyklischen Graph (DAG) im Maple Kern. Summen werden aufgelöst in Summanden und die dazugehörigen Koeffizienten, sofern diese Konstanten sind. So steht beispielsweise `SUM ->(1+x) ->2 ->x^2 ->1` (doppelt umrahmt) für eine Summe mit den Bestandteilen $(1 + x)$ und x^2 mit den Koeffizienten 2 und 1. Ein Produkt wird wiederum in Ausdrücke und Wiederholungsfaktoren zerlegt, die jetzt als Exponenten zu interpretieren sind. Nur wenn diese Exponenten keine Konstanten sind (was in der Abbildung nicht der Fall ist), ist es nötig, auf eine Struktur "Potenz" zurückzugreifen.

Abhängigkeiten ergeben, so wird dies in neueren Versionen als Fehler erkannt und führt in älteren Systemen zum Absturz.

```
> restart;
> # Ein Ausdruck wird mit einem Namen belegt
> expression:= sin(x) + 3*x^3 +1/x;
```

$$expression := \sin(x) + 3\,x^3 + \frac{1}{x}$$

```
> # wir ersetzen x durch y (keine Ausgabe) und werten neu aus
> x := y: expression;
```

$$\sin(y) + 3\,y^3 + \frac{1}{y}$$

```
> # subs() erlaubt das (temporaere) Ersetzen, hier von y
> subs( y=Pi, expression ) ;
```

$$\sin(\pi) + 3\,\pi^3 + \frac{1}{\pi}$$

```
> # eval erzwingt die Auswertung des neuen Ausdrucks
> eval( subs(y=Pi, expression ));
```

$$3\,\pi^3 + \frac{1}{\pi}$$

```
> # alternativ haetten wir auch y den Wert Pi zuweisen koennen
> y := Pi; expression;
```

$$y := \pi$$

$$3\,\pi^3 + \frac{1}{\pi}$$

```
> # Beispiel f"ur eine fehlerhafte, rueckbezuegliche Def.
> a := b:  b := 'a':
> # Fehler in revision 3: Stack overflow: pid 26029, proc mapleV, ...
> # Fehler in revision 4: too many levels of recursion
> b;
```

Selbstkontrolle:
Warum müssen wir nach der Zuweisung `x := y` im obigen Beispiel im Ausdruck `expression` die Variable `y` substituieren und nicht `x`?
Wie hätte Maple's Antwort auf die Eingabe `a := b:  b := a: b;` gelautet?
Zu welchem Ausdruck wird `subs( 1=3, x^1 + y );` ausgewertet? Berücksichtigen Sie dabei die interne Darstellung!

Wir haben oben gesehen, daß Maple einige Namen bereits von sich aus belegt: dazu gehören die Namen interner Funktionen und Prozeduren wie `eval()` und `subs()` sowie die Konstanten `Pi` und `E`. Vorsicht ist geboten bei Namen, die griechischen Buchstaben entsprechen; der Name `pi` wird von Maple zwar als π wiedergegeben, aber intern existiert keinerlei Verknüpfung zur Konstanten `Pi`. Die Konstante `gamma` ist die Euler-Mascheroni-Konstante $0.57721\ldots$.

Maple verwendet `I`, um die komplexe Einheit $\sqrt{-1}$ zu bezeichnen. Maple wertet Ausdrücke normalerweise nur dann aus, wenn die nötigen Umformungen auch über den komplexen Zahlen gültig sind. Das Ergebnis wird aber, auch wenn es komplex ist, nicht immer als komplexe Zahl dargestellt, sondern bleibt häufig als Ausdruck stehen. Um eine Zerlegung in Real- und Imaginärteil zu erreichen, benutzt man `evalc`.

```
> restart;
> exp( I * Pi +2 );
```

$$\mathrm{e}^{(I\,\pi+2)}$$

```
> evalc(");
```

$$-\mathrm{e}^2$$

```
> arcsin(2);
```

$$\arcsin(\,2\,)$$

```
> evalc(");
```

$$\frac{1}{2}\pi - I\ln\left(2+\sqrt{3}\right)$$

14.5 Eingebaute Funktionen und Prozeduren

An einem Beispiel sollen der Umfang und die Leistungsfähigkeit der Routinen demonstriert werden, die Maple bereitstellt. Wir werden nicht auf die Syntax aller folgenden Funktionen eingehen; Einzelheiten lassen sich am besten der online Hilfe entnehmen.

Maple kennt die Eigenschaften (Differenzierbarkeit, Wertebereich, Verzweigunsschnitte, spezielle Werte) einer riesigen Menge eingebauter Funktionen, die von den üblichen algebraischen und trigonometrischen Funktionen über unstetige Vertreter wie der Heaviside- oder der δ-“Funktion” bis hin zu Exoten wie Lamberts W-Funktion mit $W(x)*exp(W(x)) = x$ reichen.

Darüberhinaus gibt es viele Funktionen für die Manipulation von Ausdrücken, von denen wir oben bereits `subs` kennengelernt haben. Maple unterscheidet die Typen `procedure` als dem Namen einer Funktion/Prozedur und `function`, den man erhält, wenn ein bekannter oder nicht bekannter Name mit einer Argumentliste in runden Klammern versehen wird.

Hier gehen wir durch eine kurze Kurvendiskussion, um einen kurzen Eindruck von der Breite möglicher Operationen zu gewinnen.

```
> f:= arctan( (2*x^2-1) / (2*x^2+1));
```

$$f := \arctan(\frac{2\,x^2-1}{2\,x^2+1})$$

```
> raw_df := diff(f,x);
```

$$raw\_df := \frac{4\,\frac{x}{2\,x^2+1} - 4\,\frac{(2\,x^2-1)\,x}{(2\,x^2+1)^2}}{1+\frac{(2\,x^2-1)^2}{(2\,x^2+1)^2}}$$

```
> df:=simplify( raw_df );
```

$$df := 4\,\frac{x}{4\,x^4+1}$$

```
> F:= int( df, x);
```

$$F := \arctan(2\,x^2)$$

```
> # Berechnung der Minima und Maxima (allg. unter Nebenbedingungen)
> readlib(extrema):
> extrema( f, {}, x, 's' ), s;
```

$$\{-\frac{1}{4}\pi\}, \{\{x = 0\}\}$$

```
> zeroes:=solve( f=0, x );
```

$$zeroes := \frac{1}{2}\sqrt{2}, -\frac{1}{2}\sqrt{2}$$

```
> # Probe: Einsetzen der Nullstellen, map wendet Funktion
> #  auf Elemente einer Liste an
> map( y->subs(x=y,f), [zeroes] );
```

$$[\arctan(0), \arctan(0)]$$

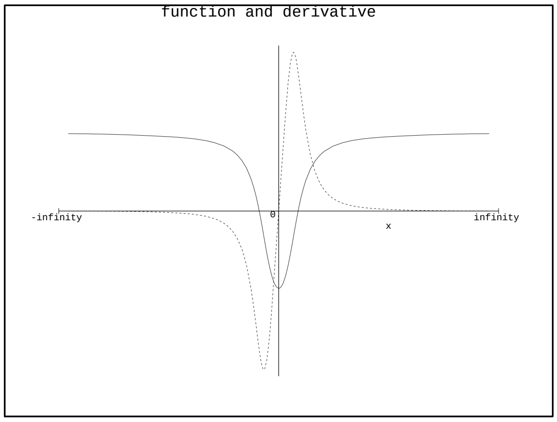
```
> # Taylorreihe um x=0 und infinity
> series( f, x=0, 15);
```

$$-\frac{1}{4}\pi + 2x^2 - \frac{8}{3}x^6 + \frac{32}{5}x^{10} - \frac{128}{7}x^{14} + \mathrm{O}(x^{15})$$

```
> series(f,x=infinity);
```

$$\frac{1}{4}\pi - \frac{1}{2}\frac{1}{x^2} + \mathrm{O}(\frac{1}{x^6})$$

```
> # Liste aus 2 Graphen die dann gemeinsam mit verschiedenen Parametern
> # dargestellt werden
> f_plot := plot(f, x=-infinity..infinity,linestyle=0,thickness=2 ):
> df_plot:= plot(df,x=-infinity..infinity,linestyle=4 ):
> plots[display](\{f_plot, df_plot\},title=`function and derivative`);
```



Ausdrücke, Funktionen und Prozeduren

Wir haben oben Beispiele von Anweisungszeilen der Form

```
> f := x^2 + a*x + 1;
```

$$f := x^2 + a\,x + 1$$

gesehen. Hier wurde `f` ein Ausdruck mit Variablen zugewiesen. Maple zeichnet keine Variable in besonderer Form aus, so daß vielen Operationen noch mitgeteilt werden muß, bezüglich welcher Variablen sie arbeiten sollen, so wir wir das oben am Beispiel der Integration und Differentiation gesehen haben. Ein Problem ist, daß die als Unbekannte benutzte Variable (sagen wir x) nachträglich an einen Wert oder einen anderen Variablennamen gebunden werden kann (vgl. Abschnitt *Zuweisungen*), was interessante Effekte hervorruft:

```
> diff(f,x);
```

$$2\,x + a$$

```
> x:=2: diff(f, x);
Error, wrong number (or type) of parameters in function diff
> x:= a: diff(f,x);
```

$$4a$$

Das liegt letztlich natürlich daran, daß Maple zunächst auswertet und dann erst die gewünschte Operation durchführt. Mit einer “echten” Funktion erhalten wir die Freiheit, den für das Argument verwendeten Parameter frei zu wählen. Maple nutzt dabei zwei äquivalente Schreibweisen, eine mittels eines Pfeiloperators, die andere mittels Schlüsselwort `proc`.

```
> f:= x->x^2;
```

$$f := x \to x^2$$

```
> f(y), f(4);
```

$$y^2,\, 16$$

```
> # ein Fehler, der haeufig auch passiert, bevor f definiert ist
> # Bedeutung: ein spezieller Wert von f wird errinnert
> f(4):=17: f(4);
> }
```

$$17$$

```
> # op erlaubt, Komponenten eines Ausdrucks zu selektieren
> # op( 0, eval(f) ) 0=Funktionsname; ..;  4=remember table
> op( 4 , eval(f) );
```

table([
4 = 17
])

```
> # Zeilenumbruch innerhalb von Prozeduren: SHIFT+RETURN
> # Typueberpuefung von Argumenten optional, auch Liste moeglich
> fibo := proc(n::nonneg)
> Zeilenumbruch SHIFT+ENTER
>   if n > 1 then fibo(n-1) + fibo(n-2)
>   else 1
>   fi;
> end;
```

$$fibo := \mathbf{proc}(n{::}nonneg)\,\mathbf{if}\,1 < n\,\mathbf{then}\,\mathrm{fibo}(n-1) + \mathrm{fibo}(n-2)\,\mathbf{else}\,1\,\mathbf{fi}\,\mathbf{end}$$

```
> fibo(3), time(fibo(20)); op( 4, eval(fibo));
```

$$3, 4.339$$

```
> # option remember:
> #  Funktion erinnert nun automatisch bereits berechnete Werte
> fibo := proc(n::nonneg) option remember;
>   if n > 1 then fibo(n-1) + fibo(n-2)
>   else 1
>   fi;
> end;
```

$$fibo := \mathbf{proc}(n{::}nonneg)\,\mathbf{option}\,remember;\,\mathbf{if}\,1 < n\,\mathbf{then}\,\mathrm{fibo}(n-1) + \mathrm{fibo}(n-2)\,\mathbf{else}\,1\,\mathbf{fi}\,\mathbf{end}$$

```
> fibo(3), time(fibo(3000));
```

$$3, 1.329$$

Die Tiefen von Maple lassen sich durch Studium des Maple Programmcodes der Maple-Prozeduren ergründen. Diese erhält man nach dem Aufruf `interface( verboseproc=3 )` mit `print`*(Prozedurname)*. Unter anderem läßt sich so auch der Differentiationsoperator `D` entweihen …

Wo Maple sonst einen Ausdruck erwartet, kann auch eine Prozedur mit entsprechenden Argumentvariablen stehen. Die Wahl des Namens dieser Variable kann jetzt aber natürlich eine andere sein als bei der Definition der Prozedur. Wie im Beispiel gesehen, muß man darauf achten, *niemals* einen Audruck der Form `f(x) := x^2` zur Funktionsvereinbarung zu verwenden. Wie wir oben gesehen haben, merkt sich Maple dann nur, was f zurückliefern soll, wenn das Argument gerade `'x'` ist.

Mit Hilfe der Funktion `unapply` ist es möglich, Variablen eines Ausdrucks als formale Argumentparameter zu kennzeichen und somit einen Ausdruck in eine echte Funktion umzuwandeln.

```
> expression := x^2 + b *x + c:
> f := unapply(expression, x);
```

$$f := x \to x^2 + bx + c$$

14.6 Verzögerung der Auswertung, Numerik

Eine Möglichkeit, die Auswertung von Ausdrücken zu unterdrücken, haben wir bereits kennengelernt. Selbst wenn der Name `x` auf ein anderes Objekt verweist, kann man sich mit linksgerichteten Akzenten auf `'x'` selbst beziehen, `x` wird nicht ausgewertet.

Eine weitere Möglichkeit, die Auswertung zu verhindern, ist die Verwendung *inerter* Prozeduren. Diese beginnen mit Großbuchstaben, schreiben sich aber sonst wie ihre fleißigeren Geschwister: `Int, Limit, Sum, Diff,` etc. So ist beispielsweise die inerte Funktion `Int` notwendig, um eine numerische *Integration* der Originalfunktion zu bewirken. Ansonsten versucht Maple das Integral zunächst analytisch auszuwerten und erst dann den entstehenden Ausdruck numerisch zu berechnen. Im folgenden Beispiel ist eine Funktion gezeigt, deren uneigentliches Integral Maple in Release 4 falsch zu 0 auswertet. Man muß daher die analytische Auswertung unterdrücken, um noch ein (richtiges) numerisches Resultat zu bekommen.

```
> # diese Funktion ist > 0 fuer alle x, daher ist das
> # Integral ueber R auch groesser als 0
> f := x-> 1/(x^4+x+1);
```

$$f := x \to \frac{1}{x^4 + x + 1}$$

```
> # maple ist leider anderer Meinung
> int(f(x), x=-infinity..infinity);
```

$$0$$

```
> # daher wollen wir die Integration numerisch durchfuehren
> Int(f(x), x=-infinity..infinity);
```

$$\int_{-\infty}^{\infty} \frac{1}{x^4 + x + 1}\, dx$$

```
> # mit 30 Stellen Genauigkeit
> evalf ( ", 30 );
```

$$2.68354824425647647957447074409$$

14.7 Objektattribute

Maple scheint bei bestimmten einfachen Umformungen zu versagen. So weigert es sich etwa, `sqrt(a*a)` zu `a` zu vereinfachen. Nun ist diese Vereinfachung tatsächlich nur dann gültig, wenn a eine positive reelle Zahl ist. In allen anderen Fällen muß die Mehrdeutigkeit der Wurzelfunktion beachtet werden. Dieses Problem tritt ganz allgemein auf bei analytisch fortsetzbaren Funktionen, die ggfs. mehrdeutige Wertebereiche aufweisen. Solche Funktionen führen Maples Integrationsroutinen häufig bei komplexen Kurventintegralen in die Irre.

Als eine teilweise Lösung des Problems erlaubt Maple es daher, mit Hilfe der `assume` Prozedur eine Variable mit einer Eigenschaft *(property)* auszustatten. Dabei sind bestimmte Schlüsselworte wie `real, positive, negative, nonneg, integer, imaginary` aber auch Intervallangaben zulässig: z.B. sind `assume( a, nonneg )` und `assume( a>= 0)` gleichbedeutend.

```
> sqrt(a*a);
```

$$\sqrt{a^2}$$

```
> # Eine mit speziellen Eigenschaften versehene Zahl ist durch
> # ein nachgestelltes ~ gekennzeichnet
> assume( a>= 0); sqrt(a^2);
```

$$a\tilde{}$$

```
> # Manchmal ist es eine gute Idee, Maple nach seinen Vorurteilen
> # zu fragen:
> about (a);
Originally a, renamed a\symbol{126}:
  is assumed to be: RealRange(0,infinity)
```

Maple ist in der Lage, mit diesen Eigenschaften zu "rechnen." Schränkt man etwa `a` darauf ein, eine reelle Zahl zu sein, dann kann Maple deduzieren, daß dann `exp(a)` eine positive reelle Zahl ist. Damit kann es den Ausdruck `sqrt( (exp(a))^2 )` zu `exp(a)` vereinfachen.

14.8 Weitere Datenstrukturen

Maple gibt in Fällen, in denen mehrere Antworten richtig sind, häufig als Ergebnis eine *Folge* zurück. Dies ist eine durch Kommata getrennte Aufzählung von Ausdrücken, die weder in eckigen noch geschwungenen Klammern steht. Z.B. bekommen wir eine Folge, wenn wir nach den Nullstellen eines Polynoms fragen. Folgen können auch explizit generiert werden. Z.B. erzeugt `seq(i^2, i=0..9)` die Sequenz der ersten 10 Quadratzahlen $0, 1, 4 \ldots 81$.

Listen

Im unteren Beispiel berechen wir mit `allvalues` eine Sequenz von Nullstellen einer Funktion, die wir mit eckigen Klammern `[]` umgeben, wodurch eine geordnete Liste entsteht, die wir dann mit `map` weiterverarbeiten. `map` entnimmt jedes Element aus der Liste, und formt mit den Ergebnissen der Anwendung der als erstes Argument spezifizierten Funktion eine neue Liste, unter Beibehaltung der Reihenfolge.

```
> # Maple scheint zu dumm, um Nullstellen zu ermitteln
> f:= x^4-x-1: solveI f=0, x);
```

$$\mathrm{RootOf}(\_Z^4 - \_Z - 1)$$

```
> # wir koennen aber noch hartnaeckiger nachfragen,
> # gleichzeitig verpacken wir die Nullstellen in eine Liste []
> zeroes := [ evalf( allvalues( ") , 20) ];
```

$$\begin{aligned} \mathit{zeroes} := [&-.72449195900051561159,\\ &-.24812606280262193189 - 1.0339820609759677567\,I,\\ &-.24812606280262193189 + 1.0339820609759677567\,I,\\ &1.2207440846057594754] \end{aligned}$$

```
> # Zur Probe Einsetzen in f, Berechnen der Funktionswerte
> map( y-> evalf( subs( x=y, f) ) , zeroes );
```

$$[0, 0, 0, 0]$$

Listen können beliebig ineinander verschachtelt werden und beliebige Datentypen als Elemente enthalten. Maple-intern sind Listen lineare Felder von Zeigern auf ihre Elemente. Die Anzahl der Elemente wird durch `nops` ermittelt, der direkte Zugriff auf Elemente erfolgt durch einen in eckigen Klammern angegebenen Index. Das erste Element der Liste trägt im Gegensatz zu C aber in Übereinstimmung mit z.B. FORTRAN den Index 1.

Mengen

Mengen (`sets`) werden in Maple durch geschweifte Klammern angedeutet. Sie unterscheiden sich in ihrem Verhalten von Listen dadurch, daß Maple automatisch gleiche Elemente eliminiert und gleichzeitig die Erhaltung der Reihenfolge nicht garantiert ist. Mengen werden häufig als Eingabeparameter von Prozeduren verwendet, wenn mehrere Variablennamen oder Gleichungen übergeben werden müssen, bei denen es ja auf die Reihenfolge nicht ankommt. Eine solche Anwendung haben wir bereits am Ende des Beispieles in Abschnitt 14.5 gesehen, als wir eine Grafik mit mehreren Kurven generiert haben.

Tabellen und Felder

Tabellen sind Zuordnungsvorschriften, die einem Element eines Schlüsseldatentyps einen "Wert" zuordnen. Wir haben Tabellen bei der Diskussion der `remember` Option von Prozeduren kennengelernt. Intern sind Tabellen als *Hash tables* organisiert, d.h. daß die Position des gesuchten Elementes i.w. durch eine Indexfunktion ermittelt wird und so ein sehr schneller Zugriff unabhängig von der Elementposition erfolgen kann.

```
> table( [seq(i=2*i,i=0..1) ] );
```

$$\begin{array}{rcl} \text{table} & ([& \\ 0 & = & 0 \\ 1 & = & 2 \\ &]) & \end{array}$$

```
> a[1];
```

$$2$$

Felder verhalten sich wie mehrdimensionale Tabellen mit speziellen Vereinbarungen für die zulässigen Indizes, die frei wählbare, aber lückenlose Bereiche von ganzen Zahlen sein müssen. Nicht alle Feldelemente müssen definiert werden. Das Feld kann zusätzliche Eigenschaften aufweisen, wie `sparse`, `tridiagonal`, `identity`, `diagonal`, `symmetric`, `antisymmetric`, die Maple erlauben, die Werte speichereffizient abzulegen. Diese Eigenschaften sind letztlich Indizierungsfunktionen, die man als Benutzer auch selbst definieren kann.

```
> a:= array( symmetric, 0..1, 0..1, [ (0,0) = 7, (0,1) = exp(3) ] );
```

$$\begin{array}{l} a := \mathrm{array}(\mathit{symmetric}, 0..1, 0..1, [\\ \quad (0, 0) = 7 \\ \quad (0, 1) = e^3 \\ \quad (1, 0) = e^3 \\ \quad (1, 1) = a_{1,1} \\ \quad]) \end{array}$$

```
> a[1,0];
```

$$e^3$$

14.9 Lineare Algebra

Der Einsatz von Feldern liegt insbesondere in der linearen Algebra, in der mit Matrizen umgegangen werden muß. Matrizen sind spezielle Felder, deren Indexbereiche immer mit 1 beginnen. Das Paket zu Aufgabenstellungen aus der linearen Algebra stellt die Funktion `matrix` zur Verfügung, mit deren Hilfe sich Matrizen leicht erzeugen lassen, wenn man eine Funktion angeben kann, die die Einträge berechnet. Ansonsten kann die Angabe der Werte auch über geschachtelte Listen erfolgen.

Die Auswertung von Ausdrücken die Matrizenaddition oder -multiplikation verwenden, geschieht mit `evalm`. Bei der Matrizenmultiplikation muß beachtet werden, daß das Symbol `*` in Maple für eine *kommutative* Multiplikationsoperation reserviert ist. Um die Nichtkommutativität von Matrizenmultiplikationen anzudeuten, wird der Operator `&*` verwendet.

Die Vielfalt der möglichen Operationen sieht man der Ausgabe von Maple beim Laden der Bibliothek durch die Anweisung `with(linalg);` an.

```
> restart:
> a:= array( symmetric, 0..1, 0..1, [ (0,0)=7,
> (0,1) = exp(5) ] );
```

$$a := \mathrm{array}(symmetric, 0..1, 0..1, [$$
$$(0, 0) = 7$$
$$(0, 1) = e^5$$
$$(1, 0) = e^5$$
$$(1, 1) = a_{1,1}$$
$$])$$

```
> with ( linalg );
Warning, new definition for norm
Warning, new definition for trace
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, *QRdecomp*, *Wronskian*, *addcol*, *addrow*, *adj*, *adjoint*, *angle*, *augment*, *backsub*, *band*, *basis*, *bezout*, *blockmatrix*, *charmat*, *charpoly*, *cholesky*, *col*, *coldim*, *colspace*, *colspan*, *companion*, *concat*, *cond*, *copyinto*, *crossprod*, *curl*, *definite*, *delcols*, *delrows*, *det*, *diag*, *diverge*, *dotprod*, *eigenvals*, *eigenvalues*, *eigenvectors*, *eigenvects*, *entermatrix*, *equal*, *exponential*, *extend*, *ffgausselim*, *fibonacci*, *forwardsub*, *frobenius*, *gausselim*, *gaussjord*, *geneqns*, *genmatrix*, *grad*, *hadamard*, *hermite*, *hessian*, *hilbert*, *htranspose*, *ihermite*, *indexfunc*, *innerprod*, *intbasis*, *inverse*, *ismith*, *issimilar*, *iszero*, *jacobian*, *jordan*, *kernel*, *laplacian*, *leastsqrs*, *linsolve*, *matadd*, *matrix*, *minor*, *minpoly*, *mulcol*, *mulrow*, *multiply*, *norm*, *normalize*, *nullspace*, *orthog*, *permanent*, *pivot*, *potential*, *randmatrix*, *randvector*, *rank*, *ratform*, *row*, *rowdim*, *rowspace*, *rowspan*, *rref*, *scalarmul*, *singularvals*, *smith*, *stack*, *submatrix*, *subvector*, *sumbasis*, *swapcol*, *swaprow*, *sylvester*, *toeplitz*, *trace*, *transpose*, *vandermonde*, *vecpotent*, *vectdim*, *vector*, *wronskian*]

```
> # matrix und eine anonyme Funktion zur Erzeugung von b
> b:=matrix( 2, 2, (i,j) -> 2*i +j ):
> # Maple merkt, dass a nicht den passenden Indexbereich besitzt
> evalm( a &* b );
```

$$a \,\&{*} \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
> # wir wandeln a also mittels Listenschreibweise in eine Matrix um
> c := matrix( 2,2, [ seq( seq( a[i,j],i=0..1),j=0..1) ] );
```

$$c := \begin{bmatrix} 7 & e^5 \\ e^5 & a_{1,1} \end{bmatrix}$$

```
> # das Ergebnis von evalm ist eine Matrix
> mproduct:= evalm ( c &* b );
```

$$mproduct := \begin{bmatrix} 21 + 5\,e^5 & 28 + 6\,e^5 \\ 3\,e^5 + 5\,a_{1,1} & 4\,e^5 + 6\,a_{1,1} \end{bmatrix}$$

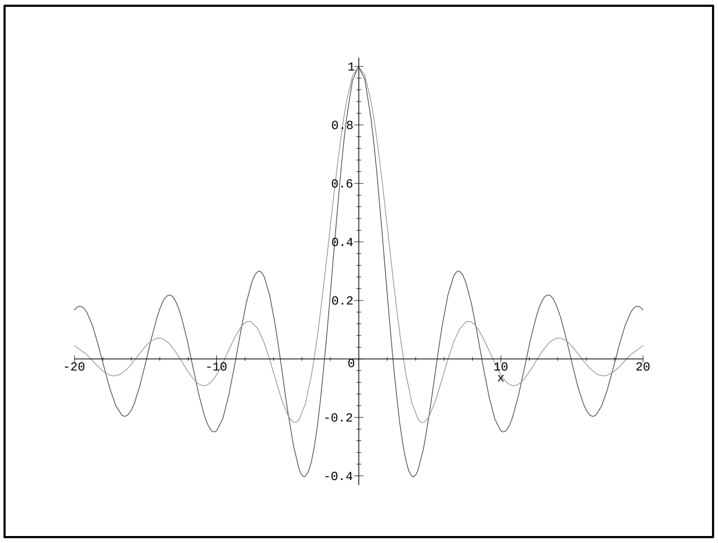
```
> # die wir als solche weiterverarbeiten k\"onnen
> eigenvals( mproduct );
```

$$\frac{21}{2} + \frac{9}{2}e^5 + 3\,a_{1,1} + \frac{1}{2}\sqrt{441 + 378\,e^5 + 308\,a_{1,1} + 73\,(e^5)^2 + 108\,e^5\,a_{1,1} + 36\,{a_{1,1}}^2},$$
$$\frac{21}{2} + \frac{9}{2}e^5 + 3\,a_{1,1} - \frac{1}{2}\sqrt{441 + 378\,e^5 + 308\,a_{1,1} + 73\,(e^5)^2 + 108\,e^5\,a_{1,1} + 36\,{a_{1,1}}^2}$$

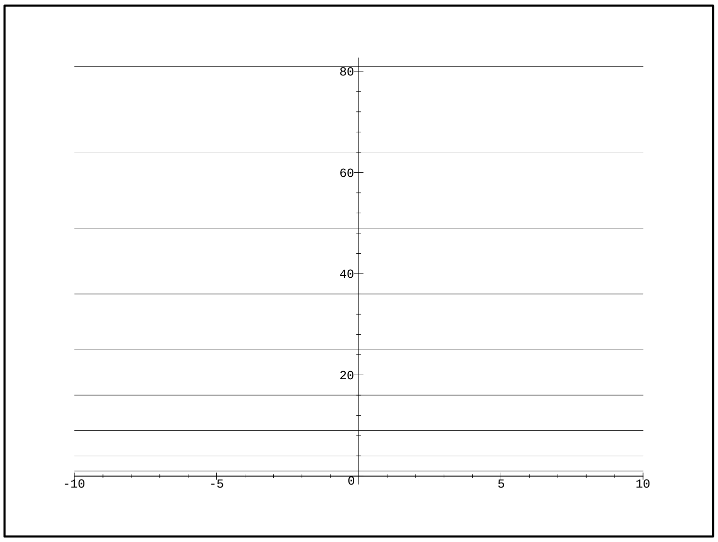
14.10 Grafik

Ein Beispiel für Grafik mit Maple haben wir bereits in Abschnitt 14.5 gesehen, als wir einen Funktionsverlauf grafisch dargestellt haben. `plot` ist in der Lage zweidimensionale Graphen, die entweder in expliziter Form (y-Wert als Formel oder Funktion, x variiert) oder in parametrischer Form (x und y als Funktionen einer weiteren Variable, des Kurvenparameters) vorgegeben sind. Über optionale Parameter wie `color, style, thickness` läßt sich die Ausgabe beeinflussen. Im Beispiel zeigen wir den Vergleich einer Bessel-Funktion mit einem abklingenden Sinus, sowie die Manipulationen, die erforderlich sind, um eine Punktwolke darzustellen.

```
> restart:
> plot( \{BesselJ(0,x), sin(x)/x\}, x=-20..20);
```



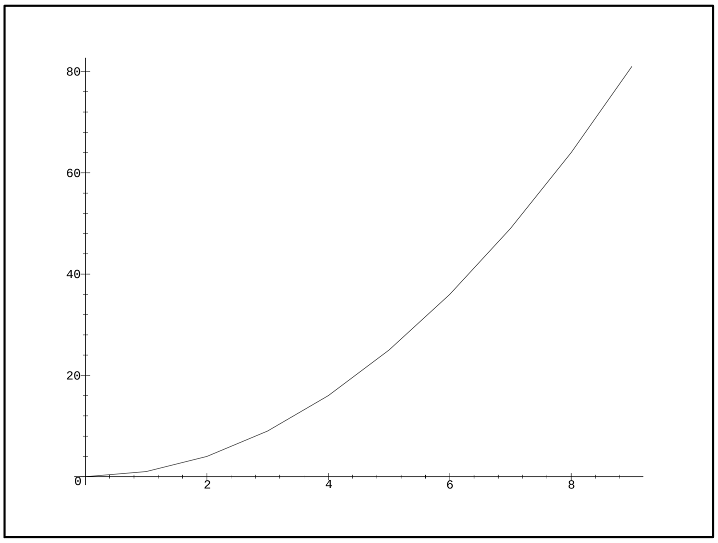

```
> # eine Liste von einzelnen Datenpunktenl, die plot leider nur
> # zu parallelen Linien ueberredet
> lst:= [seq( i^2, i=0..9 ) ]: plot ( lst );
```




```
> # um Punktmengen darzustellen, muessen wir
> # (x,y) Paare erzeugen
> pairs := \{ seq( [ i, lst[i+1] ], i=0..9 )\};
```

$$pairs := \{[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]\}$$

```
> plot ( pairs );
```



Im letzten Kommando hätten wir mit dem optionalen Argument `style=point` Einzelpunkte ausgeben können. Auch dreidimensionale Darstellungen sind in grosser Vielfalt möglich. Wir verweisen dazu auf Maple's online Hilfefunktion.

14.11 Ein- und Ausgabe, Zusammenarbeit mit anderen Programmen

Neben der Möglichkeit, die Arbeitsblätter extern abzulegen und bei Bedarf wieder zu laden, kann Maple auch Daten aus externen Daten in interne Datenstrukturen einlesen und dann bearbeiten. Für geeignet formatierte Daten gibt es dazu die Funktion `readdata`, die als erstes Argument den Namen einer Datei, als zweites die Anzahl der zu lesenden Datenspalten erwartet. Die Funktion erzeugt eine Liste von Listen, die sich zur Anzeige mit `plot` eignet.

Nehmen wir an, die Datei `data` enthalte den folgenden Inhalt,

```
1 2   3
2 5   7
3 5.2 9
```

dann erzeugt `readdata( ‘data‘, 3 )` die Liste

```
> lst := readdata( ‘data‘, 3 );
```

$$lst := [[1., 2., 3.], [2., 5., 7.], [3., 5.2, 9.]]$$

Diese kann mit

```
> writedata ( `data1`, lst );
```

wieder in eine Datei `data1` zurückgeschrieben werden. Man beachte, daß Maple Zeichenketten wie Dateinamen durch die Benutzung von Rechtsakzenten ` identifiziert. Maple sucht dann nicht nach Variablen, die es auswerten könnte.

Um die manchmal komplexen Ausdrücke, die als Ergebnis einer Sitzung mit Maple entstehen können auch in externen numerischen Rechnungen nutzen zu können, gibt es Routinen, die Ausdrücke in syntaktisch korrekte Formen externer Programmiersprachen bringen können, hier am Beispiel einer im Hinblick auf numerische Auswertungen optimierten Lösung eines Polynoms dritten Grades:

```
> restart:
> polyeqn := x^3 - a*x =1;
```

$$polyeqn := x^3 - a\,x = 1$$

```
> sols := solve ( polyeqn, x );
```

$$sols := \frac{1}{6}\,\%1^{1/3} + 2\,\frac{a}{\%1^{1/3}},\ -\frac{1}{12}\,\%1^{1/3} - \frac{a}{\%1^{1/3}} + \frac{1}{2}\,I\,\sqrt{3}\,(\frac{1}{6}\,\%1^{1/3} - 2\,\frac{a}{\%1^{1/3}}),$$
$$-\frac{1}{12}\,\%1^{1/3} - \frac{a}{\%1^{1/3}} - \frac{1}{2}\,I\,\sqrt{3}\,(\frac{1}{6}\,\%1^{1/3} - 2\,\frac{a}{\%1^{1/3}})$$
$$\%1 := 108 + 12\,\sqrt{-12\,a^3 + 81}$$

```
> # erste Loesung nehmen
> sol1 := sols[1]:
> # Funktion zur Umwandlung in C nachladen
> readlib( C);
> # und Programmcode erzeugen
> C( sol1, `optimized`);
```

proc() ... **end**

```
     t1 = a*a;
     t4 = sqrt(-12.0*t1*a+81.0);
     t6 = pow(108.0+12.0*t4,1.0/3.0);
     t9 = t6/6+2.0*a/t6;
> #   bzw gleich in eine Datei schrieben
> C( sol1, `optimized`, filename = `file.c`);
```

14.12 Weiteres

Damit haben wir unseren Schnelldurchgang durch Maple beendet und haben eine grosse Menge von Funktionalität noch nicht besprochen. Hierzu gehören viele zahlentheoretische Funktionen, die Interna der Integrations- und Differntiationsroutinen, Auswertung von Reihen und Grenzwertbildung, Maples verschiedene Möglichkeiten, Ausdrücke zu vereinfachen, zu faktorisieren, nach Vorfaktoren zu ordnen, die Löser für gewöhnliche und partielle Differentialgleichungen, sowie die Möglichkeiten der linearen Algebra und nicht zuletzt auch der 3D Grafik.

Vieles läßt sich durch die bereits mehrfach empfohlene online Hilfe finden, für weitere Interna müssen wir auf die Literatur verweisen.