Tabelle 22.9: Optionen für den Export von Grafikdaten

Option	Default	Bedeutung
ImageSize	288	Bildbreite in 1/72 Zoll
ImageRotated	False	Orientierung des Bildes
ImageResolution	Automatic	Auflösung des Bildes in Punkten/Zoll

22.7 Interna und Ergänzungen

Die internen Abläufe von Mathematica sind in [Math] gut dokumentiert. Dies ermöglich die Programmierung von nichtstandardisierten Berechnungen. Beispiele dafür sind er Umgang mit Differenzialformen und nichtkommutativen Produkten sowie die eingebauten Fähigkeiten zur Mustererkennung.

Zur Mathematica-Lizenz gehört ferner das ebenfalls gut dokumentierte Application Programming Interface (API) **mathlink**, welches in externe Programme und Anwendungen eingebunden werden kann. mathlink ermöglicht die (netzbasierte) Kommunikation beliebiger Anwendungen mit einem an anderer Stelle laufenden Mathematica Kernel.

Für die neueste Version Mathematica 5 ist das Paket **gridMathematica** erhältlich, mit dem die Fähigkeiten von Mathematica auf Hypercomputersystemen (Clustern) und Grids implementiert weden können [WR]. Der Performance von Mathematica sind damit kaum noch Grenzen gesetzt.

Bibliografie und Webliografie

[Math] The Mathematica Book

[TBMM] Henning, P.A.: *Taschenbuch Multimedia*, 3. Auflage (Fachbuchverlag Leipzig 2003)

[WR] Wolfram Research http://www.wolfram.com

[mathview] Externer Viewer mathview http://www.fbi-medialab.fh-karlsruhe.de/mathview.html

First published in: Taschenbuch Programmiersprachen. München, 2004. ISBN 3-446-22580-3

23 Maple

Britta Nestler

Maple ist ein Computeralgebrasystem, dessen Entwicklung im Jahr 1980 von Professoren und Wissenschaftlern an der Universität Waterloo in Canada begonnen wurde. Ziel war es, vielen Benutzern gleichzeitig den Zugang zu dem System zu ermöglichen, eine klare logische, verständliche Syntax zu verwenden und eine einfache Erweiterung für zusätzliche Anwendungen zu ermöglichen. Maple konzentriert sich auf die Bearbeitung konkreter mathematischer Problemstellungen wie z. B. das Lösen von Gleichungen, zwei- und dreidimensionale grafische Darstellungen von Funktionen, Nullstellenbestimmungen, Ableitungen von Funktionen, Finden von Stammfunktionen, Rechnen mit komplexen Zahlen, Integraltransformationen, Lösen von Differenzialgleichungen, Vektorrechnungen usw.

Die einzigen Voraussetzungen für die Nutzung von Maple sind grundlegende Erfahrungen und der Umgang mit Windows-Programmen. Der Kernel von Maple ist in der Programmiersprache C geschrieben und nach der Übersetzung in der Regel kleiner als ein Megabyte.

23.1 Grundlagen

Je nach System (Windows, DOS, Unix etc.) wird Maple mit `xmaple&`, `maple` oder durch Anklicken des Ahornsymbols aufgerufen.

Nach dem Einladen des Maple-Kernels wird der Benutzer durch die Anzeige von `Prompt` zur Eingabe von Befehlen aufgefordert.

Beim Starten einer Sitzung steht ein Teil der Befehle und Funktionen sofort zur Verfügung. Zahlreiche andere muss der Benutzer selbst aufrufen und einladen.

Ein eingegebener Befehl endet mit einem Semikolon ; oder mit einem Doppelpunkt : in den Fällen, für die der Rechenbefehl zwar ausgeführt, das Ergebnis jedoch nicht auf dem Bildschirm angezeigt werden soll. Dadurch können mehrere Befehle nacheinander in einer Zeile aufgereiht oder für einen längeren Befehl mehrere Zeilen benutzt werden.

Die Ausführung von Berechnungen wird durch Drücken der Tasten `Return` oder `Enter` gestartet.

Die Eingabe besteht aus einem Funktionsnamen und, dahinter in runde Klammern gesetzt, dessen Argumente wie z.B. für die Quadratwurzel

`sqrt(676)-1;` oder für die Ableitung einer Sinusfunktion nach der Variablen x `diff(sin(x),x);`.

Maple unterscheidet zwischen Groß- und Kleinschreibung. Die meisten der Funktionsnamen beginnen jedoch mit einem Kleinbuchstaben wie z.B. die grundlegenden Befehle `op`, `type` und `evalf` und mathematische Befehle wie `diff` und `max`. Leerzeichen innerhalb von Befehlen werden ignoriert, so dass deren Einfügen sich oft sehr gut zur Erhöhung der Lesbarkeit eignet.

Beachten sollte man beim Eintippen von Befehlen die Klein- und Großschreibung, die Arten der benutzten Klammern, ob eckig [ ], rund () oder Mengenklammern { }, und die Reihenfolge der verwendeten Satzzeichen wie Ausführungszeichen *, Komma, Semikolon ; und Doppelpunkt : .

Durch Einfügen des Prozentzeichens % kann das letzte Ergebnis direkt als Eingabe für den nachfolgenden Rechenbefehl weiterverwendet werden.

Das vorletzte Ergebnis kann mit zwei hintereinander gesetzten Prozentzeichen %% und das vorvorletzte mit %%% als Eingabe direkt wiederverwendet werden.

Durch Betätigen des Maple eigenen Stop-Icons kann eine lang andauernde Berechnung unterbrochen werden. Eine Maple-Sitzung kann mit den Befehlen: `quit`, `stop` oder `done` ohne abschließendes Semikolon beendet werden.

23.1.1 Online-Hilfe

Durch die Eingabe eines Fragezeichens `?`, gefolgt von einem Befehlsnamen (ohne Semikolon-Endung!) wird zu diesem Befehl eine Hilfsinformation aus dem Maple-Browser online angefordert.

Diese Hilfsinformation enthält eine ausführliche Beschreibung des erfragten Befehls, ein Schema, Querverweise zu verwandten Befehlen und andere Hilfsinformationen des Schlagwortregisters. Zusätzlich können Hilfen mit den Befehlen: `?name`, `?name, unterbegriff`, `?index`, `?pkg[name]`, `?library`, `?package`, `?datatypes`, `?index,procedure`, `?index,tables`, `?expressions`, `??abs` aufgerufen werden. Eine Sammlung verwandter Funktionen, die in der Maple-Bibliothek abgespeichert sind und mit dem `?package-` oder `with`-Befehl in eine aktuelle Sitzung

eingelesen werden, enthalten zahlreiche Operationen und Algorithmen in linearer Algebra, Kombinatorik Zahlentheorie, Statistik, Grafik usw. Sehr umfangreiche Pakete sind zur besseren Handhabung in Unterpakete `subpackages` aufgeteilt und enthalten dann kleinere Gruppen sachlich zueinander gehörender Funktionsgruppen.

23.2 Einfache Berechnungen

Maple verwendet die übliche Schreibweise für die Symbole und Programmiersprachen-Operatoren in mathematischen Ausdrücken.

Standardoperatoren sind: ! Fakultät, ^Potenzbildung, + Addition, - Subtraktion, * Multiplikation, / Division, < kleiner als, <= kleiner gleich, > größer als, >= größer gleich, = gleich, <> ungleich, := Zuweisung.

Ausdrücke werden vereinfacht bzw. expandiert durch

```
simplify(ausdruck);
expand(ausdruck);
```

Ausdrücke können auch lokal ausgewertet werden. Beispiel:

```
subs(x=x0,a);
```

Tabelle 23.1: Befehle für einfache mathematische Funktionen

Ausdruck	Befehl	Bedeutung
$\sum_{k=1}^{n} a_k$	`sum(a(k), k=1..n);`	Summe
$\prod_{k=1}^{n} a_k$	`product(a(k), k=1...n);`	Produkt
|x|	`abs(x);`	Absolutbetrag
$\sqrt{x}$	`sqrt(x);`	Quadratwurzel
ex	`exp(x); E^x;`	Exponentialfunktion
log x, ln x	`log(x); ln(x);`	Natürlicher Logarithmus
sin x, cos x, tan x	`sin(x); cos(x); tan(x);`	Trigonometrische Funktionen
Γ(z)	`GAMMA(z);`	Gammafunktion
$J_v(x)$	`BesselJ(v, x);`	Besselfunktion
$2/\sqrt{\pi}\int_0^x e^{-t^2}dt$	`erf(x);`	Fehlerfunktion
$\zeta(s)$	`Zeta(s);`	Riemannsche Zetafunktion

Ausdruck	Befehl	Bedeutung
c1<+,-,*,/,^>c2	`evalc`	komplexe Rechenoperation

Maple folgt den Standardregeln für Priorität und Assoziativität von Operatoren, d.h. Potenzen werden zuerst, dann Multiplikationen und Divisionen und zuletzt Additionen und Subtraktionen ausgeführt. Negationen stehen auch bei negativen Exponenten in runden Klammern.

Die gebräuchlichsten Zahlentypen bei Maple sind: `integer` (ganze Zahl), `fraction` (Bruch ganzer Zahlen), `complex` (komplexe Zahl), `float` (Fließkommazahl).

Rechengenauigkeit auf `n` Stellen liefert der Befehl `Digits := n` (Default: n=10).

Einige eingebaute symbolische Konstanten sind:

`Pi`: 3.14159254...

`Catalan`: Catalansche Konstante = 0.915966...

`E`: Eulersche Zahl = 2.718281828...

`gamma`: Eulersche Konstante = 0.577216...

23.3 Variablen und Gleichungen

Mit dem Zuweisungsoperator := wird ein Ausdruck einer gültigen **Variablen** zugewiesen

```
name := expression;
```

Gültige Variablennamen sind einfache Zeichenketten, die mit einem Buchstaben beginnen. Durch Einbetten eines Ausdrucks in Anführungszeichen (`' '`) wird seine Auswertung verhindert und nur eine triviale Vereinfachung zugelassen.

Eine Variable kann durch erneute Zuweisung ihres eigenen Variablennamens und durch das Setzen von Anführungszeichen auf der rechten Seite der Gleichung `>a:='a';` wieder freigegeben werden. Die Funktion `unassign` wird aus der Programmbibliothek eingeladen und gibt mehrere in Anführungszeichen gestellte Variablen auf einmal von ihren zugewiesenen Werten frei.

Der Befehl `solve(eq, var)`, `solve(uneq, var)` liefert - falls möglich - die exakte Lösung für lineare **Gleichungssysteme** oder Ungleichungen. Ist eine exakte Lösung nicht explizit angegeben, so ist

mit dem Befehl `evalf(%)` oder alternativ mit `fsolve(eq, var)` das näherungsweise Lösen einer Gleichung möglich.

Der Befehl `assign` weist das Lösungsergebnis eines Gleichungssystems Variablen zu.

Die Eingabe des Befehls `isolve` garantiert eine ganzzahlige Lösung für ein Gleichungssystem. Das allgemeine Iterationsverfahren, die Regula falsi, die Jacobi-Methode und das Newton-Verfahren benutzen zum Lösen von linearen und nichtlinearen Gleichungen die Befehle `fsolve`, `unapply`, `for`-Schleifen und für eine Näherungslösung zusätzlich `while`-Schleifen.

23.4 Grafische Darstellung von Funktionen

Maple ist für die Erstellung von Kurven, Flächen und Diagrammen sehr leistungsfähig. Zur Darstellung von Grafen in zwei und drei Dimensionen sind die Anweisungen `plot`, `smartplot` oder zahlreiche andere Befehle aus dem Maple `plots`-Package wie z.B. `logplot` oder `semilogplot` vorgesehen.

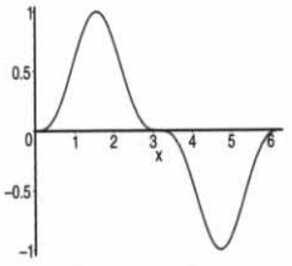

Bild 23.1: Plot von f(x) = (sin(x))^3 im Intervall [0,2 π]

Das `plots`-Package wird mit `with(plots)` aktiviert. Bei dreidimensionalen Grafiken können Drehungen, Orientierung und Projektion der Fläche, Beleuchtung und Schattierung nachträglich verändert werden.

Beispiel (siehe Bild 23.1)

```
> plot(sin(x)^3, x = 0 ..2*Pi);
```

Die Darstellung von Funktionen in einer Variablen erfolgt mit dem Befehl `plot(f(x), x = a..b, opt)` und den Argumenten `f(x)` als Funktionsausdruck, mit `x = a..b` für den Bereich der Variablen und `opt` als optionaler Parameter.

Funktionen `f(x,y)` und Rotationskörper `f(x,y,t)` werden durch den `plot3d`-Befehl und die Bereichsangaben `x = a..b` und `y =`

`c..d` dargestellt. Die Visualisierung erfolgt in Form einer Animation mit `animate` bzw. `animate3d`.

Unter `?plot3d[options]` und im `plots-Package` sind alle weiteren Optionen des `plot3d`-Befehls aufgelistet.

Tabelle 23.2 Optionen zur grafischen Darstellung in mehreren Variablen

Option	Beschreibung
`grid = [n,m]`	Dimension des Gitters n * m
`title = t`	Titel des Schaubildes
`labels = [x,y,z]`	Spezifikation der Achsen
`tickmarks = [l,m,n]`	Anzahl der Achsenmarkierungen
`contours = n`	Anzahl der Höhenlinien
`style = contour`	Einzeichnung von nur Höhenlinien
`style = patchnogrid`	Unterdrückung der Gitterlinien
`view = zmin..zmax`	Der darzustellende z-Bereich
`axes = boxed`	Zeichnung der Achsen
`thickness=<0,1,2,3,..>`	Linienstärke
`orientation=[phi,theta]`	Drehung der 3D-Grafik

Beispiel: Grafische Darstellung der Funktion

$$f(x,y) = \sin\left(\sqrt{x^2+y^2}\right) \Big/ \sqrt{x^2+y^2}$$

```
>f(x,y) := sin(sqrt(x^2 + y^2))/sqrt(x^2 + y^2);
>plot3d(f(x,y), x=-10..10, y=-10..10, style=patch);
>plot3d(f(x,y), x=-10..10,
        y= 10..10,style=patchcontour);
```

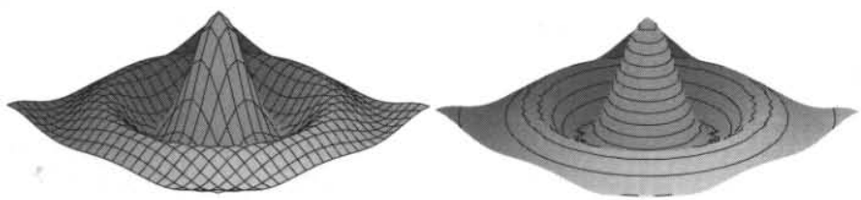

Bild 23.2: Verschiedene grafische Darstellungen (patch und patchcontour) einer Funktion f(x,y) in zwei Variablen

23.5 Programmieren mit Maple

Bei der Benutzung von Maple als Progammiersprache ist es nicht erforderlich, jede benutzte Variable oder Funktion zu definieren. Maple bestimmt den Typ eines jeden eingegebenen Ausdrucks, der anschließend noch durch die Abfrage `type` überprüft und mit verschiedenen anderen Typen durch Eingabe des `convert` Befehls konvertiert werden kann.

23.5.1 Kontrollstrukturen

Einfache Konstruktionen werden bei Verwendung der Programmstrukturen durch Schleifenbildung mit `for` oder `while` Verzweigungen mit `if` und Unterprogrammstrukturen mit `proc` erzielt.

Eine **for-Schleife** hat die Syntax:

```
> for <index> from <start> to <ende>
  do <anweisungen>: end do;
```

Eine **while-Schleife** hat die Syntax:

```
> while bedingung
  do anweisungen: end do;
```

Eine **if-else-Anweisung** hat die Syntax:

```
> if bedingung then anweisungen:
                 else anweisungen: end if;
```

Eine **proc-Konstruktion** hat die Syntax:

```
> proc() local var; anweisungen end;
```

Eine for- oder while-Schleife, eine if-else-Anweisung oder eine proc-Konstruktion müssen bei Maple immer innerhalb eines Anweisungsblocks `do ... end do` stehen.

Beispiel: for-Schleife für die Summenbildung der ersten 100 Zahlen:

```
> summe := 0:
> for i from 1 to 100
> do
>    summe := summe + i ;
> end do:
> summe;
```

Beispiel: Eine entsprechende while-Schleife sieht wie folgt aus:

```
> summe := 0: i := 0:
> while i <= 100
> do
```

```
>     summe := summe + i ;
>     i := i + 1;
> end do:
> summe;
```

Der Abschluss `end do` ist auch durch `od` austauschbar, `end if` ist austauschbar durch `fi`. Wohl definierte Abfragen enden entweder mit `true` (wahr) oder `false` (falsch), inkorrekt formulierte Abfragen und unbestimmte Ergebnisse geben `FAIL` zurück.

Beispiel: if-else-Anweisung zur Bestimmung des Absolutwerts von zahl

```
> if zahl < 0
> then betrag: = -zahl:
> else betrag: =  zahl:
> end if:
```

Beispiel: proc-Konstruktion zur Berechnung der Summe der ersten N Zahlen:

```
> summe := proc( )
> local i, summe, N;
>    N: = args [1]:
>    summe := 0:
>    for i from 1 to N
>        do summe := summe + i:
>    end do:
> end:
> summe (10);
55
```

Die Variablen i, summe und N sind innerhalb der Prozedur lokale Variablen.

Eine lokale Variable ist nur innerhalb des Prozedurkörpers, der durch die `statement-sequence` abgeschlossen ist, bekannt. Eine einmal deklarierte globale Variable ist auch weiterhin außerhalb des Prozedurkörpers für die Dauer der gesamten Maple-Sitzung bekannt. Wie das folgende Beispiel zeigt, kann auch eine Funktion mit Hilfe einer Prozedur definiert werden.

Beispiel: Definition von f als Funktion der Variablen x

```
> f := proc (x)
> x^3 + x + 1
> end:
```

Durch f(2) kann der Funktionswert an der Stelle x=2 ausgegeben werden.

Eine Auflistung von Ausdrücken, die durch Kommata voneinander getrennt sind, ist eine so genannte Folge von Ausdrücken (**expression sequence**).

Eine spezielle, mit null bezeichnete Folge enthält keine Elemente. Eine bestimmte Folgen-Vorschrift wird mit der Funktion seq erzeugt und ähnelt einer for-Schleife.

Beispiel: Durch Einsetzen der seq-Anweisung wird für die Variable x eine Folge mit Exponenten von der 20. bis zur 27. Primzahl erzeugt:

```
> seq (x^ ithprime(n), n = 20..27);
 x71, x73, x79, x83, x89, x97, x101, x103
```

Durch Einschließen der Iterationsvariablen `n` in Anführungszeichen (`'n'`) kann die hier zugewiesene Definition `n = 20..27` wieder freigegeben werden. Durch Einsetzen des `$-Operators` in dieselbe Folge wird der `seq`-Befehl verkürzt zu:

```
> n := 'n':
> x^ ithprime(n) $ n = 20..27;
```

Das Ergebnis bleibt unverändert. Sehr häufig wird zusätzlich der `$-Operator` zur Berechnung einer Ableitung von hoher Ordnung zusammen mit der `diff`-Funktion verwendet.

23.5.2 Zeichenketten

In Maple besteht eine einfache Zeichenkette (`simple string`) aus einer Kombination von Buchstaben und Ziffern. Eine in Anführungszeichen gesetzte Zeichenkette (`quoted string`) beinhaltet eine beliebige Zeichenkombination, die in hochgestellte Anführungszeichen (`‘ ’`) eingeschlossen ist. Sie kann Leerstellen, Sonderzeichen wie `+, /` und Steuerbefehle wie Zeilenumbruch enthalten. Die Kennzeichnung der leeren Zeichenkette ist (`‘ ’`).

Beispiel: Eine einfache (`simple string`) Zeichenkette ist

```
> simple := 10;
simple := 10
```

Eine in Anführungszeichen gesetzte Zeichenkette (`quoted string`) kann wie folgt aussehen:

```
> 'strange +/  quoted-string' := 20;
  strange +/  quoted-string := 20
```

Beide Zeichenketten (`simple` und `quoted string`) ergeben ineinander verschachtelt:

```
> simple + 'strange +/''quoted-string.':
 30
```

23.5.3 Bereiche, Listen und Mengen

Eine Bereichsangabe in Maple ist ein Ausdruck der Form `a..b`.

Die Bedeutung der Zahlen `a` und `b` hängt vom speziellen Kontext ab. In den `seq`- und `sum`-Anweisungen steht `a..b` für die Zahlen `a, a+1, a+2, .., a+k`, wobei `a+k` nahe `b` ist, ohne dieses zu überschreiten.

```
> sum ('i^2', 'i' = 1..10);
 385
```

Um die Stetigkeit von Funktionen zu testen, wird mit der `readlib`-Anweisung der `iscont`-Befehl aufgerufen, in dem der Bereich `a..b`, falls nicht anders zugewiesen, für das offene reelle Intervall `(a,b)` steht.

```
> readlib (iscont):
> iscont (tan(x),  x = -Pi/2..Pi/2);
 true
```

Eine Bereichsangabe als Ergebnis liefert die Funktion `limit`

```
> limit (sin(1/x),  x = 0);
-1..1
```

Der Operator `$` erzeugt aus einer Bereichsangabe eine Folge.

```
> $  1..5;
 1, 2, 3, 4, 5
```

Eine durch eckige Klammern `[ ]` eingeschlossene Folge von Ausdrücken ist eine Liste (`list`), die eine beliebige Kombination von Ausdrücken einschließlich anderer Listen enthalten kann, deren Reihenfolge von Maple nur durch explizite Aufforderung geändert wird. Eine leere Liste ist durch leere Klammern `[ ]` gekennzeichnet.

Der Befehl `op(ausdr);` liefert die Folge aller in einer Liste vorkommenden Elemente, Operanden und Komponenten zurück. Diese `op`-Funktion, versehen mit zwei Argumenten eignet sich aber auch, um bei Bedarf auf ein ausgewähltes Element oder auf einen besonderen Bereich von Elementen aus einem Ausdruck, aus einer Liste zuzugreifen: `op(i,L);` oder z.B. mit `op(bereich, ausdr);`. Die Abfrage der Anzahl in einem Ausdruck enthaltener Operanden gelingt durch Eingabe des verwandten Befehls `nops(ausdr);`.

Beispiel:

```
> list1 := [Pi, exp(1), Catalan]:
> list2 := [Pi, mu, epsilon]:
> op(list1);
π, e, Catalan
```



```
> op(1, list1);
π
> biglist := [op(list1), op(list2)];
biglist := [π,e,Catalan, π,μ,ε]
> nops(biglist);
6
> biglist [3..5];
Catalan, π,μ
> op(3..5, biglist);
Catalan, π,μ
```

In Zusammenhang mit Listen ist die Benutzung einer `do`-Schleife in Kombination mit einer `for`-Anweisung erlaubt.

Beispiel: Berechne die erste, vierte, sechsundzwanzigste und vierhundertvierzigste Primzahl.

```
> for i in [1, 4, 26, 440]
  do ithprime (i) od;
 2, 7, 101, 3079
```

Aus einer vorhandenen Liste ist mit dem `select`-Befehl und unter Eingabe eines Kriteriums mit `isprime` die Auswahl einer neuen Liste von Elementen möglich, die das Kriterium erfüllen. Der `select`-Befehl in Kombination mit einem `type` Test erlaubt die Auswahl von bestimmten Elementen aus einer Liste. Die Überprüfung, ob ein bestimmter Ausdruck, ein bestimmtes Element in einer Liste ist, erfolgt mit dem Befehl `member`.

Beispiel: Gesucht sind die Primzahlen und die geraden Zahlen aus einer Liste ganzer Zahlen.

```
> select(isprime, [$ 3000..3050]);
 [3001, 3011, 3019, 3023, 3037, 3041, 3049]
> select(type, [$ 3000..3010], even);
 [3000, 3002, 3004, 3006, 3008, 3010]
```

Beispiel: Ist das Element a in einer Liste [a, b, c, d]?

```
> member (a, [a, b, c, d]);
 true
```

Eine einheitliche Verarbeitung aller Listen-Elemente gestattet die `convert`-Funktion, gefolgt von einem Unterbefehl wie z.B. der Addition (der Summe) aller Listen-Elemente mit `> convert (list, '+');` oder der Multiplikation (dem Produkt) mit `> convert (list, '*');`.

Beispiel: Gesucht ist das arithmetische Mittel einer Zahlenliste:

```
> grades := [58,69,73,78,84,85,99]:
> convert(grades, '+') /nops(grades);
  78
```

Eine Folge von in geschweiften Klammern { } eingeschlossenen, voneinander verschiedenen Ausdrücken ist eine Menge (set). Im Unterschied zu einer Liste sind die Elemente einer Menge völlig ungeordnet, können aber mit Befehlen geordnet werden, wobei alle doppelt erscheinenden Elemente gestrichen werden. Auch bei der Menge existiert die leere Menge { }.

Die in den vorangegangenen Abschnitten vorgestellten Befehle: `op; nops; select; member; convert /'+'; convert /'*';` die `do-` und `for-`Schleife können völlig analog auch für Mengen eingesetzt werden. Spezielle Mengenoperatoren dagegen sind: `intersect; union;` und `minus;`.

Beispiel: Bestimmung der Vereinigungsmenge und der Schnittmenge

```
> set1 := {0, Pi, 2*Pi, 3*Pi}:
> set2 := {Pi, exp(1), Catalan}:
> set1 union set2;
  {0, Catalan, π, e, 2π, 3π}
> set1 intersect set2;
  {π}
```

Die Befehle `[op(menge)]` und `convert(menge,list)` wandeln eine Menge in eine Liste um, `convert (list,set)` oder `{op(list)}` formen eine Liste in eine Menge um.

23.5.4 Felder und Matrizen

Eine mehrdimensionale Datenstruktur ist ein Feld (array), das z.B. im Falle eines zweidimensionalen Feldes aus drei Reihen und zwei Spalten mit Einträgen bestehen kann. Ein zweidimensionales Feld `a[i,j]`, bestehend aus einem Zeilen- und Spaltenindex in beiden Dimensionen mit 1 beginnend, ist in Maple eine Matrix. Die Definition einer Matrix erfolgt zeilenweise, indem jede Zeile in Form einer Liste angegeben wird. Die Erstellung einer Matrix in Maple erfolgt mit dem Befehl `matrix`.

Der `array`-Befehl akzeptiert auch zusätzliche Parameter wie z.B. eine Liste von Anfangswerten und eine Indexfunktion zur Einschränkung der

Werte eines Feldes. Fünf Indexfunktionen sind zur Definition und Einschränkung von Feldern und Matrizen wichtig:

Tabelle 23.3: Spezielle Arten von Matrizen

Funktion	Beschränkung
`symmetric`	`A[i,j] = A[j,i]`
`antisymmetric`	`A[i,j] = -A[j,i]`
`diagonal`	Elemente außerhalb der Diagonalen sind 0
`identity`	Diagonalelemente sind 1, alle anderen 0
`sparse`	nicht explizit initialisierte Einträge werden als 0 vorausgesetzt

Beispiel: Die Erzeugung einer 3×3-Einheitsmatrix in Maple erfolgt mit den nachstehenden Anweisungen:

```
> id3 := array(identity, 1..3, 1..3):
> print (id3);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Eine einfachere Art, Felder und Matrizen zu definieren, liefern die eingebaute `matrix-` und ebenso die `vector-`Funktion zur linearen Algebra in dem `linalg`-Paket (`linalg-package`), das mit dem Befehl `>with(linalg)` zu öffnen ist. Die Definition einiger Vektoren und der zugehörigen Matrix erfolgt zeilenweise in Form einer Liste und ist in dem nachstehenden Beispiel zu sehen:

Beispiel:

```
> with(linalg):
> v1 := vector([1, 1, 1]):
> v2 := vector([1, 2, 4]):
> v3 := vector([1, 3, 9]):
>  m := matrix ([v1,v2,v3]);
```

$$m := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix}$$

Da der Operator * nur für die kommutative Multiplikation erlaubt ist, muss für die Matrizenmultiplikation die Operationssyntax &* verwendet werden.

23.5.5 Tabellen

Eine Tabelle (**table**) ist in Maple eine Datenstruktur, die mit verschiedenen Einträgen beliebiger Ausdrücke erstellt ist, die keine spezielle Initialisierung benötigen.

Beispiel:

```
> days[Jan] := 31:
> days[Feb] := 28:
> days[Mar] := 31:
> days[FirstQuatrer]:=days[Jan]+days[Feb]+
    days[Mar];
```

$$days_{FirstQuarter} := 90$$

Die komplette Tabelle kann mit dem `print`-Befehl ausgedruckt werden:

```
> print(days);
table([Mar=31,FirstQuater=90,Jan=31,Feb=28])
```

Die Tabelle zeigt, dass die Reihenfolge der Einträge in einer nicht vorher bestimmbaren und nicht beeinflussbaren Reihenfolge ausgedruckt wird.

23.5.6 Definition von Funktionen

Beim Starten einer Maple-Sitzung werden einige grundlegende Funktionen wie `op`, `type`, `evalf` und mathematische Befehle wie `max` und `diff` mit dem Hauptprogramm direkt in den Kernel übersetzt und stehen deshalb dem Benutzer sofort zur Verfügung. Die meisten der gebräuchlichen mathematischen Befehle in Maple wie `int`, `solve`, `plot`, `exp` werden nach der ersten Benutzung als Standard-Bibliotheksfunktionen automatisch geladen. Ihr Quellcode ist für den Anwender anschaubar durch die Eingabe:

```
> interface(verboseproc = 2);
> print (op);
```

Maple bietet zwei grundlegende Ansätze zum Schreiben einer eigenen Funktion zur Berechnung einfacher Formeln an.

Eine mathematische Funktion kann durch Benutzung des **Pfeiloperators** -> definiert werden. Dazu dient die Funktion `unapply`, die auf einen bereits definierten mathematischen Ausdruck angewandt werden kann und einen Funktionaloperator (eine Pfeilfunktion) zurückgibt, die den Ausdruck berechnet.

Auch die Definition einer Funktion mit zwei oder mehr Variablen ist mit Hilfe des Pfeiloperators -> möglich.

Beispiel:

```
> g := (s, t)-> sqrt(s^2 + t^2);
```

$$g := (s,t) \rightarrow sqrt(s2 + t2)$$

```
> g (105, 208);
```

$$233$$

Beispiel: Umwandlung eines mathematischen Ausdrucks in eine Funktion.

```
> y := 1 + sqrt(x):
> g := unapply(y,x); g(1);
```

$$g := x \rightarrow 1 + \sqrt{x}$$

$$2$$

Die Unterscheidung von Funktionen und Ausdrücken ist häufig wichtig, da die Befehle teilweise unterschiedlich lauten oder auch unterschiedlich aufgerufen werden müssen. Für elementare Funktionen, für die Definition von Funktionen und zusammengesetzten Funktionen stehen weitere Aufrufe mit `inifunctions`, `piecewise` etc. zur Verfügung.

23.6 Fehlersuche

Um auftretende Fehler aufspüren zu können, helfen die folgenden Abfragen:

- Bekam eine Variable bereits während dieser Maple-Sitzung einen Wert zugewiesen (Zweifachzuweisung)?
- Wurde versehentlich einer Funktion oder einer globalen Variablen ein Wert zugewiesen?
- Wurden alle erforderlichen Pakete und Bibliotheken geladen?
- Besitzen alle aufgerufenen Funktionen die richtige Anzahl und Typen für ihre nachgestellten Argumente?
- Sind die benutzten Indexvariablen voneinander verschieden?
- Sind alle erforderlichen Satzzeichen gesetzt worden?
- Öffnen und schließen sich alle benutzten runden, eckigen und geschweiften Klammern?
- Wurde die Groß- und Kleinschreibung exakt befolgt?

- Wurde jede Anweisung korrekt abgeschlossen und liefert sie das gewünschte Ergebnis?

Der Aufruf und die Anwendung des Progamms **Mint** unterstützen die Fehlersuche. Mint spricht Warnungen unterschiedlicher Wichtung aus und meldet Syntaxfehler innerhalb einer Datei. Durch Einladen der Bibliotheksfunktionen `showtime` und `history` können die benutzte Zeit, der Speicherplatz und in durchnummerierter Reihenfolge die Ergebnisse eines Befehls nachträglich abgefragt werden. Die Effizienz einer Prozedur verbessert sich durch die Eingabe der Option `remember`, mit der sich die Prozedur an den Wert jeder vorangegangenen Eingabe erinnert.

23.7 Ausgabe und Speicherung

Ausdrücke lassen sich mit dem `save`-Befehl in Dateien abspeichern. Der `read`-Befehl ermöglicht das Einlesen von Definitionen und Anweisungen aus einem abgespeicherten Programm oder aus einer Datei.

Abspeichern in Dateien und Einlesen von Daten aus Dateien kann auch über die Maple Icons erfolgen. Grafiken können durch Betätigen der rechten Maustaste abgespeichert werden.

Es gibt drei Arten von Maple-Ausgaben, die durch die `interface` Variable `prettyprint` gesteuert werden. Ist `prettyprint` gleich `0`, so wird die Ausgabe in einem linearen Format ausgegeben. Ergebnisse, die in diesem Format abgespeichert sind, können später als Eingabe weiterverwendet werden.

Beispiel:

```
> interface (prettyprint = 0):
> y := Int((sqrt(1+x^2),x));
 y := Int((1+x^2)^(1/2),x)
```

Hat die Variable von `prettyprint` den Wert 1, so erfolgt die Ausgabe über ein Mehrzeilenformat.

Der obige Ausdruck erscheint damit in folgender Darstellung:

```
        /
        |           2
y :=    |  sqrt(1 + x )  dx
        |
        /
```


Mit dem Befehl `prettyprint = 2` erfolgt der Ausdruck in der üblichen mathematischen Schreibweise mit Summen-, Integral- und Quadratwurzelzeichen, mit Matrixklammern und griechischen Buchstaben.

$$y := \int \sqrt{1+x^2}\,dx$$

Bibliografie und Webliografie

[Bu96] Burkhardt, W.: *Erste Schritte mit Maple*. Springer 1996.

[Devitt04] Devitt, J.S.: *Calculus with Maple V*. Brooks/Cole 1994.

[DoGo95] Dodson, C., Gonzalez, E.: *Experiments in Mathematics Using Maple*. Springer 1995.

[Ellis96] Ellis, W. et al: *Maple V Flight Manual*. Brooks/Cole 1996.

[Heck96] Heck, A.: *Introduction to Maple*. Springer 1996.

[HeiJa95] Heinrich, E., Janetzko, H.D.: *Das Maple Arbeitsbuch*. Vieweg, Braunschweig 1995.

[KoBiKo01] Kofler, M., Bitsch, G., Komma, M.: *Maple (Einführung, Anwendung, Referenz)*. Addison-Wesley 2001

[Werner98] Werner, W.: *Mathematik lernen mit Maple (Band 1 + 2)*. dpunkt 1996 + 1998.

[Wester02] Westermann, T.: *Mathematik für Ingenieure mit Maple (Band 1 + 2)*. Springer 2001 + 2002.

[WeBuDi01] Westermann, T., Buhmann, W., Diemer, L., Endres, E., Laule, M., Wilke, G.: *Mathematische Begriffe visualisiert mit Maple*. Springer 2001.

[MSoft] MapleSoft Website http://www.maplesoft.com

[MApps] Maple Application Center http://www.mapleapps.com

24 Matlab

Britta Nestler

Matlab ist ein interaktives kommerzielles Softwarepaket zur nummerischen Lösung mathematischer Probleme. Eine erste Version von Matlab entstand in den 1970er Jahren auf der Basis der Arbeiten des Mathematikers Cleve Moler. Da Matlab zur nummerischen Lösung verifizierte und etablierte mathematische Algorithmen verwendet, sind die berechneten Ergebnisse sehr verlässlich. Mit nur wenigen Befehlen können aufwendige und komplexe Lösungsverfahren und Prozeduren durchgeführt werden. Durch die Formulierung eines speziellen Satzes von Funktionen lassen sich konkrete Anwendungen behandeln. Zur Visualisierung als 2D- oder 3D- Grafik verfügt Matlab über hervorragende grafische Darstellungsmöglichkeiten.

24.1 Einfache Berechnungen

Die grundlegenden arithmetischen Operationen + - * / ^ werden zusammen mit Klammern () verwendet. Das Symbol ^ berechnet Exponenten: `2^3 = 8`. Auszuführende Befehle werden mit einem Prompt eingeleitet:

Beispiel:

```
>> 3-2^4
ans = -13
>> ans*5
ans = -65
```

Das Ergebnis der ersten Rechnung wird automatisch mit `ans` bezeichnet und für die zweite Rechnung weiterverwendet. Der Wert von `ans` wird dabei überschrieben und mit einem neuen Wert belegt.

Mit den Cursor-Tasten ↑ und ↓ kann man auf vorherige Matlab Eingaben zurückgreifen. Um einen bestimmten Befehl mit bekanntem Anfangsbuchstaben wiederherzustellen, kann der Anfangsbuchstabe gefolgt von ↑ eingegeben werden.

Die Berechnung arithmetischer Ausdrücke erfolgt entsprechend den Prioritäten:

1. Größen in Klammern,
2. Potenzen
3. Multiplikation * und Division / von links nach rechts.
4. Addition + und Subtraktion - von links nach rechts.

Der Benutzer kann den berechneten Größen eigene Namen zuweisen und später auf die so definierten Größen zurückgreifen.

Beispiel:

```
>> x = 3-2^4
x = -13
```

Dies ist ein Beispiel für eine Zuweisungsangabe: Werte sind Variablen zugewiesen. Spezielle Namen wie z.B. `eps,pi` sind bei Matlab vorbelegt und sollten daher nicht zur Namensgebung verwendet werden.

Jede Variable muss zunächst einen Wert zugewiesen bekommen, bevor sie auf der rechten Seite einer Gleichung verwendet wird. Als Variablenbezeichnung können beliebige Buchstabenkombinationen oder Ziffern beginnend mit einem Buchstaben, gewählt werden.

Matlab erkennt verschiedene Arten von Zahlen.

Tabelle 24.1: Zahlentypen

Zahlentyp	Beispiel
integer	2648, -54279
relle Zahl	3.861, -18.57
komplexe Zahl	6.16+2.1i (i=$\sqrt{-1}$)
Inf	unendlich
NaN	Not a Number

Es wird die so genannte wissenschaftliche Notation verwendet.

Beispiele:

`-2.1475e+03` = -2.1475×10^3 = -2147.5

Die Berechnungen in Matlab erfolgen in der Genauigkeit `double`. Das Ausgabeformat lässt sich durch den Befehl `format` steuern:

Tabelle 24.2: Formatfestlegung in Matlab

Befehl	Ausgabebeispiel
format short	3.1416 (4 Dezimalstellen)
format short e	3.1416e+01
format long e	3.141592653589793e01
format bank	3.14 (2 Dezimalstellen)
format compact	unterdrückt Leerzeilen
format	Zurücksetzen auf das default-Format

Mehrere Befehle können in einer Zeile aufgeführt werden. Sie sind jeweils durch Kommata oder Semikolons zu trennen.

Um eine Ausgabe zu unterdrücken, muss die Zuweisungsangabe bzw. der mathematische Ausdruck mit einem Semikolon beendet werden.

Beispiel:

```
>> x = 4*2; y = 2*x, z = y^2
y = 16
z = 256
```

Matlab verfügt über die elementaren Funktionen `abs, sqrt, exp, log, log10` sowie über die trigonometrischen Funktionen `sin, cos, tan, sinh, cosh, tanh` und deren inverse Funktionen `asin, acos, atan, asinh, acosh, atanh.`

Beispiel:

```
>> x = 9; sqrt(x), exp(x), y = 5*sin(pi/6)
ans = 3
ans = 8.1031e+03
y = 2.5000
```

Weitere elementare Funktionen sind `round` (Runden zur nächsten integer), `floor` (Abrunden), `fix` (Runden gegen null) und `ceil` (Aufrunden).

24.2 Datenspeicherung und Steuerungsbefehle

24.2.1 Speicherung von Sitzungen

Die Eingabe der Anweisung

```
>> diary mysession
```

bewirkt, dass der weitere Text auf dem Bildschirm in einem File namens `mysession` abgespeichert wird. Die Datei wird in dem Verzeichnis angelegt, von dem aus Matlab aufgerufen wurde. Es kann ein beliebiger Filename angegeben werden außer `on` und `off`.

Das Speichern einer Matlab-Sitzung kann mit `>> diary off` abgebrochen werden. Die Datei `mysession` kann mit einem Texteditor geöffnet und weiter bearbeitet werden. Falls beabsichtigt ist, eine Matlab-Sitzung während einer Berechnung zu verlassen und zu einem späteren Zeitpunkt fortzusetzen, werden durch

```
>> save thissession
```


die laufenden Werte der Variablen in einer Datei `thissession.mat` gespeichert. Dieses File kann nicht editiert werden. Beim nächsten Starten von Matlab kann die Berechnung fortgesetzt werden durch

```
>> load thissession
```

24.2.2 Skriptfiles

Ein Skriptfile ist eine ASCII-Textdatei, die Matlab-Befehle enthält. Es ist erforderlich, dass die Datei die Endung `.m` besitzt, z.B. `problem1.m`. Die Befehle lassen sich in einer Matlab-Sitzung ausführen durch den Aufruf `>> problem1.`

Hierbei wird nur das Ergebnis der Befehle am Bildschirm angezeigt, nicht aber die Befehle selbst. Die Befehle werden angezeigt durch `>>echo on` und entsprechend durch `>> echo off` wieder ausgeblendet. Text, der einem % folgt, wird ignoriert. Auf diese Weise können in das Matlab-Dokument erklärende Kommentare zur Vorgehensweise und zur Anwendung eingebracht werden.

24.2.3 Steuerungsbefehle

Eine Liste der Variablen, die innerhalb der laufenden Sitzung verwendet werden, wird durch `>> whos` angegeben.

Der Befehl `>> help` gibt eine Liste mit Kategorien an, für die Matlab eine Hilfe besitzt. Mit `>> help topic` erhält man genauere Informationen zu einem speziellen Themengebiet.

Tabelle 24.3: Allgemeine Steuerungsbefehle in Matlab

Befehl	Bedeutung
help	Interne Hilfefunktion, online Dokumentation
doc	Laden von Textdokumentationen
what	Verzeichnisauflistung aller M-, MAT- und MEX -Files
type	Auflistung von M-Files
lookfor	Stichwortsuche in der Hilfe-Dokumentation über alle Einträge
which	Lokalisierung von Funktionen und Files
who	Auflistung der aktuell verwendeten Variablen
whos	Detaillierte Auflistung der aktuell verwendeten Variablen
load	Wiederherstellen von Variablen auf der Festplatte

Befehl	Bedeutung
save	Abspeichern des Arbeitsplatzes auf die Festplatte
clear	Entfernen der Variablen und Funktionen aus dem Speicher
cd	Wechseln des aktuellen Arbeitsverzeichnisses
dir	Auflistung aller Verzeichnisse
delete	Löschen des Files
diary	Speichern des Textes einer Matlab-Sitzung
cedit	Einfügen einer neuen Befehlszeile
clc	Löschen des Eingabefensters
format	Festsetzen der Ausgabe auf Format
quit	Verlassen einer Matlab-Sitzung

24.3 Grafische Darstellung von Funktionen

24.3.1 Einfache Plots

Um den Grafen einer Funktion, z.B. y=sin(3*pi*x), in einem Intervall 0<x<1 zu zeichnen, wird die Funktion an ausreichend vielen Punkten ausgewertet, die resultierenden Koordinatenpaare (xi,yi) werden durch Geradenstücke verbunden und durch

```
>>plot(x,y)
```

grafisch dargestellt. Für N+1 Punkte gleichen Abstands h werden Punkte x = 0, h, 2h, ... 1-h, 1 definiert und der entsprechende y-Wert der Funktion bestimmt:

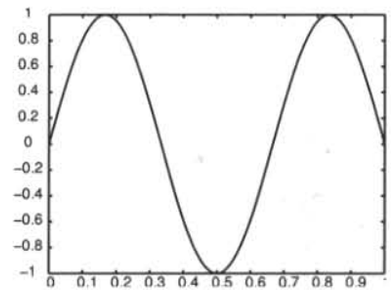

Bild 24.1: Graf der Funktion sin(3*pi*x) im Intervall [0,1] ausgewertet an 100 Stützpunkten

Beispiel:

```
>> N = 100; h = 1/N; x = 0:h:1;
>> y = sin(3*p*x); plot(x,y)
```

Die Bezeichnung eines Titels und die Beschriftung der Achsen wird erreicht durch `>> title('Titel der Abbildung')`, `>> xlabel ('x-Achse')` und `>> ylabel ('y-Achse')`.

Mit `>> grid` wird ein gepunktetes Gitter in die Grafik eingeblendet und mit `>> grid off` wieder ausgeschaltet. Eine beschreibende Legende kann eingefügt werden durch `>> legend ('Legende zu dem Grafen')`.

Die Bezeichnung innerhalb der Strings '...' kann frei gewählt werden. Die Legende wird von Matlab in eine geeignete Position gelegt, in der sie den Grafen nicht behindert. Sie kann manuell mit der Maus verschoben werden. Die default-Einstellung zeichnet den Funktionsgrafen mit einer schwarzen Linie ohne Strichelung. Sowohl die Farbe als auch der Linientyp sind einstellbar, siehe Tabelle 24.4.

Tabelle 24.4: Optionen für Farben und Linientyp einer grafischen Darstellung

Farbe	Linientyp
y yellow	* Markierung mit Sternen
m magenta	o Markierung mit Kreisen
c cyan	× Markierung mit Kreuzen
r red	+ Markierung mit Pluszeichen
g green	• Markierung mit Punkten
b blue	- durchgezogene Linie
w white	: gepunktete Linie
k black	-- gestrichelte Linie
	-. Strich-Punkt-Linie

Beispiel:

```
>> plot(x,y,'b--')
```

Diese Anweisung liefert eine blaue gestrichelte Linie.

Durch Auflistung der einzelnen Funktionsausdrücke werden mehrere Kurven in einem Diagramm gezeichnet.

Beispiel:

```
>>plot(x,sin(3*π*x),'k-',x,cos(2*π*x),'k--')
>> legend('sinus','cosinus')
>> title('Trigonometrie')
>> xlabel('x Achse'), ylabel('y Achse')
>> grid
```

Als Ergebnis der Anweisungen ergibt sich das Diagramm in Bild 24.2.

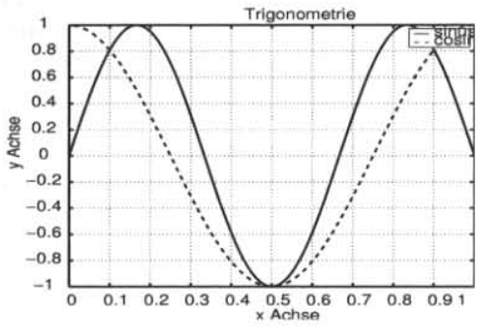

Bild 24.2: Zeichnung der Funktionen sin(3πx) und cos(3πx) in einem Diagramm mit verschiedenen Linientypen einschließlich Titel, Achsenbeschriftung und Legende

24.3.2 Grafikbefehle

Der Befehl `>> axis` besitzt vier Parameter. Die ersten beiden Größen legen das Minimum und Maximum entlang der x-Achse fest, die letzten beiden Werte den Bereich entlang der y-Achse.

Beispiel: Für `>> axis([0.2 1.8 -0.5 1.5])` wird die x-Achse im Intervall [0.2,1.8] und die y-Achse im Intervall [-0.5,1.5] angelegt.

Mit `>> zoom` lässt sich ein Teil einer Zeichnung detaillierter auflösen (Mausklick links/rechts bzw. Auswahl).

Ein Grafikfenster wird mit `>> subplot` in eine m×n-Matrix aus kleineren Fenstern unterteilt, in die einzelne Diagramme gezeichnet werden. Die Teilfenster werden mit `1` bis `mn` zeilenweise durchnummeriert, beginnend mit dem linken oberen Teilfenster.

Beispiel:

```
>> subplot(221), >> subplot(222), ...,
>> subplot(224)
```

teilt ein Fenster in eine 2×2-Matrixanordnung mit vier Bildern. Die ersten beiden Ziffern geben die Dimension der Bildermatrix an, die letzte Ziffer legt die Nummer des laufenden Bildes fest.

Durch einen neuen Aufruf `>> plot` wird als default-Vorgang das Grafikfenster gelöscht und der aktuelle Graf in einem neuen Fenster gezeichnet. Um eine Grafik zu verwahren und das Löschen des Grafikfensters zu verhindern, dient der Befehl `>> hold on`. Entsprechend gibt `>> hold off` das aktuelle Bild frei, jedoch ohne das

Grafikfenster zu löschen. Das Löschen des Grafikfensters wird durch `>> clg` bewirkt.

Ein Matlab-Diagramm kann mit

```
>> print -deps datei
```

als Encapsulated PostScript gespeichert werden. In Tabelle 24.5 sind einige wichtige Grafikbefehle zusammengefasst.

Tabelle 24.5: Zusammenfassung wichtiger Grafikbefehle

Befehl	Bedeutung
figure	Anlegen eines neuen Bildes bzw. Grafikfensters
clf	Löschen des aktuelles Bildes
close	Schließen des aktuellen Bildes und Grafikfensters
subplot	Aufteilen des Grafikfensters in Unterdiagramme
axis	Bestimmung der Achsenskalierung
hold	Festhalten der aktuellen Grafik
text, gtext	Einfügen von Text, Platzierung von Textelementen
print	Speichern des Diagramms in ein File
plot	Zeichnen eines linearen Diagramms
loglog	Zeichnen eines log-log Diagramms
polar	Polarkoordinaten
bar	Zeichnen eines Säulendiagramms
errorbar	Zeichnen von Fehlerbalken
hist	Histogramm
title	Beschriftung des Diagramms mit einem Titel
xlabel, ylabel zlabel	Festlegen der Achsenbeschriftung
grid	Einblenden von Gitterlinien
contour	Contourzeichnung
mesh	3D-Gitteroberfläche
surf	3D-schattierte Fläche

24.3.3 Zeichnen von Oberflächen

Mathematisch ist eine Oberfläche durch eine Funktion `f(x,y)` definiert, die jedem Koordinatenpunkt `(x,y)` eine Höhe `z=f(x,y)` zuordnet. Um eine solche Fläche im Raum zu zeichnen, ist es erforderlich, zunächst die x- und y-Achsenbereiche festzulegen. Dies liefert ein

rechteckiges Gebiet in der (x,y) Ebene. Weiterhin wird mit meshgrid auf diesem Gebiet ein Gitter gewählt. Der Funktionsausdruck wird auf jedem diskreten Gitterpunkt des erzeugten Gitters ausgewertet und mit plot gezeichnet.

Beispiel: Gezeichnet wird die Oberfläche der Funktion

$f(x,y) = (x-3)^2-(y-2)^2$ für $2 \le x \le 4$; $1 \le y \le 3$.

```
>> [X,Y]=meshgrid(2:.2:4,1:.2:3);
>> Z=(X-3).^2-(Y-2).^2;
>> mesh(X,Y,Z)
>> title('Sattel'),xlabel('x'),ylabel('y')
```

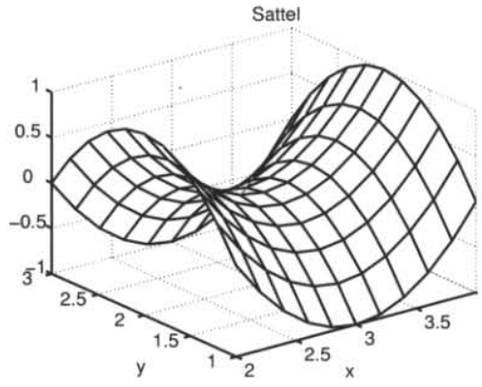

Bild 24.3: Zeichnung der Oberfläche einer Sattelfunktion

24.4 Vektoren und Vektoroperationen

Zeilenvektoren werden durch eine Liste von Zahlen angegeben, die entweder durch Kommata oder durch Leerstellen voneinander getrennt sind. Die Komponenten eines Vektors werden in rechteckige Klammern gesetzt. Mit dem Befehl >> length kann die Anzahl der Einträge abgefragt werden.

Beispiel:

```
>> v = [1, 3  sqrt(4)]
v = 1.000   3.0000  2.0000
>> length(v)
ans = 3
```

Vektoren können mit einem Skalar multipliziert oder - bei derselben Länge - komponentenweise addiert bzw. subtrahiert werden.

Doppelpunktnotation, Iterator: Durch a:b:c können die Komponenten eines Vektors produziert werden, beginnend mit dem Wert a, steigend um den Wert b, bis der Wert c erreicht ist.

Beispiel:

```
>> v = 2:6, w = 0.32:0.1:0.6
v = 2   3   4   5   6
w = 0.32   0.42   0.52
```

Spaltenvektoren werden generiert, indem die Komponenten durch Semikolons oder neue Zeilen voneinander abgetrennt werden. Durch Transponieren mit dem Befehl ' können Zeilenvektoren in Spaltenvektoren konvertiert werden und umgekehrt.

Beispiel:

```
>> a = [1; 3; sqrt(25)], transponiert = a'
a = 1.0000
    3.0000
    5.0000
transponiert = 1.0000  3.0000  5.0000
```

Das Skalarprodukt je eines gleich langen Zeilenvektors und Spaltenvektors wird durch den Operator * berechnet.

Beispiel:

```
>> v = [4, 2, 1]; w = [3;-1;0];
>> skalarprodukt = v * w
skalarprodukt  =  10
```

Der Betrag (Norm) eines Vektors wird durch die Funktion norm bestimmt.

Eine andere Art der Produktbildung zweier Vektoren ist in der Mathematik als Hadamard-Produkt oder Punktprodukt bekannt. Dabei müssen die Vektoren von derselben Länge und von demselben Typ, also beide Zeilen- oder beide Spaltenvektoren sein. Das Ergebnis ist ein Vektor desselben Typs. In Matlab wird diese Punktproduktbildung durch den Operator .* realisiert.

Beispiel:

```
>> u =[2,4,1]; v=[0,3,5]; hadamard = u.*v
hadamard = 0  12  5
```

Mathematisch ist die Division eines Vektors durch einen anderen nicht definiert. Matlab verfügt über einen Operator ./, der zwei Arrays komponentenweise dividiert. Der Operator benötigt zwei Vektoren derselben Länge und desselben Typs.

Beispiel:

```
>> a = 1:5; b = 6:10; division = a./b
```

```
division = 0.1667 0.2857 0.3750 0.4444 0.5000
```

Division durch 0 ergibt `Inf` und ein Ausdruck der Form 0/0 ergibt `NaN`. Analog zur komponentenweisen Multiplikation und Division gibt es eine komponentenweise Potenzbildung eines Vektors. Der entsprechende Operator ist .^.

Beispiel:

```
>> u = [10  11  12]; potenz = u.^2
potenz = 100  121  144
```

24.5 Matrizen

Eine m×n-Matrix ist ein rechteckiges Array von Zahlen aus m Zeilen und n Spalten. In Matlab werden die Einträge einer Matrix zeilenweise in derselben Syntax wie Vektoren eingegeben.

Beispiel:

`>> A = [4 6 8; 1 3 5]` entspricht der Matrix $A = \begin{matrix} 4 & 6 & 8 \\ 1 & 3 & 5 \end{matrix}$

Mit `>> size(A)` lässt sich die Dimension der Matrix abfragen. Große Matrizen lassen sich durch Aneinanderhängen aus mehreren kleineren Matrizen aufbauen.

Durch analoge Eingabe `>> A'` wie bei Vektoren werden Matrizen transponiert, d.h. die Zeilen und Spalten der Matrix werden vertauscht.

Matlab stellt einige Prototyp-Matrizen zur Verfügung.

Tabelle 24.6: Vordefinierte Matrizen

Matrix	Bedeutung
ones(*m*,*n*)	*m*×*n*-Matrix aus Eins-Einträgen
zeros(*m*,*n*)	*m*×*n*-Matrix aus Null-Einträgen
eye(*n*)	*n*×*n*-Einheitsmatrix
diag(*d*)	Diagonalmatrix aus Einträgen auf der Hauptdiagonalen entsprechend dem Vektor *d*

Eine bequeme Art der Eingabe einer Diagonalmatrix ist über die Eingabe eines Vektors *d* gegeben.

Beispiel:

`>> d = [9 -8 7]; D = diag(d)` ergibt $D = \begin{matrix} 9 & 0 & 0 \\ 0 & -8 & 0 \\ 0 & 0 & 7 \end{matrix}$

Für eine beliebig gegebene Matrix A gibt `diag(A)` die Diagonalelemente aus.

Der Eintrag der i-ten Zeile und j-ten Spalte einer Matrix A wird in Matlab mit `A(i,j)` adressiert. Ein Doppelpunkt kennzeichnet die komplette Zeile oder Spalte, abhängig davon, ob er als erster oder zweiter Eintrag in `A(i,j)` verwendet wird.

Beispiel: Für die Matrix D aus dem vorherigen Beispiel liefert

`>> D(3,3)` ergibt $ans = 7$

`>> D(:,3)` ergibt $ans = \begin{matrix} 0 \\ 0 \\ 3 \end{matrix}$

Ähnlich wie bei Vektoren gibt es auch bei Matrizen ein Punktprodukt .*, das die Einträge zweier Matrizen derselben Größe komponentenweise multipliziert.

Das Produkt einer Matrix mit einem Vektor ist nur für Spaltenvektoren definiert. Diese müssen dieselbe Anzahl Einträge haben wie die Matrix Spalten.

Beispiel:

```
>> A = [5 7 9; 1 -3 -7];x = [8; -4; 1];A*x
```

$ans = \begin{matrix} 21 \\ 13 \end{matrix}$

Das Produkt einer m×n-Matrix mit einer n×p-Matrik ergibt eine m×p-Matrix. In Tabelle 24.7 sind verschiedene Matrixoperationen und lineare Algebra-Funktionen zusammengestellt.

Tabelle 24.7: Matrixoperationen in Matlab

Funktion	Bedeutung
norm	Norm einer Matrix bzw. eines Vektors
rank	Anzahl linear unabhängiger Zeilen oder Spalten
det	Determinante der Matrix
trace	Spur einer Matrix, d.h. Summe der Diagonalelemente

Funktion	Bedeutung
orth	Orthogonalisierung
null	Nullraum
size	Größe einer Matrix
length	Länge eines Vektors
eig	Eigenwerte und Eigenvektoren
poly	Charakteristisches Polynom
inv	Inverse einer Matrix
chol	Cholesky-Faktorisierung
qr	Orthogonalzerlegung

24.6 Vergleiche und Kontrollstrukturen

24.6.1 Logische Operatoren

Matlab stellt `true` und `false` durch die integer-Werte `0` und `1` dar.

Tabelle 24.8: Logische Operatoren

Op	Bedeutung	Op	Bedeutung
`= =`	gleich	`~ =`	ungleich
`>`	größer	`<`	kleiner
`>=`	größer gleich	`<=`	kleiner gleich
`~`	nicht	`&`	und
`|`	oder		

Die logischen Operatoren lassen sich auf einzelne Elemente, aber auch auf Vektoren und Matrizen komponentenweise anwenden. Mit & und | können einzelne Operationen verbunden werden.

Beispiel:

```
>> x = pi; x ~= 3, x ~= pi
ans = 1
ans = 0
>> x = [-2  3.1  5; -1  0  1];
>> test = x > 3 & x < 4
test =
     0   1   0
     0   0   0
```


Als eine Anwendung können auf diese Weise durch Punkt-Multiplikation `x.*test` Matrixelemente herausgefiltert werden.

24.6.2 for-Schleife

Eine `for`-Schleife hat die generelle Struktur:

```
>> for Zähler
     Anweisungen
   end
```

Alle Befehle zwischen `for` und `end` werden so oft wiederholt wie der *`Zähler`* angibt. Dafür existieren zwei Möglichkeiten

- Als Iterator, z. B. *`Zähler`* `= 1:10`. Der Zähler nimmt dabei die Werte `1,2,3,...,10` an.
- Als Aufzählung, z. B. *`Zähler`* `= [17 5 12 6 37]`. Der Zähler nimmt nacheinander die Werte `17,5,` usw. an.

24.6.3 while -Schleife

Eine `while`-Schleife hat die generelle Struktur:

```
>> while test
     Anweisungen
end
```

Beispiel: Gesucht ist die größte Zahl n, die in der Summe $1^2+2^2+3^2+...+n^2$ verwendet werden kann, bei der die Summe kleiner als `100` ist.

```
>> S = 1; n = 1;
>> while S+(n+1)^2 < 100
  n = n+1; S = S+n^2;
  end
>> [n, S]
ans = 6   91
```

24.6.4 if...then...else...end -Anweisungen

Eine bedingte Anweisung hat die generelle Struktur:

```
if test1
    Anweisungen
elseif test2
    Anweisungen
end
```

24.7 Funktion m-files

Funktion `m-files` sind eine Kombination aus den Skript-`m-files` und mathematischen Funktionen. Die Hauptschritte zur Definition einer Matlab-Funktion sind:

1. Wahl eines Funktionsnamens, der nicht mit einem in Matlab vorhandenen Namen identisch ist. Der Name kann z.B. `name` sein. Es wird eine Datei `name.m` angelegt.
2. Die erste Zeile des Files muss das Format haben:
   `function [Liste der Ausgabedaten] =`
   `funktions_name(Liste der Eingabedaten)`
3. Dokumentation der Funktion: Kurze Beschreibung über den Nutzen der Funktion und wie sie eingesetzt werden kann. Die Kommentare werden mit % eingeleitet und auf diese Weise bei der Ausführung der Funktion nicht berücksichtigt.
4. Zuletzt wird der die Funktion definierende Code eingefügt. Dieser Teil sollte ebenfalls mit ausreichend Kommentaren versehen werden, damit auch andere Nutzer den Lösungsprozess nachvollziehen können.

Beispiel: Die Fläche eines Dreiecks mit Seitenlängen a,b und c ist gegeben durch $A = \sqrt{s(s-a)(s-b)(s-c)}$, wobei $s = (a+b+c)/2$ ist. Ein Matlab-Programm zur Bestimmung des Flächeninhaltes eines beliebigen Dreiecks lautet

```
function [A] = Flaeche(a,b,c)
% Programm zur Bestimmung des Flaecheninhaltes
% eines Dreiecks mit Seitenlaengen a,b, und c.
% Eingabeparameter sind die Seitenlaengen: a,b,c
% Ausgabeparameter ist der Flaecheninhalt: A
% Bedienung: Dreiecksflaeche = Flaeche(2,5,3)
s = (a+b+c)/2;
A = sqrt(s*(s-a)*(s-b)*(s-c));
```

Der Aufruf `>> help Flaeche` liest alle Kommentarzeilen aus dem File heraus und stellt somit eine Anleitung zur Bedienung dar. Um den Flächeninhalt eines speziellen Dreiecks mit Seitenlängen 10, 15 und 20 zu berechnen, wird die selbst geschriebene Funktion aufgerufen mit:

```
>> Dreiecksflaeche = Flaeche(10,15,20)
Dreiecksflaeche = 72.6184
```

Das Ergebnis der Berechnung wird der Variablen `Dreiecksflaeche` zugewiesen. Die Variable `s`, die in der Funktion verwendet wird, ist eine lokale Variable. Nur innerhalb der Funktionsumgebung kann auf sie zurückgegriffen werden. Ansonsten müsste sie als Ausgabeparameter mitübergeben werden.

24.8 Verschiedene Funktionen

Matlab verfügt über eine Liste nützlicher Funktionen, die ein Anwender direkt benutzen kann und nicht selber programmieren muss. Eine Auswahl von Funktionen ist in Tabelle 24.9 zusammengefasst.

Tabelle 24.9: Weitere Matlab-Funktionen

Funktion	Anwendung
sum(x) sum(A)	Summe der Komponenten eines Vektors, Summe der Matrixeinträge jeder Spalte einer Matrix A, es wird ein Zeilenvektor ausgegeben
min(x) max(x)	Bestimmung der kleinsten bzw. größten Komponente eines Vektors $\vec{x}$
min(A) max(A)	Ausgabe eines Zeilenvektors mit den jeweils kleinsten bzw. größten Einträgen in jeder Spalte der Matrix A
rand rand(m,n)	Ausgabe einer willkürlichen Zahl zwischen 0 und 1, Ausgabe einer $m \times n$-Matrix mit willkürlichen Einträgen, die zwischen 0 und 1 liegen (wiederholtes Ausführen ergibt unterschiedliche Ergebnisse)
find(*log. Bedingung*)	Ausgabe einer Liste von Stellen (Indizes), an denen die Elemente eines Vektors eine bestimmte logische Bedingung erfüllen
find(*log. Bedingung*)	Ausgabe einer Liste von Stellen (Indizes), an denen Matrixeinträge eine bestimmte logische Bedingung erfüllen

Beispiel:

```
>> x = [1 3 5]; summe = sum(x)
summe = 9
>>y = [1.3 -2.4  0  2.3]; maximum = max(y)
maximum = 2.3
>>z = rand, Z = rand(2,3)
z = 0.8654
Z = 0.3267  0.1592  0.5576
    0.7895  0.7120  0.2481
>> k = find(y>1.0)
k = 1  4
```

24.9 Zeitnahme

Matlab erlaubt eine Zeitnahme für einzelne Programmabschnitte in einem Code. Die entsprechenden Funktionen heißen `tic` und `toc`.

`tic` stellt eine Stoppuhr an und `toc` stoppt diese wieder und gibt die CPU-Zeit (Central Processor Unit) in Sekunden an. Die Zeitnehmung variiert abhängig von dem Computermodell.

Bibliografie und Webliografie

[UebKatz02] Überhuber, C., Katzenbeisser, S.: *Matlab 6.5: Eine Einführung*, Springer-Verlag 2002

[GramWer00] Gramlich, G., Werner, W.: *Numerische Mathematik mit Matlab. Eine Einführung für Naturwissenschaftler und Ingenieure.* dpunkt.verlag 2000

[Beucher02] Beucher, O.: *MATLAB und SIMULINK lernen.* Pearson Studium 2002

[Grupp02] Grupp, F., Grupp, F.: *MATLAB 6 für Ingenieure.* Oldenbourg 2002

[MathWorks] The MathWorks – Matlab User Documentation, The Language of Technical Computing http://www.mathworks.com

[IndUni] Indiana University, A General Overiew of Matlab, http://www.indiana.edu/~statmath/math/matlab/overview.html

25 ABAP

Horst Keller

25.1 Übersicht

ABAP® (Advanced Business Application Programming) ist die Programmiersprache der **SAP®**. Die SAP ist einer der weltweit führenden Anbieter für betriebswirtschaftliche Software [SAP]. Die meisten kommerziellen Anwendungen der SAP sind in ABAP geschrieben.

25.1.1 SAP-Basis und SAP Web Application Server

Voraussetzung für die Verwendung der Programmiersprache ABAP ist die Installation einer **SAP-Basis** oder eines **SAP Web Application Servers®**, der seit Release 6.10 die SAP-Basis umfasst. Jedes SAP-System basiert auf einer dreistufigen Client-Server-Architektur mit einer Präsentations-, einer Applikations- und einer Datenbankschicht.

- Die Präsentationsschicht ist auf die Einzelplatzrechner der einzelnen Anwender verteilt und stellt die Benutzungsoberfläche des SAP-Systems dar (SAP GUI® oder Web-Browser).
- Die Applikationsschicht besteht aus einem oder mehreren Applikationsservern. Die Applikationsschicht enthält die ABAP-Laufzeitumgebung, in der ABAP-Programme ausgeführt werden.
- Die Datenbankschicht besteht aus einem Datenbanksystem, in dem der Datenbestand eines SAP-Systems gespeichert ist.

Die **Hardware** eines SAP-Systems ist skalierbar. In der Regel gibt es für jede Schicht eigene Rechner. Für Übungs- und Testzwecke sind Mini-Versionen der SAP-Basis und des Web Application Servers erhältlich, die auf einem einzigen PC installierbar sind. MS-Windows-Versionen liegen den Büchern [Kel2000] und [Kel2002] bei. Linux-Versionen sind im *SAP Knowledge Shop* erhältlich [SAPshop].

25.1.2 Umfang und Einsatzgebiete der Sprache ABAP

ABAP ist die **Programmierschnittstelle** des SAP-(Web)-Applikationsservers. Seit der Erweiterung um **ABAP Objects** zu Release 4.6 ist ABAP mit allen Eigenschaften einer objektorientierten Programmier-