1 Grundkenntnisse von MATLAB ®

1.1 Bekanntschaft schließen mit MATLAB

1.1.1 Die Arbeitsoberfläche von MATLAB

Der Start der Applikation MATLAB[1] erfolgt abhängig vom Betriebssystem auf unterschiedliche Weise:

- in den Microsoft Windows Systemen durch einen Doppelklick auf das Icon von MATLAB oder durch Anwählen des Menüs oder Untermenüs in der Abfolge
  `Programme -> ... -> MATLAB`
- auf Unix/Linux-Systemen durch Eintippen des Kommandos `matlab &`
  (Starten von MATLAB mit dem Programmnamen und installieren als unabhängigen Prozess durch das Zeichen '`&`'). Nur bei gewissen Desktop Manager Programmen kann MATLAB ebenfalls durch Anwählen eines Icons gestartet werden.
  Achten Sie darauf, dass im Kommandomodus das Startkommando `matlab &` in Kleinbuchstaben eingegeben wird, obwohl Sie das Markenzeichen MATLAB immer in Großbuchstaben gesetzt sehen.

Darauf erscheint kurz das quadratische MATLAB-Markenzeichen, Banner genannt. Dieses Bild stellt eine komplizierte mathematische Fläche dreidimensional dar. Kurz danach zeigt sich das in einen linken und einen rechten Bereich unterteilte Hauptfenster von MATLAB.

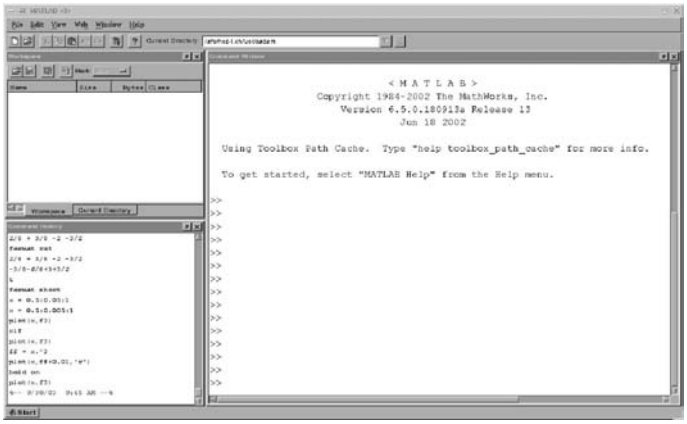

Abbildung 1.1: Das Hauptfenster von MATLAB mit normaler Unterteilung.

[1] ® MATLAB ist ein eingetragenes Warenzeichen von The MathWorks Inc., Natick, MA, USA.

MATLAB® und Mathematik kompetent einsetzen: Eine Einführung für Ingenieure und Naturwissenschaftler. Stefan Rudolf Alfred Adam

ISBN: 3-527-40618-2

Im linken Bereich sind zwei Teilfenster vertikal übereinander angeordnet. In beiden linken Teilfenstern können verschiedene Inhalte zur Anzeige angewählt werden. Im rechten, meist breiteren und nicht weiter unterteilten Bereich befindet sich das **Kommandofenster**, in dem der größte Teil der Arbeit mit MATLAB stattfindet. Der Hinweis an den Benutzer, dass das nächste Kommando eingegeben werden kann, der sogenannte „Prompt“ hat in MATLAB die Form von zwei nach rechts zeigenden Winkeln, also wie hier: >>

Ein weiteres Fenster dient zum Aufrufen der sehr gut ausgebauten Hilfefunktion von MATLAB. Die Teilfenster im linken Bereich neben dem Kommandofenster können je nach Vorwahl die Verzeichnisstruktur, die “Command History” und den “Workspace” anzeigen. Der “Workspace” ist die Übersicht über die momentan definierten Variablen und deren Struktur. Alle diese Fenster von MATLAB sowie die zusätzlich erscheinenden Fenster für die Ausgabe von Grafik kann man auch voneinander entkoppeln und beliebig auf der Arbeitsoberfläche platzieren. Die Auswahl für die Anzeige verschiedener Teilfenster findet sich im Hauptfenster mit dem Titel MATLAB unter dem Menü-Punkt “View”.

1.1.2 Zum Einstieg: Berechnungen mit einfachen Zahlen

Das Programm MATLAB ist ein äußerst vielseitiges Werkzeug für alle Arten von Berechnungen und Simulationen. Somit muss logischerweise das Rechnen mit einfachen Zahlen darin enthalten sein: Nachdem Sie MATLAB gestartet haben, können Sie also als Einstieg ein paar einfache Berechnungen eintippen wie z. B.:

`2 + 5 <ret>` (Das Drücken der <return>-Taste am Zeilenende wird im Weiteren stillschweigend vorausgesetzt)
Die Resultatausgabe versieht den Resultatwert mit dem Namen `ans`:

```
ans =
         7
2 * 5 + 7
ans =
        17
a = 2
a =
        2
b = 5
b =
        5
a*b
ans =
        10
```

`5^2` (dies ist die MATLAB-Schreibweise für 5^2, also 5 hoch 2 = 5*5)

```
ans =
        25
```

In diesen trivialen Beispielen stecken bereits einige Grundprinzipien von MATLAB, welche in der Anwendung immer wieder auftreten:

- MATLAB kennt noch **andere arithmetische Operatoren** als nur die elementaren '+' , '-' , '*' , '/' , hier haben wir mit '^' ein erstes Beispiel kennengelernt, weitere werden folgen.

- Die **Multiplikation** von zwei nebeneinanderstehenden Größen muss durch das **'*'-Zeichen** explizit verlangt werden. Bei Zahlen ist dies auch in der Mathematik erforderlich: 22 bedeutet etwas anderes als 2*2 (bzw. $2 \cdot 2$). Bei der Verwendung von Buchstaben als symbolische Zahlen darf man jedoch in der Mathematik die Kurzform 2ab schreiben statt 2*a*b.
  In MATLAB wird hingegen immer die ausführliche Schreibweise verlangt, also ist nur `2*a*b` eine richtige Eingabe.

- Alle Zwischen- und Schlussresultate in MATLAB brauchen einen **Namen** (= Variablennamen, Bezeichner), der später dazu dienen kann diese mathematische Größe in einer weiteren Berechnung wieder zu verwenden.
  Diese Namen sind in MATLAB empfindlich auf Groß- und Kleinschreibung (engl. "case sensitive"). Somit ist also `A` etwas anderes als `a` .
  Variablennamen müssen mit einem Buchstaben beginnen, dahinter dürfen auch Zahlen enthalten sein. Ungewohnt ist, dass keine Bindestriche in den Namen erlaubt sind (auch nicht bei Filenamen), weil MATLAB diese als Minuszeichen interpretiert.

- Für jede Berechnung kann der Programmbenutzer den Resultatnamen vorgeben durch die Formulierung `'name' =` wie z. B. in `c2 = a^2 + b^2` (gefolgt von `c = sqrt(c2)` für die Berechnung der Hypothenuse nach Pythagoras).

- Als Reaktion auf das Eintippen eines bereits definierten Variablennamens gibt MATLAB den unter diesem Namen gespeicherten Wert (bzw. die Werte bei Vektoren und Matrizen) auf dem Bildschirm aus.

- Falls der Benutzer keinen Resultatnamen angibt, setzt MATLAB immer automatisch denselben Allerweltsnamen `ans` ein (Kurzform von 'answer'). Dies erlaubt unmittelbar nach der Berechnung noch die Zuweisung an einen richtigen Namen z. B. durch `c2 = ans`.

- **Vorsicht!** Bei jeder Berechnung (Wertzuweisung) wird der Resultatname ohne Warnung mit dem neuen Wert überschrieben! Sie haben also keine andere Chance, das vorhergehende `ans` zurückzugewinnen, außer indem Sie die dazugehörige Berechnung von Grund auf neu ausführen! Ähnlich ergeht es Ihnen, wenn Sie zweimal hintereinander `a = 'Berechnung'` schreiben, dann ist das vorherige `a` überschrieben.
  Fazit: Wählen Sie nicht zu kurze und möglichst verschiedene Variablennamen!

1.1.3 Weitere einfache Berechnungsbeispiele

Einfache Taschenrechneranwendungen

Nachdem Sie MATLAB gestartet haben, tippen Sie nach Belieben einfache Berechnungen ein, in der Art wie Sie mit einem einfachen Taschenrechner umgehen würden. Lassen Sie Ihre Phantasie walten oder versuchen Sie es mit den folgenden Kommandoeingaben:

```
1+2+3+4
5+6+7+8+9+10
ans + 1+2+3+4
sum(1:10)
```

Taschenrechner mit Zehnerpotenzen

Der nachfolgende Befehl schaltet die Anzeige auf hohe Präzision und Exponentenschreibweise um, anschließend können Sie auch Berechnungen in größeren Bereichen der Zehnerpotenzen ausführen.

```
format long e
1/1000
ans^4
1/ans
sqrt(ans)
log(200*ans)
```

Taschenrechner mit Variablennamen

Den Komfort, dass man die Ausgangszahlenwerte und die Zwischenresultate mit Namen belegen kann, bieten die einfachen Taschenrechner schon nicht mehr.

```
a = 20
b = 2
p = a*b
s = a+b
sq = (a+b)^2
a^2 + 2*a*b + b^2
ans - sq
```

Hier ist der Hinweis am Platz, dass MATLAB kein Programm für die Manipulation von symbolischen algebraischen Ausdrücken ist, sondern eine Umgebung zur Ausführung von Berechnungen. Damit wird die Fehlermeldung verständlich, die auftritt, wenn man in den Berechnungsausdrücken eine Variable einsetzt, der vorher noch kein Zahlenwert zugeordnet wurde, die also zu diesem Zeitpunkt noch nicht definiert ist.
Das Zusatzpaket “symbolic algebra” erlaubt die Einführung von unbekannten Variablen und die Manipulation von Formeln, im Basisprogramm ist diese Option nicht enthalten. In Kapitel 8 wird das Arbeiten mit dem Zusatzpaket “symbolic algebra” behandelt.

Übungen zur expliziten Schreibweise von Produkten

Die Tatsache, dass in MATLAB das Multiplikationszeichen immer explizit geschrieben werden muss, braucht für den Anfang ein wenig Übung. Dies ist eine glänzende Gelegenheit, die binomischen Formeln aufzufrischen. Können Sie sich noch an das Pascal'sche Dreieck erinnern? Ergänzen Sie die fehlenden Terme bei den Fragezeichen!

```
a = 40
b = 3
p3 = (a-b)^3
q3 = a^3 - 3*a^2*b + 3*a*b^2 - b^3
p4 = (a-b)^4
q4 = a^4 - 4*a^3*b + 6*a^2*b^2 ..?
p5 = (a-b)^5
q5 = a^5 - 5*a^4*b + 10*a^3*b^2 ..?
```

Ausblick: mit fortgeschrittenen Kenntnissen von MATLAB könnte man versuchen, den Anfang des Pascal'schen Dreiecks automatisch auszudrucken.

1.1.4 Basiswissen zu MATLAB

Ein weiteres Anwendungsbeispiel soll weitere Grundprinzipien von MATLAB aufzeigen. Testen Sie die Reaktion von MATLAB auf die folgenden Eingabezeilen: (Alle Zeilen sollen mit einem Semikolon abgeschlossen werden, außer der letzten, welche die Anzeige des Inhaltes der Variablen `f` anfordert.)

```
r = 5 ;
s = 3 ;
f = 0.1/r^s ;
f
```

Zu den Reaktionen des Programms MATLAB auf diese Eingabe ergeben sich die folgenden Erklärungen:

- Ein **Semikolon am Zeilenende** unterdrückt die Anzeige des Zwischenresultates auf dem Bildschirm, ohne die Ausführung der Berechnung zu beeinflussen.
- MATLAB kennt im Gegensatz zu den meisten Programmiersprachen keine Unterscheidung zwischen ganzzahligen und gebrochenen Zahlenwerten (in diesem Punkt kann man daher auch keine Fehler machen); alle Zahlen sind doppelt genaue Gleitkommazahlen (vom Grundtyp 'real', 'float' oder 'double').
- In MATLAB müssen die Variablen nicht deklariert werden: Bei jedem Auftreten eines neuen Namens in einer Wertzuweisung wird dieser automatisch in die Liste der Variablen eingefügt.
- Alle bisherigen **Ausgabeformate** sind in der gleichen Standard-Darstellung erfolgt. (z. B. `25.0000` oder `0.0008`).

– Die Anzeige der Resultate kann mit dem Befehl `'format'` gesteuert werden. Die wichtigsten Optionen von `'format'` sind:
`format short` (der Standard)
`format short e` (Exponentialschreibweise)
`format short g` (Exponentialschreibweise nur wenn nötig)
`format long , format long e , format long g`
und für kommerzielle Anwendungen:
`format bank` (zwei Stellen nach dem Dezimalpunkt: `500.25`)
Da Sie jederzeit im Hilfesystem nachschauen können, genügt es, wenn Sie sich die Gedankenverbindung: **Anzeigeformat** $->$ **'format'** merken, ohne alle Optionen auswendig zu lernen.
Mit den Befehlen `format compact` und `format loose` kann man zwischen dem engen und weiten Zeilenschaltungsmodus hin- und herschalten.

Runde Klammern: Für das Formulieren von komplizierteren Ausdrücken gelten die Regeln zur Priorität der Operatoren wie in der Mathematik und ebenso kann man mit **(ausschließlich runden) Klammern** die Operatorenabfolge beliebig festlegen. Dazu ein Beispiel:

```
a = 100
b = 4
a2mb2 = (a+b)*(a-b)
```

Eckige Klammern: Die eckigen Klammern werden in MATLAB für die Definition von Matrizen und Vektoren gebraucht. Alle Zahlen die zu einer Matrix oder einem Vektor gehören, stehen zwischen einem Paar von öffnenden und schließenden eckigen Klammern. Im Innern dieser Klammern werden Elemente, die in derselben Zeile nebeneinander stehen, durch Leerzeichen oder Kommata getrennt. Ein Fortschreiten auf die nachfolgende Zeile wird bei der Matrix- oder Vektoreingabe durch ein Semikolon angezeigt.
Die zwei Beispiele der MATLAB-Eingabe eines dreidimensionalen Spaltenvektors
`v = [2; 0.5; 4]` entsprechend der mathematischen Formulierung $v = \begin{pmatrix} 2 \\ 0.5 \\ 4 \end{pmatrix}$
und einer 2x2 Matrix
`M = [1 4; 3.5 2]` entsprechend der mathematischen Formulierung
$M = \begin{pmatrix} 1 & 4 \\ 3.5 & 2 \end{pmatrix}$ sollen dies illustrieren.

Fortsetzung einer Zeile: In MATLAB ist der Zeilenumbruch von Bedeutung, jeder Befehl steht in einer einzigen Zeile. Falls Ihnen bei der Eingabe von komplexen Ausdrücken, von Matrizen oder von Funktionsaufrufen der Platz in einer einzigen Zeile zu knapp werden sollte, so ...
können Sie die **Fortsetzung** auf der nächsten Zeile durch die Angabe von **drei aufeinander folgenden Punkten** (wie oben) verlangen.

Mehrere Befehle in derselben Zeile: Das Zeilenende bestimmt den natürlichen Abschluss eines Befehls. Falls man mehrere Befehle auf dieselbe Zeile schreiben möchte, so trennt man

diese mit Kommata oder Semikola. Diese Trennzeichen bestimmen dann die einzelnen Befehlsabschlüsse. Wie beim Zeilenende unterdrückt dabei ein Semikolon die Resultatausgabe, ein Komma lässt diese zu.

Vordefinierte Konstanten

Verschiedene Konstanten sind in MATLAB vordefiniert, so z. B. die Zahl π unter dem Namen `pi`. Achten Sie auf die Groß- und Kleinschreibung!

```
r = 230
U = r*pi
A = r^2 *pi
Vs = 4*pi/3*r^3
As = 4*pi*r^2
```

Die Zahl π könnte man auch durch einen Funktionsaufruf erhalten, indem man die Tatsache verwendet, dass `atan(1.0)` = $\pi/4$ ist. Dies ist ein nützlicher Hinweis bei der Anwendung von anderen Programmiersprachen, welche die Konstante π nicht als vordefinierte Konstante enthalten.

Eine weitere in MATLAB vordefinierte Konstante ist die Imaginäre Einheit 'i', welche man zum Definieren von imaginären und komplexen Zahlen (wie z. B. $a + i * b$) braucht. In der Elektrotechnik hat sich allerdings die Verwendung des Buchstabens 'i' (odar auch 'I') für Ströme eingebürgert. Deshalb verwendet man in diesem Fachgebiet meist den Buchstaben 'j' für die Imaginäre Einheit. Man schreibt also $a + j * b$.

Beide Namen, also sowohl 'i', als auch 'j', sind in MATLAB als Imaginäre Einheit vordefiniert.

Dass man dem Namen 'pi' selbst einen anderen Wert zuweist und damit die vordefinierte Konstante unbrauchbar macht, dürfte eher selten vorkommen. Bei Schleifenkonstruktionen in Programmen kann es einem aber schon einmal passieren, dass man 'i' oder 'j' als Schleifenvariable benutzt. Solange die Berechnungen alle reell sind, wird dies auch ohne Folgen bleiben. Wenn man den Fehler bemerkt, so kann man mit `clear i` und Ändern der Schleife wieder den ursprünglichen Zustand herstellen. Es wird hier trotzdem empfohlen, generell auf die Variablen i und j in MATLAB-Programmen zu verzichten. Man kann ja beim Unsetzen einer Formel, in der zwecks kompakter Schreibweise i oder j vorkommen, stattdessen im Programm ii und jj verwenden.

Eine andere vordefinierte Konstante ist z. B. `eps`, die relative Rechengenauigkeit des verwendeten Computers, das ist die kleinste Änderung der Zahl 1.0000, so dass 1.000 + `eps` noch als verschieden von 1.000 registriert wird.

Übung 11–1: Die Euler'sche Konstante e
Wie kann man die Funktion `exp(x)` $= e^x$ verwenden, um die Euler'sche Konstante `e` (siehe Seite 92) zu erhalten? Diese ist in MATLAB nämlich nicht vordefiniert.

M-Files

Die reiche Vielfalt der Anwendungsmöglichkeiten von MATLAB beruht darauf, dass komplexe Berechnungsaufgaben auf einfachste Weise in Teilaufgaben zerlegt werden können.

Die zwei in MATLAB vorhandenen Zerlegungsmechanismen sind die Skript-M-Files und die Funktions-M-Files.

Bei einem Skript-M-File werden einfach alle MATLAB-Befehle zum Erledigen einer Teilaufgabe in ein File mit dem Namenzusatz `'.m'` geschrieben. Die Eingabe des entsprechenden Namens in MATLAB lässt dann diese Befehlsfolge ablaufen. Man nennt die Funktionalität in der Informatik auch Makroaufruf.

Die Verwendung von Funktions-M-Files ist eng damit verwandt. Beim Aufruf werden der Funktion allerdings Parameter übergeben, die in runden Klammern an den Funktionsnamen angehängt werden. In den meisten Fällen haben Funktionen auch einen oder sogar mehrere Rückgabeparameter; dies ist aber nicht unbedingt erforderlich. Ein großer Teil der vielfältigen, in MATLAB zur Verfügung stehenden Befehle beruht auf solchen Funktions-M-Files. MATLAB ist also zu einem großen Teil in MATLAB geschrieben. Durch Erstellen von eigenen Funktionen kann die Spannweite durch den Anwender selbst beliebig erweitert werden. Je ein Beispiel für einen Funktionsaufruf ohne, mit einem und mit mehreren Rückgabeparametern wird im Folgenden gezeigt:

`plot(x,y)`	(hier wäre ein Rückgabeparameter möglich, dieser wird aber nur in Spezialfällen verwendet.)
`M = zeros(4,6)`	(erzeugt eine 4x6 Matrix mit lauter Nullen.)
`[n,m] = size(M)`	(Das Umschließen der Liste von mehreren Rückgabeparametern ist eine weitere Anwendung von eckigen Klammern.)

1.1.5 Hintergrundinformation und Hilfefunktionen

Workspace – aktive Variablen: Weitere nützliche Tips für das Arbeiten mit MATLAB sind die Hinweise auf die Auskunftsfunktionen `who` und `whos`, auf die Möglichkeit jede Variable durch Eintippen ihres Namens auszugeben, sowie auf die selektive bzw. totale Löschfunktion `clear 'variable'` bzw. `clear`.

- Die Funktion `who` zeigt die Namen aller aktuell aktiven (jemals benutzten und noch nicht gelöschten) Variablen an. (Der Funktionsname 'who' ist eine Anlehnung an das Lexikon mit persönlichen Daten von gesellschaftlich bedeutsamen Personen "Who is Who".)

- Die Funktion `whos` ist eine kleine Erweiterung der Funktion `who` im Sinne von who-size, bei der neben den Namen auch die Dimensionen der mit den Namen verbundenen mathematischen Größen angezeigt werden, eine Information, welche vor allem in Zusammenhang mit dem Rechnen mit Vektoren und Matrizen nützlich ist.

- Die Anzeige der mit einem Variablennamen verbundenen Zahl (bzw. der Zahlen) erfolgt, wenn man den Variablennamen selbst eintippt. Dieselbe Wirkung, jedoch ohne erneute Ausgabe des Namens, hat die Funktion `disp()`. Diese kann auch zur Ausgabe eines Textes wie bei `disp('Schlussresultat:')` verwendet werden.

- Eine Variable, z. B. `a` kann mit dem Befehl `clear a` gelöscht werden, der Befehl `clear` ohne nachfolgene Angabe von Namen löscht **alle** bisher verwendeten Variablennamen (d. h. den gesamten "workspace").

Hilfefunktionen: Zur großen Erleichterung für alle Anfänger, aber ebenso für die Fortgeschrittenen gibt es in MATLAB gut ausgebaute Hilfefunktionen und Demonstrationsbeipiele von ganz verschiedenen Schwierigkeitsgraden. Diese sind über die Kommandos `help` , `help stichwort` bzw. `help kategorie/stichwort` , über `lookfor allgemeinbegriff` und `demo` aufrufbar. Im speziellen Hilfefenster, in der Menüleiste unter Help, sowie über die Schaltfläche mit dem Fragezeichen existieren weitere Zugangsmöglichkeiten.

- Das Kommando `help` und die Varianten `help stichwort` und `help kategorie/stichwort` sind die schnellsten Nachfragemöglichkeiten, die oft zum Nachschauen des genauen Aufrufformates (der Signatur) benutzt werden, wenn man den Namen der Funktion kennt.
  Achtung! In den **help**-Texten werden alle Funktionsnamen mit **Großbuchstaben** angezeigt, für die **Anwendung** der Funktionen müssen diese Namen aber **klein** geschrieben werden.
- Der Befehl `lookfor allgemeinbegriff` gibt Hinweise auf alle Stellen in den Hilfedateien, in welchen der Allgemeinbegriff vorkommt. Dies kann enorm nützlich sein, wenn man die Prozedurnamen nicht kennt, sondern nur deren prinzipielle Funktion.
- Nach dem Eintippen des Befehls `doc` oder `doc stichwort` erscheint die ausführliche Dokumentation in einem unabhängigen Hilfefenster. Hier gibt es eine große Vielfalt von Suchmöglichkeiten in den Verzeichnisbäumen und Volltextsuche in den Erklärungstexten.
- Auf die Eingabe von `demo` erscheint ein separates Demo-Frame mit einem mehrstufigen Menu, über welches man Demonstrationsprogramme aus einer Serie von vielen verschiedenartigen MATLAB Kommandosequenzen auswählen kann. Deren Arbeitsweise wird auf dem Bildschirm vorgeführt und das zugehörige Programm kann angezeigt werden.

Protokollfile und Command History: Eine nützliche Hilfe, vor allem bei älteren Versionen von MATLAB, bietet der Befehl `diary 'filename'`, mit dem Sie alle auf diesen Befehl folgenden MATLAB-Befehle in das mit dem von Ihnen gewählten Namen `filename` bezeichnete Protokollfile eintragen lassen. Dies gilt solange, bis Sie das Protokollieren mit dem Abschaltbefehl `diary off` beenden. In den neueren Versionen von MATLAB liefert allerdings das "command history" Fenster dieselbe Hilfe in noch kompakterer Form.

Am Anfang des Arbeitens mit MATLAB empfiehlt es sich besonders, die in letzter Zeit eingegebenen Befehle gelegentlich in einer Rückblende zu inspizieren. Damit kann man sich an diverse kleine Eingabefehler nochmals erinnern, mit dem Zweck die häufigsten Fehler in Zukunft zu vermeiden.

1.1.6 Datenaustausch mit Files

Für die Anwendung auf größere technische Probleme ist die interaktive Eingabe der Daten im Arbeitsfenster von MATLAB ungeeignet. Nach jeder Unterbrechung eines Arbeitsgangs müssten die Daten wieder neu eingegeben werden, was ein großer, unnötiger Aufwand wäre. Die interaktive Eingabe ist auch viel anfälliger auf Fehler als das Einlesen aus einem Datenfile. Deshalb gibt es in MATLAB verschiedene Möglichkeiten, die in der Berechnung verwendeten Daten aus Dateien einzulesen und die Resultate in Dateien zu speichern.

Tabellen einlesen und abspeichern

Die einfachste Art, Zahlenwerte von Datenfiles in den Arbeitsbereich von MATLAB zu transferieren ist das Einlesen einer einfachen Tabelle von lauter Zahlen mit dem Befehl

```
load filename.ext@load filename.ext
```

Mit dem Befehl '`load filename.ext`' wird ein File, das eine Anzahl Zeilen enthält, in denen jeweils dieselbe Anzahl von Zahlen stehen, einer Matrix zugeordnet. Der Name der Matrix, welche nach Ausführung des 'load'-Befehles existiert, ist genau der Filename, aber ohne dessen Namenserweiterung. Die Anzahl der Zeilen im File entspricht der Zeilenzahl, die Anzahl Zahlen pro Zeile der Spaltenzahl der mit 'load' definierten Matrix.

Das Pendant zum Einlesen einer einfachen Tabelle ist das Schreiben einer einzelnen Matrix z. B. mit dem internen Namen 'A' auf ein File. Dies erfolgt mit dem Befehl:

```
save -ascii filename.ext A   bzw.
save -ascii -double filename.ext A
```

Die erste Form ergibt eine Fileausgabe mit acht Dezimalstellen Genauigkeit, die zweite Form schreibt die Zahl mit voller Speichergenauigkeit in die textartige Fileausgabe.

Dasselbe Befehlspaar 'save' und 'load' kann auch zum Abspeichern und Wiedereinlesen von Variablen des aktuellen 'workspace' im Binärformat verwendet werden. Bei 'save' entfällt dann selbstverständlich die Option "-ascii". Die Namen solcher 'workspace' 'save'-Files sollten möglichst ohne Dateinamenserweiterung angegeben werden, dann wählt MATLAB automatisch die Erweiterung '.mat' für das MATLAB- und maschinenspezifische Binärfile-Format. Ganz ohne Angabe eines Filenamens wird bei 'save' und 'load' ein File mit dem Namen '`matlab.mat`' erzeugt, bzw. gelesen. Bei Verwendung des binären Formates ist aber zu beachten, dass solche Files nicht zwischen verschiedenen Computersystemen ausgetauscht werden können.

Formatiertes Einlesen und Abspeichern

In ein File, das mit dem Befehl `filehandle = fopen('filename.ext', 'w')` eröffnet wurde, können mit dem Befehl

```
fprintf(filehandle, ' formatstring' , var1, var2, ...)
```

in der von der Programmiersprache C her gewohnten Art Daten und Texte formatiert ausgegeben werden. Das fertige File ist mit `fclose(filehandle)` abzuschließen.

Die entsprechenden Sequenzen zum formatierten Einlesen eines Files sind:

```
filehandle = fopen('filename.ext', 'r')
fscanf(filehandle, ' formatstring' , var1, var2, ...)
...
fclose(filehandle) .
```

Die Arbeitsweise dieser formatierten Ausgabe in Kürze:
Die zwei Bestandteile 'formatstring' (eine Zeichenkette zur Festlegung der Formatierung) und die Liste der Variablen werden bei der Ausgabe nach dem Reißverschlussprinzip ineinander gefügt und ergeben so die ins File übertragene Zeichenkette.

Ein paar ganz einfache Beispiele mit dazugehörigen Erklärungen:

```
fprintf(fhdl, ' Matrixdimensionen: %d x %d \n', nzeilen, nspalten )
```

Im Format-String sind alle direkt auszugebenden Zeichen, inklusive Leerzeichen enthalten, sowie die mit dem %-Zeichen beginnenden Vorschriften, wie die Variablen aus der anschließenden Liste im File dargestellt werden sollen (hier `%d` für die Ausgabe von ganzzahligen Werten). Die Konstruktion `\n` steht für 'newline', bewirkt also einen Zeilenvorschub am Schluss der Ausgabezeile. Die nach dem Reißverschlussprinzip zusammengefügte Ausgabezeile sieht für dieses Beispiel wie folgt aus:

```
Matrixdimensionen: 4x6
```

Zur Strukturierung des Ausgabefiles sind auch reine Leerzeilen nützlich:

```
fprintf(fhdl, '\n'
```

Das nächste Beispiel gibt dieselbe Zahl in verschiedenen Formaten aus:

```
fprintf(fhdl,'Gleitkomma- %15.6f und Exponentialform %15.8e \n',wert, wert )
```

Die zugehörige Ausgabezeile im File hat dann dieses Aussehen:

```
Gleitkomma- 1835.217000 und Exponentialform 1.83521700e+03
```

Zwischen dem %-Zeichen und dem charakteristischen Buchstaben für den Typ des Formates (d, i, f, g, e, c, s etc.) können noch ein oder zwei Zahlen stehen (bei zwei Zahlen durch einen Punkt getrennt), welche die gesamte Anzahl Ziffern, bzw. die Anzahl Ziffern nach dem Dezimalpunkt vorgeben. Die charakteristischen Buchstaben d, i, u, o, x, X stehen für ganzzahlige Werte (o=Oktal-, x und X Hexadezimalausgabe), die Buchstaben f, e, E, g, G für real oder double precision (Standard in MATLAB) und c für Einzelzeichen und s für Zeichenketten. Weitere Details erfahren sie mit `help fprintf`.

Ein weiteres Beispiel zeigt nochmals das Reißverschlussprinzipvon `fprintf()`. In diesem Fall werden die Elemente einer Matrix so auf das File geschrieben, wie sie zur Eingabe der Matrix in MATLAB eingetippt werden müssten.

```
fprintf(fhdl,' M = [ %8.4f %8.4f; %8.4f %8.4f ] \n',...
M(1,1), M(1,2), M(2,1), M(2,2) )
```

Wiederum gibt es für jeden ausgegebenen Zahlenwert aus der Matrix M je eine Formatierungsvorschrift im Formatstring (hier alle '%8.4f'). Diese Plätze werden der Reihe nach mit den Werten in der nachfolgenden Variablenliste gefüllt.
Dies ergibt im Ausgabefile folgende Zeile:

```
 M = [ 11.0000 12.0000; 21.0000 22.0000 ]
```

Für eine gute Dokumentation der durchgeführten Berechnungen in einem ausdruckbaren File ist auch das Einfügen des Datums in die Ausgabe notwendig. Dazu gibt es die MATLAB-Funktion `date`. Die Befehlskombination

```
datstr = date ;
fprintf(fhdl, ' %s \n',datstr)
```

leistet dies.

1.2 Mathematische Grundlagen der Matrizenrechnung

1.2.1 Definitionen zur Matrizenrechnung

Definition einer Matrix

Matrizen sind mathematische Objekte mit denen wir im täglichen Leben recht selten in Berührung kommen. Darum erscheint uns im ersten Moment das Gebiet der Matrizenrechnung als fremdartig und abstrakt. Da jedoch eine große Vielfalt von konkreten Berechnungen auf der Mathematik mit Matrizen aufbaut, lohnt es sich, die anfängliche Scheu zu überwinden und die Welt der Matrizen näher kennenzulernen.

Matrizen sind Zusammenfassungen von gewöhnlichen Zahlen, angeordnet in einem Rechteckschema wie z. B.

$$M = \begin{pmatrix} 1 & 2 & 0 & 3 \\ 7 & 1 & 1 & 0 \\ 5 & 2 & 0 & 6 \end{pmatrix}$$

Jede Gruppe von nebeneinander stehenden Zahlen bildet eine Zeile der Matrix. Das Beispiel enthält also drei Zeilen. Übereinander angeordnete Zahlen bilden je eine Spalte, in diesem Beispiel sind vier Spalten zu sehen.

Matrizen gehören damit in die Klasse der **Verbundvariablen**, wie sie in der Informatik häufig verwendet werden: Unter einem einzigen Namen (hier M) werden mehrere zusammengehörige Dinge (hier die 12 Zahlen) zusammengefasst. Ein anderes Beispiel für eine Verbundvariable in der Informatik ist eine Postadresse mit den Bestandteilen Vorname, Nachname, Straße, Postleitzahl, Ort, etc. Matrizen sind im Vergleich zu anderen Verbundvariablen relativ einfach strukturiert: alle Einträge in diesem Verbund sind von gleicher Art, nämlich Zahlen, und die Anordnung erfolgt in einem festen Rechteckschema. In dieser einfachen Form werden Verbundvariablen auch **Feldvariablen** genannt (englisch: **array**, Mehrzahl **arrays**).

Untersuchen wir kurz den gesamten Informationsgehalt der obigen Matrix M: Da sind einmal die 12 Zahlen selbst, von jeder ist aber ihr Platz in der Matrix von Bedeutung (nicht nur, dass sie in der Matrix vorkommt, wie das bei Elementen einer Menge der Fall wäre). Darüber hinaus gehören aber auch noch die Dimensionen (Anzahl Zeilen und Anzahl Spalten) zur vollständigen Definition der Matrix M. In diesem speziellen Fall mit 12 Elementen wären für die beiden Dimensionszahlen die Varianten 2x6, 3x4, 4x3 und 6x2 möglich. Dazu kommen zwei Anordnungsmöglichkeiten als eine einzige Zeile oder eine einzige Spalte. Diese Grenzfälle von Matrizen mit Anordnungen von Zahlen entlang einer einzigen Linie nennt man Vektoren. Den Dimensionszahlen 1x12 entspricht ein Zeilenvektor, den Dimensionszahlen 12x1 ein Spaltenvektor.

Unter dem einen Variablennamen einer Matrix sind also die zwei Dimensionszahlen und alle darin enthaltenen Zahlen mitsamt ihrer Anordnung zusammengefasst.

Wie alle Typen von Verbundvariablen haben auch Matrizen eine eigene Bedeutung, die auf der Zusammenfassung der enthaltenen Informationsbestandteile beruht und über die reine Auflistung der enthaltenen Teilinformation hinausgeht. Auch für Matrizen gilt also der philosophische Satz, dass das Ganze mehr ist als die Summe seiner Teile.

Die in der Matrix enthaltenen Zahlen werden Elemente der Matrix genannt. Diese können ganzzahlig, positiv, negativ oder null sein, aber auch beliebige reelle und auch komplexe

Zahlenwerte annehmen. Falls eine Matrix komplexe Zahlen enthält, spricht man von einer komplexen Matrix.

Im Sinn der objektorientierten Programmierung gehört eigentlich zu jeder Matrix neben den Daten (den enthaltenen Zahlen und den Dimensionen) noch das gesamte mathematische Konzept, wie man mit Matrizen umgeht. Die Klasse „Matrix“ ist erst vollständig bekannt, wenn auch die zusätzlichen Kenntnisse verfügbar sind, welche Methoden für Matrizen existieren und wie diese Methoden arbeiten.

Das Anordnungsprinzip in einer Matrix

Im täglichen Leben spielen Matrizen nur eine geringe Rolle, das in einer Matrix verwendete Anordnungsprinzip ist aber trotzdem im menschlichen Geist tief verankert. Die Strukturierung einer größeren Zahl von Dingen durch Platzierung in Reihen und Spalten kommt auch außerhalb der Matrizenrechnung immer wieder vor (so z. B. im Städtebau oder in militärischen Marschformationen).

Sehr eindrucksvoll wird einem vor Augen geführt, wie grundlegend dieses Prinzip für den menschlichen Geist sein muss, wenn man die Anordnungen der Steinblöcke (“alignement”) bei Carnac, Frankreich betrachtet. Schon vor mehr als 5000 Jahren haben die Menschen das gleiche Anordnungsprinzip für die großen Steinblöcke angewandt, wie es heute in den Matrizen für Zahlen benutzt wird!

Abbildung 1.2: „Alignements de Kermanio“ bei Carnac, Frankreich.

Definition der wichtigsten Begriffe der Matrizenrechnung

Die in einer Matrix nebeneinander stehenden Zahlen bilden je eine **Zeile**, das englische Wort dafür ist **row**. Die untereinander stehenden Zahlen bilden je eine **Spalte** der Matrix. Gelegentlich findet man auch den Ausdruck **Kolonne** dafür, dies entspricht dem englischen Begriff **column** (lateinisch = Säule).

Wenn von einer Matrix angegeben wird, sie sei eine **nxm**-Matrix, dann hat sie **n Zeilen** und **m Spalten**. Dies ist eine Konvention, die man sich merken muss, ohne dass es eine weitere Begründung dafür gibt: die erste bei den **Dimensionszahlen** genannte Zahl entspricht der Zeilenzahl (= der Höhe), die zweite der Spaltenzahl (= der Breite der Matrix).

Eine Matrix mit derselben Anzahl von Zeilen und Spalten heißt **quadratisch**. Quadratische Matrizen sind eine wichtige Teilmenge der Gesamtheit von möglichen Matrizen, weil die quadratischen Matrizen zu einer festen Dimension unter sich eine Algebra bilden. Zur Unterscheidung verwendet man bei allgemeinen, nicht quadratischen Matrizen die Bezeichnung **rechteckige** Matrix oder spezifischer **hohe** Matrix ($n > m$), bzw. **breite** Matrix ($n < m$).

In einer quadratischen Matrix bilden die Zahlen, welche auf der Verbindungsgeraden zwischen den Ecken links oben und rechts unten liegen, die **Diagonale** der Matrix. Eine Matrix, welche **nur** auf der Diagonalen Zahlenwerte verschieden von null aufweist, nennt man **Diagonalmatrix**:

$$\text{Beispiel einer Diagonalmatrix } D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 6 \end{pmatrix}$$

Die zur Diagonalen parallelen Linien, welche gegen oben (rechts), bzw. gegen unten (links) direkt an die Diagonale anschließen, heißen **Nebendiagonalen**. Dies muss besonders erwähnt werden, weil in gewissen Lehrbüchern dieser Begriff fälschlicherweise für die (in der linearen Algebra bedeutungslose) Linie von Matrixelementen von links unten nach rechts oben verwendet wird. Dafür könnte man eventuell die Bezeichnungen „Gegendiagonale“ oder „Antidiagonale verwenden, zwei mögliche Umschreibungen, die aber keine Fachausdrücke sind.

Gelegentlich wird auch bei Rechtecksmatrizen von der Diagonalen gesprochen; diese beginnt auch in diesen Fällen immer links oben und endet bei hohen Matrizen am rechten Rand und bei breiten am unteren.

Eine quadratische Matrix, welche nur auf der Diagonalen und oberhalb, bzw. rechts davon, Elemente aufweist, die von null verschieden sind, wird **obere Dreiecksmatrix** genannt oder auch **Rechts-Dreiecksmatrix**. Dementsprechend hat eine **untere Dreiecksmatrix**, auch **Links-Dreiecksmatrix** genannt, die von null verschiedenen Elemente nur auf der Diagonalen und unterhalb (links) davon. Bei beiden müssen also alle Elemente auf der anderen Seite der Diagonalen null sein; auf der mit dem Namen bezeichneten Seite können beliebig Null-Elemente oder von null verschiedene Elemente auftreten.
Beispiele von unteren (Links-) und oberen (Rechts-) Dreiecksmatrizen:

$$L = \begin{pmatrix} 11 & 0 & 0 \\ 21 & 22 & 0 \\ 31 & 32 & 33 \end{pmatrix} \qquad R = \begin{pmatrix} 11 & 12 & 13 \\ 0 & 22 & 23 \\ 0 & 0 & 33 \end{pmatrix}$$

Aus dieser Definition folgt, dass eine Matrix, welche sowohl die Bedingung für eine Links- als auch für eine Rechts-Dreiecksmatrix erfüllt, eine Diagonalmatrix sein muss.

Ebenfalls von großer Bedeutung sind **Vektoren**, die man natürlich als spezielle Matrizen betrachten kann, die in einer Richtung die Dimension 1 aufweisen. Die beiden Varianten davon sind **Zeilenvektoren**, die den 1xn Matrizen entsprechen, und die aus der Vektorgeometrie wohlbekannten **Spaltenvektoren**, meist einfach **Vektoren** genannt, die den nx1 -Matrizen entsprechen.

$$z = \begin{pmatrix} z_1 & z_2 & z_3 \end{pmatrix} \qquad v = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Der letzte hier erwähnte Spezialfall sind die **1x1-Matrizen**, die nur eine Zahl enthalten und in MATLAB automatisch wie einfache normale Zahlen behandelt werden. Sobald bei einer Berechnung eine 1x1-Matrix entsteht, verliert diese sofort ihre Matrixeigenschaften und wird weiterbehandelt wie eine normale Zahl. Eine solche einzelne Zahl wird zwecks Unterscheidung von den Matrizen und den Vektoren auch **Skalar** genannt. Die automatische Einstufung von 1x1 Matrizen als Skalare in MATLAB zeigt sich daran, dass deren Addition und Subtraktion sowie deren Multiplikation mit Matrizen beliebiger Dimension erlaubt ist.

Übung 12–1 Die Fülle von neuen Begriffen soll sogleich mit ein paar einfachen Aufgaben gefestigt werden:

- Wieviele Freiheitsgrade (frei wählbare Zahlen) hat eine allgemeine nxm Matrix?
- Wieviele Freiheitsgrade hat eine nxn Diagonalmatrix?
- Wieviele Freiheitsgrade hat eine obere nxn Dreiecksmatrix?
- Welche Matrix erfüllt die Bedingungen einer oberen Dreiecksmatrix und gleichzeitig diejenigen einer unteren Dreiecksmatrix?
- Stellt ein gewöhnlicher Vektor eine „hohe“ oder eine „breite“ Matrix dar?
- Wieviele Elemente enthalten die beiden Nebendiagonalen einer nxn-Matrix?
- Vervollständigen Sie die englische Dimensionsdeklaration einer Matrix:
  “The size of a matrix is indicated by the number of . . . followed by the number of . . . . (with an ’x’ in between)!”

1.2.2 Indizieren der Matrixelemente

Falls man nicht nur die Matrix als Ganzes behandelt, sondern auf die einzelnen Elemente zugreifen will, benutzt man zwei Zahlen, welche die Position des Elementes innerhalb der Matrix beschreiben: **die Indizes** (Einzahl **„der Index“**). Die Zählung für die beiden Indizes beginnt je mit 1 in der linken oberen Ecke. Passend zur Angabe der Matrizen-Dimensionen ist

der erste Index für die Angabe der Zeilennummer und der zweite für die Spaltennummer des anzusprechenden Elementes:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Das Element a_{11} steht bei einer nxm Matrix A also links oben und entsprechend steht a_{nm} rechts unten. Welches sind in diesem Beispiel die Indizes der Elemente rechts oben und links unten?

Mit der Index-Definition kann die Bedingung für eine Diagonalmatrix mathematisch definiert werden durch $d_{ij} = 0$ für $i \neq j$.

Übung 12–2 Welche Indizes haben die Elemente rechts oben, rechts unten, links unten, sowie in der Mitte der untersten Zeile bei einer 4x5 Matrix?

Übung 12–3 Geben Sie die Indizes der Diagonalelemente einer 4x4 Matrix der Reihe nach von links oben beginnend an!

1.2.3 Das Transponieren einer Matrix

Die Operation, bei welcher die Elemente einer Matrix ihre Plätze so ändern, dass der Zeilenindex zum Spaltenindex wird und umgekehrt, heißt Transponieren. Da für die Elemente auf der Diagonalen beide Indizes gleich sind, bleiben sie dabei am selben Ort. Für Rechtecksmatrizen werden dabei natürlich auch die beiden Dimensionszahlen untereinander vertauscht.

Eine quadratische Matrix, welche beim Transponieren in sich selbst übergeht, bei der also $S^T = S$ gilt, heißt **symmetrisch**.

Man kann das Transponieren auch als Spiegeln der Elemente an der Matrixdiagonalen beschreiben. Die Grafik zeigt dies für quadratische und rechteckige Matrizen und einen Vektor.

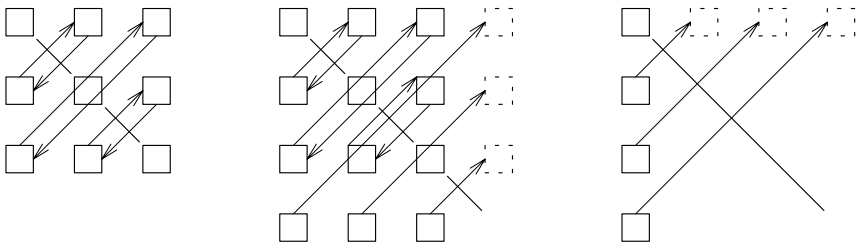

Abbildung 1.3: Transponieren einer quadratischen Matrix, einer Rechtecksmatrix und eines Vektors.

Die Operation des Transponierens wird in der Mathematik mit einem hochgestellten „T“ angezeigt: A^T ist die zu A transponierte Matrix. MATLAB verwendet als Transpositionsoperator den Apostroph: In diesem Buch wird bei MATLAB-Befehlen also meistens A' eingesetzt für die zu A transponierte Matrix A^T.

Anmerkung

Der MATLAB-Operator ’ bedeutet genaugenommen Transponieren mit Übergang zur konjugiert komplexen Zahl (Fachausdruck: Adjungieren). (Für das reine Transponieren gibt es den Operator ” .’ “ – Punkt, gefolgt vom Apostroph).

Solange man mit reellen Matrizen arbeitet ist diese Unterscheidung jedoch bedeutungslos und daher wird der Einfachheit halber meist nur der Operator ' verwendet. Den Unterschied sollte man sich jedoch im Hinterkopf merken, um verwirrende Überraschungen zu vermeiden.

Zur Illustration sei das Transponieren einer Rechtecksmatrix ausführlich dargestellt (die Dimensionszahlen vertauschen dabei ihren Platz, aus der 3x4 wird eine 4x3 Matrix) :

$$\begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \end{pmatrix}^T = \begin{pmatrix} 11 & 21 & 31 \\ 12 & 22 & 32 \\ 13 & 23 & 33 \\ 14 & 24 & 34 \end{pmatrix}$$

Übung 12–4 Schreiben Sie die Transponierten z' des Zeilenvektors $z = (1\ 2\ 3\ 4)$ sowie M' der 2x3 Matrix M= $\begin{pmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \end{pmatrix}$ auf!

1.2.4 Addition und Subtraktion von Matrizen

Für zwei Matrizen A und B, welche dieselben Dimensionszahlen aufweisen, die also beide n×m Matrizen sind, ist die Addition $A + B$ und die Subtraktion $A - B$ (und auch $B - A$) der beiden definiert. Diese Operationen erfolgen durch Addition $a_{ij} + b_{ij}$ bzw. Subtraktion $a_{ij} - b_{ij}$ aller einander entsprechenden Elemente $i = 1 \ldots n,\ j = 1 \ldots m$. Man sagt, die Addition und Subtraktion von Matrizen sei **elementweise** definiert.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} - \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11} - b_{11} & a_{12} - b_{12} & a_{13} - b_{13} \\ a_{21} - b_{21} & a_{22} - b_{22} & a_{23} - b_{23} \\ a_{31} - b_{31} & a_{32} - b_{32} & a_{33} - b_{33} \end{pmatrix}$$

Merke: Addition und Subtraktion sind nur zwischen Matrizen mit identischen Dimensionen definiert!

Die Multiplikation einer Matrix mit einem Skalar

Die Multiplikation einer Matrix A mit einem Skalar s (d. h. mit einer normalen einzelnen Zahl s) ist ebenfalls elementweise definiert: als Multiplikation jedes einzelnen Elementes mit diesem (Skalar-) Zahlenwert.

$$s \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} s \cdot a_{11} & s \cdot a_{12} & s \cdot a_{13} \\ s \cdot a_{21} & s \cdot a_{22} & s \cdot a_{23} \\ s \cdot a_{31} & s \cdot a_{32} & s \cdot a_{33} \end{pmatrix}$$

Übung 12–5 Bilden Sie die Transponierte A' der Matrix $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ und anschließend die Matrizen $A' + A,\ \ A - A',\ \ (A + A),\ \ 2 * A$!

1.2.5 Das Produkt von zwei Matrizen

Definition des Matrizenproduktes

Das Produkt von zwei Matrizen $A \cdot B$ ist wesentlich komplizierter als die Addition und Subtraktion, da nicht einfach einzelne Matrixelemente miteinander multipliziert werden. Für die Beschreibung des Matrizenproduktes bleiben wir beim Beispiel der zwei 3x3 Matrizen A und B und zeigen die Entstehung von deren Produktmatrix $A \cdot B$, die wir C nennen.
Vorsicht! Bei der Matrizenmultiplikation ist die Reihenfolge der Faktoren wichtig, $A \cdot B$ ist also im Allgemeinen verschieden von $B \cdot A$.

Bei der Definition des Matrizenprodukts $A \cdot B = C$:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

ist z. B. c_{11} definiert durch die Formel
$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31}$

Die allgemeine Formel für das Element c_{ij} der Resultatmatrix lautet:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj}$$

Die Abläufe in diesem Summationsverfahren werden kurz vorgestellt:
Die Indizes i und j sind die Positionen (Zeile i, Spalte j) des zu berechnenden Elementes. Gleichzeitig ist i die Zeilennummer der aus der linken Matrix zur Verarbeitung auszuwählenden Zeile $a_{i\,?}$. Ebenso ist j die Spaltennummer der aus der rechten Matrix zu verarbeitenden Spalte $b_{?\,j}$. Der Index k durchläuft alle Werte von 1 bis n, wobei jedesmal ein Produkt aus einem a- und einem b-Element gebildet wird und anschließend diese Produkte zum Resultat-Element summiert werden.

Die Anwendung der allgemeinen Formel für die Berechnung des Elementes c_{32} soll dies nochmals illustrieren:
c_{32} steht in der 3. Zeile und der 2. Spalte von C,
also muss die gesamte 3. Zeile von A d. h. a_{31}, a_{32}, a_{33}
mit der gesamten 2. Spalte von B, d. h. b_{12}, b_{22}, b_{32} verarbeitet werden zu:
$c_{32} = a_{31} * b_{12} + a_{32} * b_{22} + a_{33} * b_{32}$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \mathbf{a_{31}} & \mathbf{a_{32}} & \mathbf{a_{33}} \end{pmatrix} * \begin{pmatrix} b_{11} & \mathbf{b_{12}} & b_{13} \\ b_{21} & \mathbf{b_{22}} & b_{23} \\ b_{31} & \mathbf{b_{32}} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & \mathbf{c_{32}} & c_{33} \end{pmatrix}$$

Existenzbedingung für das Matrizenprodukt

Dieses Prinzip des Verarbeitens einer Zeile aus der linken Matrix mit einer Spalte aus der rechten bedingt natürlich, dass die Zeilen der linken Matrix dieselbe Länge aufweisen wie die Spalten der rechten. Bei der Multiplikation von quadratischen Matrizen derselben Dimension ist diese Forderung automatisch erfüllt, aber bei der Multiplikation von Rechtecksmatrizen

oder von Matrizen mit Vektoren ist dies eine notwendige Voraussetzung, damit eine Multiplikation überhaupt möglich ist:

Eine Multiplikation von zwei Rechtecksmatrizen ist an die Bedingung geknüpft, dass die Zeilenlänge (= Anzahl Spalten, = 2. Dimensionszahl) der links stehenden Matrix mit der Spaltenlänge (= Anzahl Zeilen, = 1. Dimensionszahl) der rechts stehenden Matrix übereinstimmt. Nochmals anders formuliert: die 2. Dimensionszahl der linken muss der 1. Dimensionszahl der rechten Matrix entsprechen, oder: die **innenliegenden Dimensionszahlen** müssen gleich sein. Dies entspricht der MATLAB-Fehlermeldung "inner dimensions must agree", die beim Versuch, zwei Matrizen zu multiplizieren, bei denen diese Bedingung verletzt ist, ausgegeben wird.

Bedingung für die Existenz des Produktes von zwei Matrizen Das Produkt $A \cdot B$ einer n_1xm_1 Matrix A mit einer n_2xm_2 Matrix B existiert, falls $m_1 = n_2$ ist. Die innenliegenden Dimensionszahlen müssen also übereinstimmen. Das Resultat erhält dann die Dimensionen n_1xm_2 (d. h. der außenliegenden Dimensionszahlen).

$$(n_1 \times m_1) \cdot (n_2 \times m_2) \rightarrow (n_1 \times m_2), \quad \text{falls} \quad (m_1 = n_2)$$

Matrix–Vektor-Multiplikation

Eine häufige Anwendung der Multiplikation von Rechtecksmatrizen ist die Matrix–Vektor-Multiplikation:

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

Das Skalarprodukt

Auch das altbekannte Skalarprodukt von zwei Vektoren ist nach dieser Definition eine Matrixmultiplikation von zwei Rechtecksmatrizen, einem Zeilenvektor von links mit einem Spaltenvektor von rechts: $w'(1x3) \cdot v(3x1) = s(1x1)$

$$s = w' \cdot v = \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = w_1 \cdot v_1 + w_2 \cdot v_2 + w_3 \cdot v_3$$

Das dyadische Produkt von zwei Vektoren

Die andere Reihenfolge für die Multiplikation von zwei gleich langen Vektoren (nach den Regeln auch möglich), das dyadische Produkt, ist wenig bekannt. Dieses Produkt von einem (nx1) Spaltenvektor mit einem (1xn) Zeilenvektor ergibt eine nxn-Matrix.

$$D = v \cdot w' = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} = \begin{pmatrix} v_1 \cdot w_1 & v_1 \cdot w_2 & v_1 \cdot w_3 \\ v_2 \cdot w_1 & v_2 \cdot w_2 & v_2 \cdot w_3 \\ v_3 \cdot w_1 & v_3 \cdot w_2 & v_3 \cdot w_3 \end{pmatrix}$$

Da in diesem Fall die innenliegenden Dimensionen beide Eins sind, gibt es bei den Resultat-Elementen statt der Summe aus mehreren Produkten nur ein einziges Produkt.

Das Matrizenprodukt, definiert durch Skalarprodukte

Das Produkt von zwei Matrizen lässt sich auch als Zusammenstellung von lauter Skalarprodukten darstellen. Dazu definiert man zwei Ausschneideoperatoren $r_i(A)$ und $c_j(B)$. Der Operator $r_i(A)$ (row(i,A) schneidet aus der Matrix A (Faktor links) einzelne Zeilen heraus und der Operator $c_j(B)$ (col(j,B) einzelne Spalten aus der Matrix B (Faktor rechts). Dies ergibt:

$$A \cdot B = \begin{pmatrix} r_1(A) \cdot c_1(B) & r_1(A) \cdot c_2(B) & r_1(A) \cdot c_3(B) \\ r_2(A) \cdot c_1(B) & r_2(A) \cdot c_2(B) & r_2(A) \cdot c_3(B) \\ r_3(A) \cdot c_1(B) & r_3(A) \cdot c_2(B) & r_3(A) \cdot c_3(B) \end{pmatrix}$$

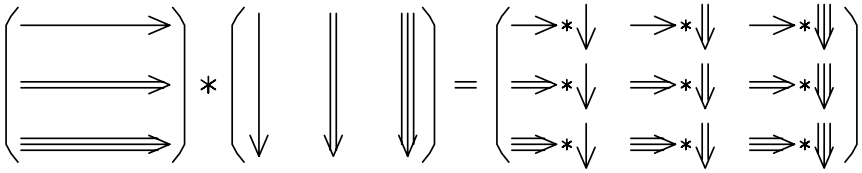

Abbildung 1.4: Grafik zum Prinzip der Matrixmultiplikation.

Das grundlegende Prinzip der Matrix–Matrix-Multiplikation soll zudem durch die obige grafische Darstellung einprägsam präsentiert werden!

Übung 12–6 Verwenden Sie die Matrix $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ und deren Transponierte A' zum Bilden der Produkte $A' * A,\ \ A * A',\ \ A * A$!

Zusammenfassung Bei der Matrizenmultiplikation werden Zeilen aus der links stehenden Matrix mit Spalten aus der rechts stehenden verarbeitet. Diese Verarbeitung bildet für jedes Element des Resultates die Summe aus lauter paarweisen Produkten. Die innen aneinander stoßenden Dimensionszahlen müssen deshalb gleich sein, sonst ist die Matrixmultiplikation gar nicht definiert. Für rechteckige Matrizen ist oft nur eine Reihenfolge des Produktes von zwei Matrizen überhaupt definiert. Für quadratische Matrizen ist die Multiplikation von zwei Matrizen in beliebiger Reihenfolge möglich. Das Resultat hängt aber im Normalfall von der Reihenfolge der Faktoren ab. Im Gegensatz zum altbekannten Rechnen mit Zahlen gilt also für Matrizen das kommutative Gesetz **nicht**, d. h., als Formel geschrieben, im Allgemeinen gilt (Spezialfälle ausgenommen) für zwei Matrizen A und B: $A \cdot B \neq B \cdot A$.

1.2.6 Die Einheitsmatrix

Als einfache Anwendung der Multiplikationsvorschrift für Matrizen suchen wir die spezielle Matrix, welche eine beliebige andere bei der Multiplikation unverändert lässt. Diese Matrix heißt Einheitsmatrix und übernimmt im Umfeld der Matrizen die Rolle, welche der Eins beim Zahlenrechnen zukommt. Gelehrt ausgedrückt, nennt man sie auch das Neutralelement der Multiplikation.

Für 2x2 Matrizen lässt sich die Einheitsmatrix relativ einfach bestimmen. Eine 2x2 Matrix mit beliebigen Zahlen a, b, c, d soll durch Multiplikation mit der zu bestimmenden 2x2 Einheitsmatrix $\left(\begin{smallmatrix} x & y \\ z & u \end{smallmatrix}\right)$ in sich selbst überführt werden.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} * \begin{pmatrix} x & y \\ z & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Aus der Multiplikationsvorschrift ergeben sich die Gleichungen

$$a \cdot x + b \cdot z = a \qquad a \cdot y + b \cdot u = b \qquad c \cdot x + d \cdot z = c \qquad c \cdot y + d \cdot u = d$$

deren Lösungen leicht durch Erraten gefunden werden können. Es sind dies $x = 1$, $y = 0$, $z = 0$ und $u = 1$. Die 2x2 Einheitsmatrix ist also:

$$I = \begin{pmatrix} x & y \\ z & u \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Natürlich haben wir damit die Einheitsmatrix nur für den Fall 2x2, und nur für die Multiplikation von rechts her bestimmt, eben im Sinne eines Beispiels.

Das Prinzip gilt jedoch ganz allgemein: Die nxn Einheitsmatrix ist die Diagonalmatrix mit lauter Einsen in der Diagonalen. Eine beliebige Matrix bleibt sowohl bei der Multiplikation mit einer passenden Einheitsmatrix von links her als auch mit einer passenden Einheitsmatrix von rechts her unverändert. Die gleiche Art Matrix, die Einheitsmatrix, funktioniert als Links-Neutralelement und als Rechts-Neutralelement der Matrixmultiplikation. Der englische Fachausdruck für die Einheitsmatrix lautet “identity” oder “identity matrix” und der gängige Formelname dafür ist „I“.

In MATLAB gibt es eine Funktion, welche eine nxn Einheitsmatrix liefert, sie heißt `eye(n)` und wird nach dem Schema aufgerufen `I = eye(n)`. Als Merkhilfe kann man sich vergegenwärtigen, dass das Wort eye im Englischen genauso ausgesprochen wird wie der Buchstabe I. Eine weitere Gedächtnisstütze ergibt sich aus der raffinierten Reklame eines Brillengeschäftes “find your eye-dentity!” frei übersetzt: „Finden Sie über Ihre Augen und Ihre Brille zu Ihrer wahren Identität!“

1.2.7 Kann man auch durch Matrizen dividieren?

Das Dividieren durch eine Matrix ist nicht direkt als Matrizenoperation definiert. Indirekt definiert man die Division durch eine Matrix als Multiplikation mit der inversen Matrix. Die zu einer Matrix A inverse Matrix wird mit A^{-1} umschrieben. Es wird also mit A^{-1} multipliziert statt durch A dividiert. Für das Paar von Matrizen A und A^{-1}, einer Matrix und ihrer Inversen, gilt immer die Vertauschbarkeit der Multiplikationsreihenfolge. Es gilt also immer (falls A^{-1} existiert): $A \cdot A^{-1} = A^{-1} \cdot A = I$, d. h. die Links-Inverse ist gleich der Rechts-Inversen. Die inverse Matrix A^{-1} ist dadurch definiert, dass ihr Produkt mit der Originalmatrix die Einheitsmatrix I ergeben muss, also $A^{-1} \cdot A = I$.

Die Möglichkeit durch eine Matrix zu dividieren, hängt also davon ab, ob zur gegebenen Matrix eine Inverse existiert. Dies ist grundsätzlich nur für quadratische Matrizen möglich. Man erinnere sich in diesem Zusammenhang an die Forderung, dass es zur Lösung eines linearen Gleichungssystems ebensoviele Gleichungen wie Unbekannte braucht.

Da die Matrizenmultiplikation von links und von rechts her mit einer beliebigen Matrix B in der Regel verschiedene Resultate liefert, muss auch für die Division unterschieden werden zwischen der Division durch eine rechts stehende und der Division durch eine links stehende Matrix. Der „Rechts“-Division entspricht die Multiplikation von rechts mit A^{-1}

$$A/B \stackrel{\text{def}}{=} B \cdot A^{-1}$$

und der „Links“-Division entspricht die Multiplikation von links mit A^{-1}

$$A\backslash B \stackrel{\text{def}}{=} A^{-1} \cdot B$$

In der Mathematik ist keine der beiden Divisions-Schreibweisen gebräuchlich, Operationen mit Matrizen, welche Divisionen entsprechen, werden in mathematischer Formulierung immer durch Multiplikation mit der inversen Matrix A^{-1} umschrieben.

In MATLAB hingegen entspricht diese Schreibweise dem Normalfall. Dazu wird in MATLAB ein neuer Operator eingeführt, der Links-Divisionsoperator.

Die Schreibweise $A\backslash B$ benutzt den in MATLAB für die Links-Division neu eingeführten Operator $\backslash$, um die Division durch eine Matrix von links in einer, der normalen Division möglichst ähnlichen Art, darstellen zu können.

Der Operator der Links-Division kommt in MATLAB häufig vor, weil damit die Lösung von linearen Gleichungssystemen formuliert wird. Ein Gleichungssystem $A \cdot x = b$ hat die Lösung $x = A^{-1} \cdot b$ oder eben `x = A \ b`.

Das hier vorgestellte Prinzip der Umschreibung einer Division als Multiplikation mit einem aus dem Divisor berechneten Wert findet sich auch bei den komplexen Zahlen:

$$t/z \;=\; t \cdot \left(\frac{\overline{z}}{z \cdot \overline{z}} \right)$$

Existenz der inversen Matrix

Bereits beim Rechnen mit normalen Zahlen kann es vorkommen, dass eine Division nicht durchführbar ist, wenn nämlich der Divisor null ist.

Beim Rechnen mit Matrizen verschiebt sich die Frage, ob die Division durch eine gegebene Matrix durchführbar ist, vorerst auf die Frage, ob zur gegebenen Matrix eine Inverse existiert. Die Existenz einer Inversen zu einer vorliegenden Matrix wiederum wird durch die Frage umschrieben, ob eine Matrix **regulär** sei (dann existiert eine Inverse) oder **singulär** (dann existiert keine Inverse). Dieser Problemkreis wird in Kapitel 3 eingehend diskutiert. Deshalb werden hier nur einige der wichtigsten Tatsachen zu dieser Frage erwähnt.

Regulär können nur quadratische Matrizen sein und sie sind es nur, wenn ihre Spalten, als Vektoren betrachtet, voneinander linear unabhängig sind (dann sind es auch ihre Zeilenvektoren). Bei linear unabhängigen Vektorsätzen ist keiner dieser Vektoren durch irgend eine Kombination der anderen ersetzbar. Damit spannen die Spaltenvektoren einer regulären nxn Matrix den gesamten n-dimensionalen Vektorraum auf. In diesem regulär genannten Fall existiert eine inverse Matrix und damit ist auch die Division durch diese Matrix definiert.

1.3 Matrizenrechnung in der Praxis mit MATLAB

Matrizenrechnung ist die eigentliche Stärke von MATLAB – alle mathematischen Objekte in MATLAB sind im Grunde Matrizen. Benutzen Sie diese Tatsache, um sich durch die Anwendung von MATLAB mit der Matrizenrechnung näher vertraut zu machen!

1.3.1 Das Arbeiten mit M-Files und Funktions-M-Files

Skript M-Files

Für die im nächsten Abschnitt folgenden Beispiele kann es sich bereits lohnen, die Möglichkeit zum Verwenden von **M-Files** einzusetzen. MATLAB verfügt über eine ausgedehnte Makro-Funktionalität: alle Befehle, die Sie in MATLAB interaktiv eingeben würden, können Sie stattdessen in ein File mit der Erweiterung **.m** schreiben. Anschließend können Sie dieses Befehls-Makro innerhalb von MATLAB durch Angabe des entsprechenden Dateinamens – ohne die Endung .m – ablaufen lassen. Als Beipiel können Sie die untenstehenden Befehle (mit dem in MATLAB enthaltenen M-File Editor, mit xemacs oder irgend einem anderen Texteditor) in das File `matdefs.m` schreiben. Die vorher eingetippten Befehle können Sie dann in MATLAB durch Eintippen von `matdefs` abrufen. Solche M-Files können beliebige Aufrufe von anderen M-Files in beliebiger Verschachtelungstiefe enthalten. Bei Funktions-M-Files ist sogar ein Aufruf der eigenen Funktion im Innern erlaubt, also eine Rekursion.

Funktions-M-Files

Über die reine Makro-Funktion der als „Skript-M-Files“ zu bezeichnenden (gewöhnlichen M-Files) hinausgehend, sind in MATLAB **Funktions-M-Files** möglich.
Die Definition einer solchen Funktion beginnt mit dem Codewort “function”. Dann folgt die Deklaration der Rückgabeparameter, dann ein Gleichheitszeichen, gefolgt vom Funktionsnamen, und zum Schluss werden in runden Klammern die Eingabeparameter deklariert. Der Funktionsname muss mit dem Filenamen übereinstimmen (ohne die Erweiterung .m).

Als Beispiel wird die Erstellung von Winkelfunktionen gezeigt, deren Eingabeparameter in Grad angegeben werden können. Das File `cosdeg.m` enthält somit die folgenden Befehle:

```
% Funktion cosdeg(w) , liefert Kosinus,
% ~~ erwartet Winkel in Grad
function cosbk = cosdeg(w)
cosbk = cos(w*pi/180)}
```

Das entsprechende File `sindeg.m` sieht dann wie folgt aus:

```
% Funktion sindeg(w) , liefert Sinus,
% ~~ erwartet Winkel in Grad
function sinbk = sindeg(w)
sinbk = sin(w*pi/180)
```

In MATLAB ist es möglich, mehrere Rückgabeparameter zu verwenden. Um diese Möglichkeit auszunutzen muss eine Gruppe von Rückgabeparametern sowohl bei der Deklaration

als auch beim Funktionsaufruf durch ein Paar von eckigen Klammern zusammengefasst werden.

Der Funktionsaufruf, also die Aktivierung eines Funktions-M-Files erfolgt in der von anderen Programmiersprachen her gewohnten Weise nach dem Schema

```
Resultatvariable = Funktionsname(Eingabeparameter1, Parameter2)
```

bzw. für mehrere Rückgabeparameter durch:

```
[Resultat1, Resultat2 ] = Funktionsname(Eingabeparameter, ...)
```

Achtung! Der offizielle Name M-File wird mit großem 'M' geschrieben, die Namenserweiterung '.m' muss aber immer klein geschrieben werden!

Der Hauptteil des Dateinamens (ohne die Erweiterung) darf Groß- und Keinbuchstaben, sowie Zahlen enthalten. Der Name muss aber mit einem Buchstaben beginnen. Der Aufruf muss bezüglich Groß- und Kleinschreibung exakt mit der Deklaration der Funktion übereinstimmen!

Sehr zu empfehlen ist es, die Möglichkeit zum Einfügen von Kommentartext in M-Files eifrig zu benutzen: Alle Bestandteile einer Zeile, welche nach einem Prozentzeichen (%) stehen, gelten in MATLAB als Kommentar. Diejenigen Kommentarzeilen, welche in einem M-File vor dem ersten ausführbaren Kommando stehen, werden durch die Hilfefunktion von MATLAB ausgewertet und angezeigt. Beim vorherigen Beispiel `cosdeg.m` erscheint auf das Kommando `help cosdeg` der Text der beiden Kommentarzeilen auf dem Bildschirm. Benutzen Sie diese Möglichkeit, um die Hilfefunktion auch bei Ihren eigenen M-Files verfügbar zu machen!

Ein wesentlicher Unterschied zwischen Skript-M-Files und Funktions-M-Files besteht in der Sichtbarkeit der innerhalb des M-Files verwendeten Variablen. Ein Skript-M-File holt alle Variablen mit genau den im Skript-Text enthaltenen Namen aus der allgemeinen Arbeitsumgebung, dem "workspace" und legt auch alle Resultate dort ab. Ganz im Gegensatz dazu werden an eine Funktion nur die mit dem Funktionsaufruf übergebenen Werte übermittelt. Man nennt diese die aktuellen Eingabeparameter. Die im Innern der Funktion vorkommenden Variablennamen, inklusive die Namen der deklarierten Eingabe- und Ausgabeparameter, sind vom Workspace aus unsichtbar. Erst die beim Funktionsaufruf angegebenen Ausgabeparameter sind wieder Variablennamen, die der Arbeitsoberfläche bekannt sind.

Funktions-M-Files rekursiv

Wie in verschiedenen anderen Programmiersprachen können auch in MATLAB die Funktionsaufrufe rekursiv sein. Das bedeutet, dass eine Funktion sich selbst im Inneren aufrufen darf. Das birgt natürlich die Gefahr von unendlichen Schleifen. Um dem vorzubeugen sind rekursive Funktionen streng an drei elementare Regeln gebunden:

- Die rekursive Funktion enthält einen Parameter zur Beschreibung der Komplexitätsstufe.
- Im Innern dürfen nur Aufrufe mit tieferer Komplexitätsstufe vorkommen, als beim Aufruf der Funktion verlangt wurde.
- Es muss eine tiefste Komplexitätsstufe geben, bei der das Resultat direkt erarbeitet werden kann, ohne im Inneren noch einen weiteren Aufruf der Funktion zu verlangen.

Fakultät, rekursiv berechnet Ein ganz einfaches Beispiel, das helfen kann, dieses algorithmische Prinzip zu verstehen, ist die Berechnung des Fakultätswertes einer natürlichen Zahl mit Hilfe einer rekursiven Funktion.

Der Fakultätswert einer natürlichen Zahl n wird mit $n!$ bezeichnet und ist definiert als

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

Diese Funktion wird nun rekursiv formuliert:
Zu einem gegebenen Wert n ist der Fakultätswert n mal der Fakultätswert von $(n-1)$.
Falls n größer ist als 1, so wird die Fakultätsfunktion für den Wert $(n-1)$ aufgerufen und dieses Resultat mit n multipliziert.
Falls $n \leq 1$ ist, so kann direkt das Resultat $n! = 1$ zurückgegeben werden.

Als Bedingung für den einfachsten Fall wurde $(n \leq 1)$ eingesetzt statt $n = 1$. Damit ist auch der exotische Fall abgedeckt, dass man den Wert von $0!$ als $0! = 1$ definiert, obwohl das nach der elementaren Definition als unlogisch erscheint. Diese Definition hat sich aber als sehr praktisch erwiesen, weil damit verschiedene Formeln der Kombinatorik und der Reihenentwicklung ohne Formulierung von Ausnahmefällen auskommen.

```
%fakurekur.m
% rekursive Funktion zum Berechnen der Fakultaet
function fw = fakurekur(n)
if n <= 1
  fw = 1;
else
  fw = n * fakurekur(n-1);
end
```

Die Türme von Hanoi Das berühmteste Übungsbeispiel zur rekursiven Programmierung ist die Konzentrations- und Meditationsübung mit dem Namen „die Türme von Hanoi“.

Mehrere Scheiben mit ständig abnehmendem Durchmesser ergeben aufeinandergeschichtet einen Turm. Die Aufgabe besteht darin, den Turm durch Bewegen von nur einer Scheibe auf einmal an einen anderen Platz zu verschieben. Es steht dazu nur ein Hilfsablageplatz zur Verfügung, und die Grundregel verlangt, dass in Zwischenstufen immer nur kleinere Scheiben auf größere gelegt werden dürfen. Selbstverständlich können auch wieder Scheiben auf den ursprünglichen Platz gelegt werden, solange die Grundregel nicht verletzt wird.

Die rekursive Lösung dieses Problems beruht auf der Einsicht, dass man einen um eins kleineren Turm auf den Hilfsplatz verschieben kann, dann die unterste Scheibe einzeln auf den definitiven Platz setzt und anschließend den kleineren Turm vom Hilfsplatz auf die Scheibe am definitiven Platz aufschichtet. Die Aufgabe, einen Turm der Höhe n von Platz A, mit Hilfe von B nach C zu verschieben, kann also in die folgenden Teilschritte unterteilt werden:

- verschiebe Turm $(n-1)$ von A mit Hilfe von C nach B
- verschiebe einzelne Scheibe $(n = 1)$ von A nach C
- verschiebe Turm $(n-1)$ von B mit Hilfe von A nach C

In der rekursiven MATLAB-Funktion wird zusätzlich ein Parameter mitgeführt, welcher die Scheibennummer anzeigt:

```
% hanoireku.m
% Uebung der Tuerme von Hanoi
% Aufruf-Beispiel hanoireku(4,'A','B','C')
function hanoireku(n,  von , via, nach)
format compact
if n > 1
  hanoireku(n-1,  von, nach, via);
    disp(sprintf('Scheibe %d von %s nach %s ', n, von, nach ));
  hanoireku(n-1,  via, von, nach);
else
  disp(sprintf('Scheibe %d von %s nach %s ', 1, von, nach ));
end
```

Die Übung selbst mit einigen wenigen Scheiben zu versuchen, ist recht amüsant. Beim sonntäglichen Frühstück können eine Tasse, eine Untertasse, ein Dessertteller und ein großer Teller als Scheiben verwendet werden.

1.3.2 Einstieg in die Matrizenrechnung mit MATLAB

Addition und Subtraktion von Matrizen

Geben Sie die 4 Matrizen A, B, C und D ein (alle 2x2):

```
A = [ 1 0 ; 0 0 ]
B = [ 0 2 ; 0 0 ]
C = [ 0 0 ; 3 0 ]
D = [ 0 0 ; 0 4 ]
```

und bilden Sie die verschiedenen Summen A+B, ... B+D, ... A+B+C+D!
Ebenso sind die Summen A+A, C+C+C von Interesse. Das Erfreuliche an dieser Übung ist, dass man das Resultat gut im Voraus erahnen kann.

Multiplikation mit einem Skalar

Die Summe aller Matrizen A+B+C+D aus dem vorherigen Beispiel soll der Variablen S zugewiesen werden. Untersuchen Sie anschließend, was sich ergibt, wenn Sie einige Beispielrechnungen wie etwa die Folgenden ausführen:

```
T = 0.02*S
R = 20000*S
U = 50.001* T
```

Lassen Sie sich diese Matrizen ausgeben, nachdem Sie das Ausgabeformat geändert haben (mit `format ..`). Um nach dem Ändern des Ausgabeformates nochmals festzustellen, welche Variablen im “workspace” verfügbar sind, dienen die Kommandos `who` und `whos`. Den Inhalt einer Variablen lässt man sich anzeigen, indem man den Variablennamen eintippt (natürlich ohne Semikolon am Ende!).

Definierte und undefinierte Matrizenmultiplikationen

Erzeugen Sie neben der 2x2 Matrix S aus dem vorherigen Beispiel noch eine 3x2 Rechtecksmatrix W und einen 2x1 Vektor v. Benutzen Sie den Transpositionsoperator **()'**, um den vorerst als Zeile eingegebenen Vektor v sofort in die übliche Spaltenvektor-Form zu bringen.

```
W = [ 1 1 ; 3 0 ;0 1 ]; S = [1 2; 3 4]
v = [ 4 5]'
```

Die vollständige Liste der anschließend verwendeten Matrizen und Vektoren, inklusive deren Transponierte ist:

```
S, S', W, W', v, v'
```

Zur besseren Übersicht können Sie sich diese zuerst anzeigen lassen. Dabei erhalten Sie zugleich eine anschauliche Demonstration des Transponierens einer Matrix.

Probieren Sie anschließend aus, vielleicht erst nachdem sie sich überlegt haben, was das Ergebnis sein sollte, welche der folgenden Operationen legal sind, und welches Resultat sich im erlaubten Fall ergibt.

```
W * S
S * W
S*W'
S*v
W*v
v*v
v'*v
v*v'
```

Erzeugen einer Diagonalmatrix

Mit dem Befehl `Dia = zeros(5)` können sie als Start eine 5x5 Matrix aus lauter Nullen erzeugen. Anschließend sollen Sie mit dem MATLAB-Indizierungsprinzip die Diagonalelemente mit den Zahlen 1 bis 5 füllen: `Dia(1,1) = 1` ... etc.

Ausblick: Diese Aufgabe könnte für ein beliebiges, vorgegebenes n mit einer Schleifenkonstruktion gelöst werden, es existiert aber auch eine Bibliotheksfunktion `diag()` dafür.

Produkte von Diagonalmatrizen

Bei einem Produkt von zwei Diagonalmatrizen ist die Reihenfolge der Faktoren im Gegensatz zum allgemeinen Fall immer vertauschbar. Testen Sie diese Tatsache, indem Sie mit Hilfe der Funktionen `rand()` und `diag()` zufällige Paare von Diagonalmatrizen erzeugen.

Die Eingaben in MATLAB dazu lauten:

```
D1 = diag( rand(1,5) )  ;   D2 = diag( rand(1,5) )
P1 = D1*D2
P2 = D2*D1
P2 - P1
```

Das Produkt einer Rechtecksmatrix mit ihrer Transponierten

Jede beliebige Rechtecksmatrix kann mit ihrer transponierten Matrix multipliziert werden. Durch das Transponieren werden die Indizes und damit auch die Dimensionszahlen vertauscht.

Bei der Reihenfolge $A \cdot A^T$ ergibt sich durch das Transponieren aus der Spaltenzahl der links stehenden Matrix die Zeilenzahl der rechts stehenden. Bei $A^T \cdot A$ ergibt sich durch das Transponieren bei A^T die Zahl von Spalten, die der Zahl von Zeilen bei A entspricht. In beiden Fällen ist die Bedingung für das Multiplizieren der beiden Matrizen erfüllt.

Die Resultate der beiden Multiplikationsreihenfolgen sind voneinander verschieden. In beiden Fällen ist das Resultat eine quadratische Matrix; das eine Mal mit der kleineren, das andere Mal mit der größeren Dimension der ursprünglichen Rechtecksmatrix. Beide quadratischen Matrizen, sowohl $A \cdot A^T$ als auch $A^T \cdot A$, sind übrigens symmetrische Matrizen.

Beispiel:

```
M = [ 1 2 3 4 5 ; 1 0 2 0 3]
K = M * M'
G = M' * M
```

mit den Resultaten:

$$K = \begin{pmatrix} 55 & 22 \\ 22 & 14 \end{pmatrix} \qquad G = \begin{pmatrix} 2 & 2 & 5 & 4 & 8 \\ 2 & 4 & 6 & 8 & 10 \\ 5 & 6 & 13 & 12 & 21 \\ 4 & 8 & 12 & 16 & 20 \\ 8 & 10 & 21 & 20 & 34 \end{pmatrix}$$

Für den Spezialfall von Rechtecksmatrizen, den Vektoren, sind wir diesem Prinzip im vorhergehenden Abschnitt schon begegnet. Das Produkt Zeilenvektor mal Spaltenvektor ist das Skalarprodukt. Das gilt gleichermaßen für das Produkt des transponierten Spaltenvektors mit dem Original, also für $v^T \cdot v$; es ergibt sich das Skalarprodukt von v mit sich selbst. Die andere Reihenfolge der Faktoren, $v \cdot v^T$ liefert bei Vektoren der Länge n eine nxn Matrix, das dyadische Produkt von v mit sich selbst.

Suchen Sie das Neutralelement der Addition bei 2x2 Matrizen

Das neutrale Element bezüglich einer Operation (hier der Addition) lässt eine beliebige Matrix unverändert, wenn sie mit diesem verarbeitet wird: $A+N_{add} = A$. Aus der Matrizengleichung für die Matrix A mit den beliebigen Elementen `a,b,c,d` und der obigen Bedingung für das Neutralelement können die Unbekannten `x,y,z,u` leicht durch Nachrechnen per Hand bestimmt werden:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} + \begin{pmatrix} x & y \\ z & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Das Resultat überrascht nicht, N_{add} ist die Nullmatrix, also die Matrix, welche lauter Nullen enthält.

$$N_{add} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Neutralelemente der Multiplikation bei 2x2 Matrizen

Bei der Multiplikation muss man vorerst von zwei möglichen Neutralelementen ausgehen, da die Faktoren bei der Matrixmultiplikation nicht vertauschbar sind. Die beiden Matrizengleichungen lauten:

– Für das Links-Neutralelement der Multiplikation

$$N_{mult-L} * A = A \quad : \quad \begin{pmatrix} x & y \\ z & u \end{pmatrix} * \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

– und für das Rechts-Neutralelement der Multiplikation

$$A * N_{mult-R} = A \quad : \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix} * \begin{pmatrix} x & y \\ z & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Führen Sie in beiden Fällen die Matrixmultiplikation von Hand aus und bestimmen Sie jedesmal x, y, z, u aus den daraus entstehenden Gleichungen. (Das Ausführen von Matrizenmultiplikationen von Hand verstärkt Ihr Verständnis für diesen komplizierten Vorgang). Dabei ist zu beachten, dass die Matrizengleichungen für die Übereinstimmung jedes einzelnen Elementes je eine gewöhnliche Gleichung ergeben, und dass die Gleichungen für beliebige Werte von a, b, c, d gelten müssen (ein Neutralelement soll die Neutralitätseigenschaft nicht nur für spezielle ausgewählte Fälle aufweisen).

Das Resultat wird zeigen, dass die jeweiligen Neutralelemente der Multiplikation von links und von rechts dieselben sind. In beiden Fällen ergibt sich die 2x2 Einheitsmatrix.

$$N_{mult-L} = N_{mult-R} = \begin{pmatrix} x & y \\ z & u \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Allgemein besitzt die nxn Einheitsmatrix auf der Diagonalen lauter Einsen und außerhalb nur Nullen. Sie ist gleichzeitig das Links- und das Rechts-Neutralelement der Matrixmultiplikation. Die Form der Einheitsmatrix unterstreicht die Bedeutung der Diagonalen.

1.3.3 Einfache Beispiele von linearen Gleichungssystemen

Formulieren von linearen Gleichungssystemen in Matrizenform

Die traditionelle Art der Schreibweise von linearen Gleichungssystemen führt die einzelnen Gleichungen als Zeilen ein, wobei die Unbekannten als Faktoren auf die Zeilen verteilt sind:

$$\left| \begin{array}{rcrcrcr} 5 \cdot x & + & & & 2 \cdot z & = & 6 \\ & & y & - & z & = & 5 \\ 3 \cdot x & + & 2 \cdot y & & & = & 12 \end{array} \right|$$

Dies entspricht nach den Regeln der Matrizenmultiplikation der Matrizengleichung:

$$\begin{pmatrix} 5 & 0 & 2 \\ 0 & 1 & -1 \\ 3 & 2 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 6 \\ 5 \\ 12 \end{pmatrix}$$

Dabei ist darauf zu achten, dass eine in der Zeile fehlende Unbekannte zu einer Null in der Matrix führt, und dass bei einer Unbekannten ohne Vorfaktor eine Eins eingesetzt wird.

Die Lösung dieser Gleichung mit MATLAB erfolgt durch Definieren der Matrix, sowie des Vektors der rechten Seite (Spaltenvektor!) und anschließende Auflösung der Matrizengleichung $A \cdot x = b$ durch $x = A^{-1} \cdot b$, wobei die Multiplikation mit A^{-1} von links in MATLAB mit dem **Links-Divisionsoperator** „\“ formuliert wird:

```
A = [ 5 0 2; 0 1 -1; 3 2 0]   ;    b = [ 6 5 12]'
x = A \ b
```

Die gefundene Lösung kann bei jedem Gleichungssystem durch Einsetzen der Unbekannten in die Ausgangsgleichung geprüft werden:

```
btest = A*x
```

Einige Beispiele von linearen Gleichungssystemen

Lösen Sie die folgenden Beispiele mit MATLAB, indem Sie das oben beschriebene Vorgehen anwenden:

13–1

$$\left|\begin{array}{rcrcrcr} 3\cdot x & + & 2\cdot y & + & z & = & 33 \\ & & y & - & z & = & 9 \\ & & y & + & z & = & 11 \end{array}\right|$$

13–2

$$\left|\begin{array}{rcrcrcrcr} x_1 & - & x_2 & & & & & = & 4 \\ & & x_2 & - & x_3 & & & = & 2 \\ & & & & x_3 & - & x_4 & = & 1 \\ x_1 & & & & & + & x_4 & = & 9 \end{array}\right|$$

13–3

$$\left|\begin{array}{rcrcrcrcr} x_1 & - & x_2 & + & x_3 & - & 3\cdot x_4 & = & 0 \\ & & x_2 & - & x_3 & + & 2\cdot x_4 & = & 4 \\ & & & & x_3 & - & x_4 & = & 3 \\ & & & & & & x_4 & = & 2 \end{array}\right|$$

Das dritte Beispiel ergibt zufällig als Matrix eine Dreiecksmatrix. Wie Sie bei der eingehenden Diskussion der Lösung von Gleichungssystemen sehen werden, wäre diese durch Rückwärtseinsetzen sofort lösbar.

In der früheren Schulzeit wurde Ihnen wahrscheinlich das bekannte Eliminationsverfahren zur Lösung von linearen Gleichungssystemen erklärt: Die Gleichungen werden dabei paarweise kombiniert, um eine Unbekannte nach der anderen zu eliminieren. Dies entspricht im Wesentlichen dem Gauß-Algorithmus der im 3. Kapitel eingehend behandelt wird. Bei diesem Verfahren wird die vorgegebene Matrix zuerst in eine Dreiecksform überführt. Die Schlussphase der Lösung besteht dann aus dem Rückwärtseinsetzen: Beginnend mit der untersten Zeile ist eine Unbekannte nach der anderen ganz einfach bestimmbar, indem die Werte aller bereits berechneten Unbekannten schrittweise in den weiter oben liegenden Zeilen eingesetzt werden.

1.3.4 Matrizen zur Darstellung von Daten

Die häufigste Art, Daten in Zahlenform übersichtlich zu präsentieren, ist die Tabellenform. Um einen Überblick über die tabellierten Daten zu erhalten, werden daraus auch häufig grafische Darstellungen abgeleitet. Tabellen bilden auch die wesentlichen Bestandteile in allen relationalen Datenbanken.

Jede Tabelle kann im Prinzip als Rechtecksmatrix betrachtet werden. Diese Interpretation passt allerdings bei Tabellen mit Texteinträgen nicht mehr zur Grundeigenschaft einer Matrix, dass nämlich alle Elemente von gleicher Art sein sollten. Eine nx2 Matrix, bei der die eine Spalte die Artikelnummer darstellt und die andere den Preis, ist kein geeigneter Fall für eine Matrix. Falls in der Artikelnummer Buchstaben vorkommen, kann die Tabelle nämlich gar nicht als Matrix gespeichert werden.

Beobachtungsreihen

Als Beispiel kann das Verkehrsaufkommen an den verschiedenen Tagen in einem Jahr dienen. Diese Daten können in einer 53x7 Matrix zusammengefasst werden. Alle Elemente sind Verkehrs-Frequenzen an je einem Tag, mit dem Wochentag und der Wochennummer als Indizes. Um diese Daten als echte Matrix darstellen zu können, müssen allerdings die erste und die letzte Zeile durch Daten aus dem Vorjahr und dem nachfolgenden Jahr ergänzt werden.

Beobachtungreihen von Sonnenscheindauer, Regenmenge, Temperatur etc. bilden für jeden einzelnen Ort je einen Vektor. Oft werden die gleichartigen Werte von verschiedenen Orten nebeneinandergestellt und so in einer Rechtecksmatrix zusammengefasst. In vergleichbarer Art können beliebige Messreihen von technischen, physikalischen, biologischen, geografischen oder demografischen Untersuchungen in Tabellen zusammengestellt werden. Bei solchen Daten stehen beider Verarbeitung vor allem Mittelwertbildungen und die Bestimmung von statistischen Streumaßen (siehe Kapitel 8) im Vordergrund.

Klassische Tabellenkalkulation

Für die übersichtliche Darstellung der Finanzbewegungen eines kleinen Geschäftes, für die private Buchhaltung und für die Bewertung eines Aktienpaketes, der in den USA üblichen Form des Sparens für das Alter, bieten Tabellenkalkulationsprogramme seit ihrem Aufkommen eine leicht bedienbare Berechnungshilfe. Die Programme zur Tabellenkalkulation haben deshalb in den 80er Jahren zur großen Verbreitung von einfachen PCs entscheidend beigetragen. Die häufigste Verarbeitung dieser Art von Daten ist die Bildung von Zeilen- und Spaltensummen.

Distanztabellen

Ein weiteres Beispiel für Daten in Matrizenform sind Distanztabellen, allerdings ohne die Anschriften (die ja gar keine zahlenförmigen Datenbestandteile sind). Bei Distanztabellen wird üblicherweise nur eine Hälfte der Matrix, eine untere Dreiecksmatrix angegeben, weil ja die Distanz von Zürich nach Basel dieselbe ist wie von Basel nach Zürich (zumindest im geografischen Sinn!).

km	Ba	Be	C	G	Lg	Lz	SG	Si	Z
Basel	*								
Bern	100	*							
Chur	210	247	*						
Genf	260	164	407	*					
Lugano	259	271	141	355	*				
Luzern	88	111	147	271	159	*			
St. Gallen	172	209	96	369	232	123	*		
Sion	249	153	203	154	153	184	273	*	
Zürich	88	125	122	285	191	56	88	274	*

Genau genommen besteht eine solche Distanzmatrix aus den Absolutwerten einer antisymmetrischen Matrix. Aus dieser Betrachtungsweise zeigt sich auch, dass in der Diagonale nur die Werte 0 erlaubt sind.

Ein Beispiel einer Distanzmatrix ohne Absolutwertbildung zeigt die Matrix der Höhendifferenzen. Bei dieser ist klar, dass die Höhendifferenz Matterhorn–Zermatt positiv ist während die umgekehrte Differenz Zermatt–Matterhorn eigentlich als negativ eingesetzt werden sollte. Man erhält so die Bedingung für eine antisymmetrische Matrix $d_{jk} = -d_{kj}$ (bzw. $D^T = -D$). Gleichzeitig ergibt sich eine gute Illustration der Tatsache, dass die Diagonalelemente einer antisymmetrischen Matrix Null sein müssen. Null ist die einzige Zahl die beim Vorzeichenwechsel gleich bleibt: $d_{kk} = -d_{kk}$ hat nur die Lösung $d_{kk} = 0$.

km	D	M	G	Z	B
Dufourspitze	0	−156	−814	−3014	−3956
Matterhorn	156	0	−658	−2858	−3800
Gornergrat	814	658	0	−2200	−3142
Zermatt	3014	2858	2200	0	−942
Brig	3956	3800	3142	942	0

Übung 13–4, Erstellen einer Distanztabelle Überlegen Sie sich, mit welchen Befehlsfolgen von MATLAB-Befehlen Sie die obenstehende Differenzenmatrix erhalten, wenn Sie vom Vektor der Höhenwerte `v = [ 4634 4478 3820 1620 678 ]'` ausgehen!

1.4 Einfache grafische Darstellungen mit MATLAB

Neben dem Arbeiten mit Matrizen und Vektoren bietet MATLAB auch einfach zu benutzende Möglichkeiten für grafische Darstellungen. Mit solchen leicht erstellbaren Grafiken sollen die mathematischen Vorgänge illustriert werden, um das Verständnis für die oft abstrakten Zusammenhänge zu erleichtern.

1.4.1 Funktionsdarstellungen

In MATLAB baut eine Darstellung von Funktionen grundsätzlich auf **Tabellen** auf. Die Normal-Eingabe für einen x-y-Plot ist also ein Paar von Spaltenvektoren. Die Eingabe eines Paars von Zeilenvektoren funktioniert für die Erstellung einer Grafik ebenfalls. Die Kombination von einem Zeilen- und einem Spaltenvektor als Eingabe führt dagegen zu einer Fehlermeldung.

Falls statt eines Vektors eine Matrix in die Funktion `plot(.)` eingegeben wird, erfolgt jedoch die konsequente Anwendung des Spaltenvektor-Prinzips, und jeder Einzelne der darin enthaltenen Spaltenvektoren wird als separate Funktion interpretiert.

Beim Aufruf der Funktion `plot(.)` mit einem einzigen Spaltenvektor- oder Matrixparameter wird diese Eingabe als Ordinate betrachtet und automatisch die Zeilennummer als Abszisse dazugenommen.

Anhand von einigen direkten Versuchen im interaktiven Arbeiten mit MATLAB lassen sich die verschiedenen Plot-Optionen viel leichter voneinander unterscheiden:

Plot einer Parabel

```
x = -5:5
y = x.^2
plot(x,y)
```

Dies sind der Einfachheit halber Zeilenvektoren (mit dem Befehl `whos` leicht zu kontrollieren). Mit den entsprechenden Spaltenvektoren

```
xv = x'
yv = y'
plot(xv,yv)
```

sollte es genauso funktionieren.

Vergleichen Sie nun die obige Variante mit der hier folgenden – wo liegt der Unterschied?

```
plot(y)
```

In den obigen Beispielen wurden zwei neue Operatoren angewandt:

- der Operator „`:`“ in `-5:5` produziert eine Aufzählung: es werden die 11 Zahlenwerte **von -5 bis und mit 5** definiert (mit der Schrittweite 1, falls wie hier kein weiterer Doppelpunkt mit einer Schrittweitenangabe dazwischen eingegeben wurde).
- der Operator „`.^2`“ wird zum Quadrieren der einzelnen Komponenten des Vektors `x` benötigt, weil `x^2` das Quadrieren von `x` mit einer Matrixmultiplikation `x*x` bedeuten würde, was in diesem Fall nicht definiert ist. Allgemein bewirkt ein Punkt, der einem Operator vorangestellt wird, die elementweise Ausführung der Operation.

Markieren der Tabellenwerte

Oft möchte man wissen, welche Tabellenwerte die gezeichnete Funktion definieren. Dazu können die eingegebenen Punkte markiert werden mit einem zusätzlichen Parameter vom Typ Characterstring (Zeichenkette, d. h. in einfache Apostrophzeichen eingeschlossene Buchstabenfolgen):

`plot(y,'-+')` bzw. im vorherigen Beispiel `plot(x,y,'-+')`.

In diesen Beispielen sind die zwei in die Apostrophe eingeschlossenen Zeichen ein Minus, um eine Verbindungslinie zu verlangen, und ein Plus, um ein Plus-Zeichen als Markierung zu erhalten. Experimentieren Sie mit anderen Markierzeichen, die Sie mit dem Befehl `help plot` leicht herausfinden können!

Parabel mit feinerer Auflösung

Besonders, wenn man den Darstellungsbereich einschränkt, z. B. mit dem Befehl `axis([-3 3 0 9])`, allgemein:

`axis([xmin,xmax,ymin,ymax])`, nach dem Aufruf von `plot()`,

wird die obige Parabel ein wenig eckig. Abhilfe ist einfach (Angabe einer Schrittweite kleiner als Eins in der Aufzählung):

```
xf = -10:0.02:10
yf = xf.^2
plot(xf,yf)
```

Zum Vergleich sind die drei Varianten des Parabelplots nebeneinandergestellt:

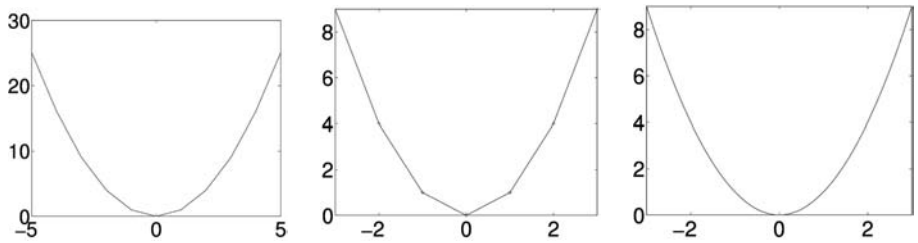

Abbildung 1.5: Parabelplots mit automatischer und vorgegebener Wahl des Anzeigebereiches, mit und ohne Markierungen sowie mit grober und feiner Tabellierung.

Demonstration des kolonnenweisen Zeichnens

Die drei Zeilenvektoren `z1, z2, z3` haben alle die Länge 11 und stellen die Funktionen $z1 = x$, $z2 = x^2$ und $z3 = x^3$ dar. Sie können auf einfache Weise zu einer 3x11 Matrix zusammengefügt werden (jeder Vektor bildet eine Zeile):

```
z1 = 0:0.1:1
z2 = z1.^2
z3 = z1 .^3
P= [ z1 ; z2 ; z3 ]
plot(P)
```

Wenn man diese Matrix als einzigen Parameter der Funktion plot() übergibt, so erscheint nicht das gewünschte Bild. Die einzelnen geplotteten Funktionslinien entsprechen nämlich den Spalten dieser Matrix, also den Werten von x, x^2 und x^3 an je einer Stelle. Unsere zusammengehörenden Funktionswerte stehen aber in den Zeilen der Matrix P.

Bei der Übergabe einer Matrix als Parameter in `plot()` gelten in MATLAB allgemein die Werte in derselben Spalte als zu derselben Funktion gehörend. Diese Konvention beruht auf dem häufigen Gebrauch von langen einzulesenden Messwerttabellen mit vielen Zeilen und nur einigen wenigen Spalten.
Die Rückkehr zum Plotten einer linearen, quadratischen und kubischen Funktion, ist mit `plot(P')`, durch Transponieren von P, einfach zu bewerkstelligen.

Eine ausführliche Variante, mit der einzelne Zeilen aus einer Matrix ausgewählt und geplottet werden können, bietet die Auswahl mit dem Aufzählungsoperator „`:`“ (auch impliziter Schleifenoperator genannt). Dieser wird im Beispiel an der zweiten Indexposition eingesetzt und bedeutet *alle* an dieser Stelle möglichen Indexwerte.

```
plot(P(1,:), P(1,:))   ;   hold on   ;
plot(P(1,:), P(2,:))   ;   plot(P(1,:), P(3,:))
```

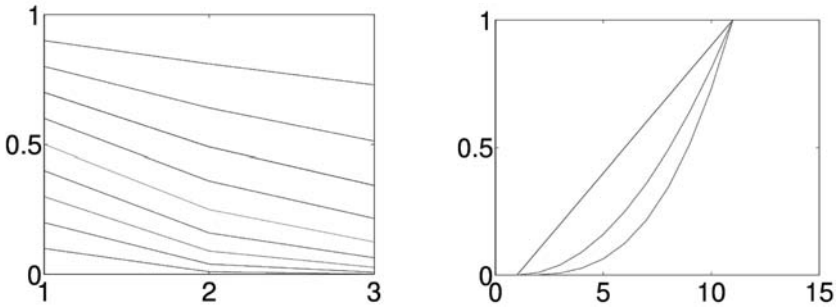

Abbildung 1.6: Demonstration der Konvention, dass beim Zeichnen der Werte einer Matrix die Spalten als Funktionsvektoren gelten. Links: `plot(P)`, rechts: `plot(P')`.

Sinus- und Kosinusfunktionen

Die Sinus- und Kosinusfunktionen $\sin(x)$ und $\cos(x)$ sind in der Mathematik, und damit auch in MATLAB mit dem Argument x im Bogenmaß definiert. Die meisten Leser sind aber viel besser vertraut mit der Gradeinteilung. Die Umrechnung ist dank der vordefinierten Zahl Pi einfach. (Es ist zu beachten, dass in MATLAB die Zahl als `'pi'` mit zwei Kleinbuchstaben geschrieben werden muss!) Eine Alternative wäre die Verwendung der selbst erstellten Funktionen `cosdeg()` und `sindeg()`. (Siehe vorherigen Abschnitt 1.3.1, S. 23.)

```
degrad = pi/180
w = 0:5:360)   ;     wb = w*degrad   ;
si = sin(wb)   ;     co = cos(wb)
plot(wb,si)   ;     hold on
plot(wb,co,'r')   ;     hold off
```

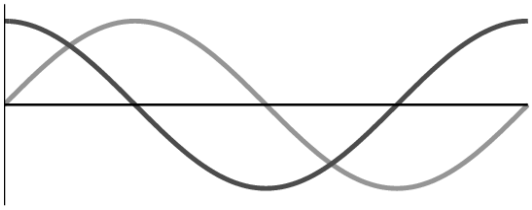

Abbildung 1.7: Kosinus- und Sinusfunktion über eine Periode der Länge 2π.

1.4.2 Polygone, Kreise, Sterne

Einen Kreis zeichnen

Wenn die x-Werte der Punkteschar und die y-Werte je separat als Funktion der variierenden Größe (in diesem Beispiel des Winkels) definiert werden, so spricht man von einer Kurve in Parameterdarstellung. Der Parameter, auch Laufparameter genannt, ist in diesem Fall der Winkel. Auf diese Weise lässt sich mit den oben eingeführten Kosinus- und Sinusvektoren ein Kreis zeichnen:

```
degrad = pi/180 ;
w = 0:5:360 ;
si = sin(w*degrad) ;
co = cos(w*degrad) ;
plot(co,si)
```

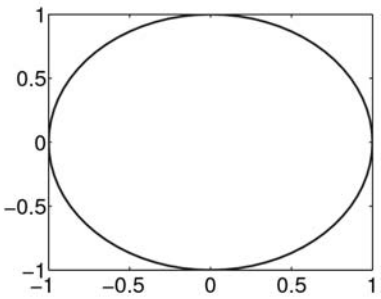

Abbildung 1.8: Erster Versuch, mit MATLAB einen Kreis zu zeichnen.

Leider ist nur eine Ellipse herausgekommen! Experimentieren Sie mit dem Befehl
`axis equal` bzw. mit dem Befehlspaar
`axis([-1.2 1.2 -1.2 1.2]);`
`axis square` (nach dem Aufruf von `plot()` einfügen)!

Diese Art, einen Kreis grafisch darzustellen, beruht auf der Definition von Kosinus und Sinus als x- und y-Koordinaten eines Punktes auf dem Einheitskreis. Wem an diesem Kreis doch noch zu stark die Ecken sichtbar sind, der kann dasselbe mit einer feineren Winkeleinteilung versuchen, z. B. mit `w = 0:1:360` und anschließend dieselben Befehle verwenden wie oben.

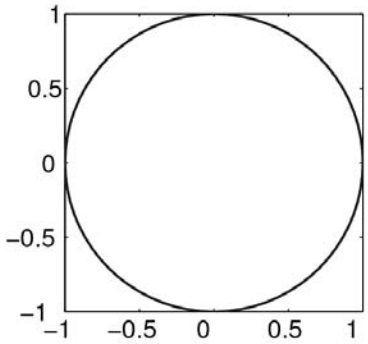

Abbildung 1.9: Ein mit MATLAB gezeichneter Kreis mit Anpassung der Achsenskalierung.

Der Winkelvorschub zwischen den Tabellenwerten bestimmt also die Anzahl der Ecken. Die Kreise werden bei diesem Verfahren nur durch reguläre Polygone mit sehr vielen Ecken angenähert. Durch Ansetzen von absichtlich großen Werten für den Winkelvorschub kann man also beliebige reguläre Vielecke zeichnen.

Reguläre Polygone

Die Linienzüge zum Zeichnen einer geschlossenen Figur enthalten immer einen Punkt mehr als die Anzahl der Ecken angibt; man muss ja den letzten Punkt wieder mit dem Anfangspunkt verbinden. Kontrollieren Sie diese Tatsache im Folgenden mit der Funktion `whos`.

Als erstes soll die Anzahl Ecken vorgegeben werden: so zum Beispiel `neck = 4`.
Daraus ergibt sich der Winkelvorschub und damit die Tabellen der Winkel, der x- und der y-Koordinaten für die zu zeichnenden Punkte:

```
w = 0:360/neck:360 ; degrad = pi/180;
x = cos(w*degrad) ; y = sin(w*degrad) ; plot(x, y)
```

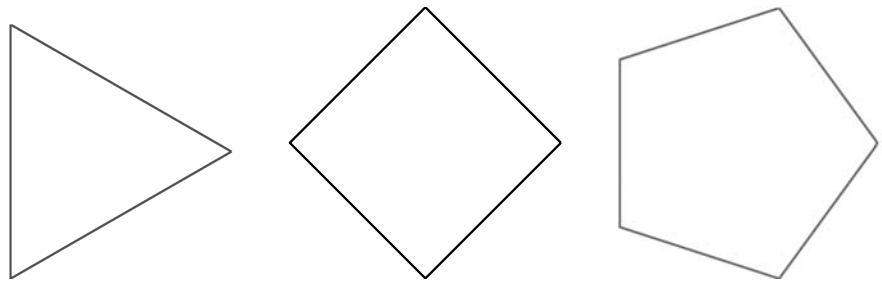

Abbildung 1.10: Durch Kosinus- und Sinustabellen gezeichnete Polygone.

Eine kleine Modifikation der MATLAB Befehle, die Einführung eines beliebigen Radiuswertes statt des Wertes Eins für den Einheitskreis und der Einbezug eines anderen Startwinkels statt wie bisher 0° erlaubt es, alle möglichen regulären Polygone zu zeichnen.

Übung 15–1 Zeichnen Sie die regulären Polygone vom 3- bis 12-Eck mit einer Ecke nach oben zeigend! Einzeln, und mehrere übereinander mit verschiedenen Farben.

In einer einfachen Variante sind alle im Einheitskreis einbeschrieben. Etwas anspruchsvoller zu realisieren ist die Bedingung, dass alle die Seitenlänge 1 haben sollen. Dazu muss der Umkreisradius aus der gewünschten Seitenlänge und der Anzahl Ecken bestimmt werden.

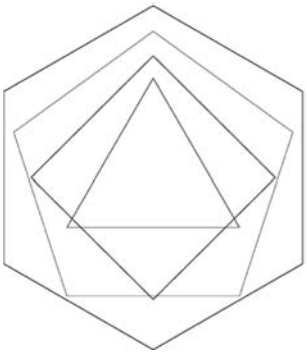

Abbildung 1.11: Konzentrische reguläre Polygone mit fester Seitenlänge.

Damit ist unbestreitbar die Erstellung einer attraktiven Mandala-Figur aus einfachsten mathematischen Ausgangsbedingungen gelungen!

Sterne zeichnen

Der Schritt vom regulären Fünfeck zum Pentagramm, dem fünfzackigen Stern, der in einem Zug gezeichnet werden kann, ist nur ganz klein. Von jeder Ecke aus wird statt der nächsten die übernächste mit dem doppelten Winkelvorschub angepeilt. Dafür muss der Kreisumfang zweimal durchlaufen werden. Wir erhalten also (mit Start oben, bei 90°):

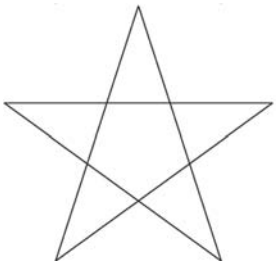

Abbildung 1.12: Pentagramm – in einem Zug gezeichneter fünfzackiger Stern.

```
degrad = pi/180 ; w = (0:2*72:2*360) + 90 ;
si = sin(w*degrad) ; co = cos(w*degrad)
plot(co,si)
```

Dieses Verfahren funktioniert für alle Sterne mit ungerader Eckenzahl. Wenn ein Stern eine gerade Anzahl von Ecken aufweist, so muss dieser mit zwei separaten Linienzügen gezeichnet werden, die um den einfachen Vorschubwinkel zueinander verschobene Startwinkel haben. Jeder der beiden hat den doppelten Winkelvorschub, aber dafür nur einen Umgang um den Kreis. Testen Sie das untenstehende Skript mit neck = 6 und mit neck = 8

```
neck =6 ; degrad = pi/180 ; w = (0:2*360/neck*360) + 90 ;
xa = cos(w*degrad) ; ya = sin(w*degrad)
xb = cos((360/neck+w)*degrad) ; yb = sin((360/neck+w)*degrad)
plot(xa,ya,'k', xb, yb,'k')
```

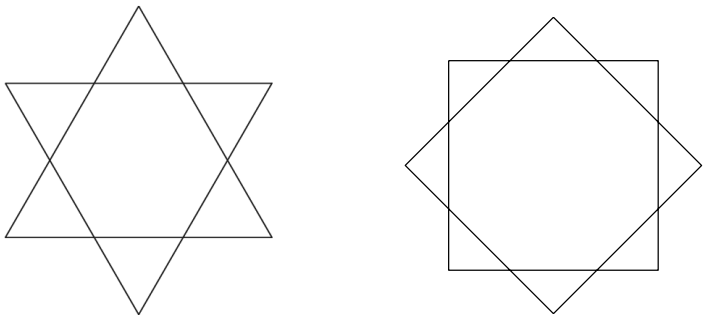

Abbildung 1.13: Sechszackige und achtzackige Sterne mit Liniengrafik aus je zwei Streckenzügen.

1.4.3 Flächen malen

Mit dem Aufruf `fill(xlin,ylin,farbdef)` malt MATLAB eine ausgefüllte Farbfläche. Dabei sind die anzugebenden Parameter die Vektoren der x- und der y-Werte eines geschlossenen Linienzuges, sowie als dritten Parameter eine Definition der zu verwendenden Farbe mit einer Zeichenkettenvariablen.

Als erstes ganz einfaches Beispiel soll ein Rechteck gezeichnet werden:

Abbildung 1.14: Mit Farbe ausgefülltes Rechteck.

```
xre = [ 0 12 12 0 0]
yre = [0 0 9 9 0]
fill(xre, yre, 'g')
```

Ein kurzer Test, ob das Pentagramm wohl auch als auszumalende Figur funktioniert?

```
degrad = pi/180 ; w = (0:2*72:2*360) + 90 ;
si = sin(w*degrad) ; co = cos(w*degrad)
fill(co,si,'g')
```

Abbildung 1.15: Pentagramm, übergeben an die Funktion `fill()`.

Die Figur ist nicht ganz mit Farbe ausgefüllt, das innenliegende Fünfeck ist leer. Dieses Schema, nach dem nur diejenigen Flächen gefüllt werden, welche durch eine ungerade Anzahl Linien vom Außengebiet getrennt sind, nennt man “even odd” Regel. Es ist die vorherrschende Methode beim Flächenzeichnen und findet auch in MATLAB Anwendung.

Um doch noch Sterne mit beliebiger Anzahl von Ecken als gefüllte Flächen zu zeichnen, werden abwechselnd Punktkoordinaten von einem äußeren Kreis und einem inneren Kreis in die x- und y-Vektoren eingetragen. Dies erreicht man durch Eingeben von zwei Zeilen in eine Matrix und anschließendes Umdefinieren in einen einzigen Zeilenvektor. Solche Uminterpretationen von Matrixdimensionen erreicht man mit der Funktion `reshape(M,nzeilneu,nspaltneu)`.

```
% FILLSTAR Skript zum Fuellen eines Sterns, es braucht
% vordefinierte Werte NECK, ROUT, RIN, WSTARTDEG, FILLCOL
%  zusaetzlich Angabe des Zentrums  XZENT, YZENT
wstartrado = WSTARTDEG*pi/180;
wstartradi = wstartrado + 2*pi/2/NECK;
wout = 0+wstartrado:2*pi/NECK:2*pi+wstartrado;
win= 0+wstartradi:2*pi/NECK:2*pi+wstartradi;
% x- und y-Koordinatenvektoren aussen und innen
xo = ROUT* cos(wout) + XZENT;
yo = ROUT*sin(wout)+ YZENT;
xi = RIN* cos(win)+ XZENT;
yi = RIN*sin(win)+ YZENT;
% in Matrix kombinieren und Abfolge o-i-o-i erzeugen
tmpmatx = [ xo; xi];
tmpmaty = [ yo; yi];
xf = reshape(tmpmatx,2*NECK+2,1)
yf = reshape(tmpmaty,2*NECK+2,1)
fill(xf(1:2*NECK+1),yf(1:2*NECK+1),FILLCOL)
axis equal
```

Abbildung 1.16: Vollständig gefüllter fünfzackiger Stern.

Mit dem M-File Skript ’fillstar.m’ ist nun das Zeichnen von beliebigen gefüllten Sternen möglich.

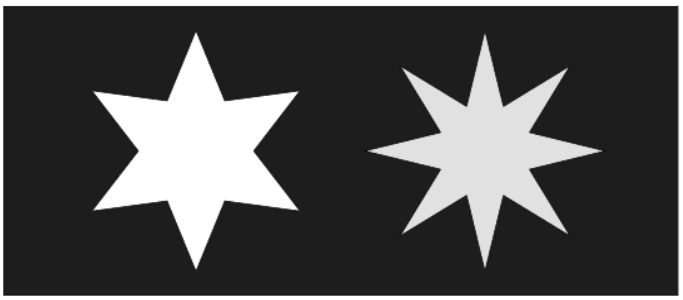

Abbildung 1.17: Gefüllte sechs- und achtzackige Sterne auf blauem Hintergrund.

1.5 Übersicht über die wichtigsten Grundbefehle in MATLAB

1.5.1 In MATLAB definierte Operatoren und Grundbefehle

Die meisten der im Folgenden zusammengestellten MATLAB-Operatoren und -Grundbefehle wurden bereits im Textzusammenhang erläutert. Sie werden hier zur besseren Übersicht noch einmal aufgeführt und kurz in den Gesamtzusammenhang gestellt.

Grundfunktionen

- Die Eingabe eines Kommandos gilt am Zeilenende als abgeschlossen, wenn nicht eine Markierung mit drei Punkten . . . am Zeilenende die **Fortsetzung** verlangt.
- **Kommentare** können hinter einem **Prozentzeichen %** eingefügt werden.
- Nach jedem Berechnungsschritt werden die momentanen Zwischenresultate auf dem Bildschirm ausgegeben, außer man **unterdrückt die Ausgabe** mit einem **Semikolon am Zeilenende: ;** .
- Eine Kontrolle der Dimensionen der durch einen Befehl generierten Variablen ist immer mit dem Befehl `whos` (who-size) möglich.
- Die einfachere Variante dieses Abfragebefehls, `who` , zeigt eine Liste an mit allen momentan belegten Variablennamen.
- Mit `clear ‘‘varnam’’` kann eine einzelne, und mit `clear` können alle Variablen gelöscht werden.

Die Standardoperatoren

Wie im normalen Zahlenrechnen sind in MATLAB die Standardoperatoren der Grundrechenarten definiert: $+$, $-$, $*$, $/$.

Für Matrizen bedeutet der Operator $*$ die Matrizenmultiplikation, während der Operator $/$ für die Division durch eine Matrix von rechts steht; also bezeichnet A/B die Operation $A * B^{-1}$.

Neue Operatoren in MATLAB

Damit auch eine Matrizendivision von links her beschrieben werden kann, die sich ja von der Matrizendivision von rechts her unterscheidet, wird in MATLAB ein neuer Operator eingeführt. Die Bedeutung des neuen Operators der **Links-Division** $\backslash$ ist: $A\backslash B = A^{-1} * B$.

Zusätzlich zu den Elementar-Operationen gibt es auch den **Potenzierungsoperator** ^ .

Die Multiplikations-, Divisions- und Potenzoperatoren sind alle für Matrizen nach den Regeln der Matrizenmathematik definiert, deshalb wird für die Fälle, bei denen doch eine elementweise Operation durchgeführt werden soll, eine Variante zu jedem dieser Operatoren benötigt: **die Punkt-Operatoren** – $.*$, $./$, $.\hat{}$, aber auch $.\backslash$ – verlangen die elementweise Ausführung des dem Punkt folgenden Operators.

Der Transpositionsoperator

Eine in der Matrizen- und Vektorrechnung sehr häufig benötigte Operation ist das Transponieren einer Matrix bzw. eines Vektors (das Spiegeln der Matrix an ihrer Diagonalen bzw. das Umsetzen eines Vektors aus der Spaltenform in die Zeilenform oder umgekehrt). Die Mathematik verwendet als Operatorsymbol dafür den hochgestellten Buchstaben „T“, also A^T für die zu A transponierte Matrix.

In MATLAB wird als Transpositionsoperator das **Apostrophzeichen** (**'**) benutzt: `A'` gibt also in MATLAB die Transponierte zu `A` an. Dass damit nicht die Ableitung gemeint sein kann, die man aus der Analysis her kennt, ist vom Anwendungs-Umfeld her klar. Das Apostroph wirkt auf eine Matrix, und nicht auf eine Funktion.

Achtung! Genau genommen bedeutet in MATLAB das Apostroph „transponieren und gleichzeitig komplex konjugieren“ (Fachausdruck: Adjungieren), was für reelle Matrizen identisch ist mit Transponieren. Deshalb wird in diesem Kurs normalerweise nur das einfache Apostroph verwendet. Nur Transponieren, ohne Komplex-Konjugation einer komplexen Matrix kann mit dem entsprechenden Punkt-Operator verlangt werden, also mit `.'`.

Vergleichsoperatoren

Die üblichen Vergleichsoperatoren vergleichen zwei Matrizen mit identischen Dimensionen elementweise miteinander und liefern als Resultat eine Matrix mit gleicher Dimension und mit 1 für “true” und 0 für “false” an den Positionen der Zahlen.

Die genauen Definitionen sind:

$'<'$, $'<='$, $'>'$, $'>='$, $'=='$, $'\sim='$

Der weitaus häufigste Gebrauch dieser Vergleichsoperatoren dient dem Vergleich von skalaren Ausdrücken bei der Formulierung von Bedingungen (if-statements), wenn man MATLAB als Programmiersprache einsetzt.

Eine andere nützliche Anwendung der logischen Operatoren besteht in der Auswahl von Elementen aus langen Vektoren (z. B. aus Messreihen).

Als Beispiel dient die Auswertung von Strahlpositionsmessungen, gespeichert in einem Vektor `p`. Die Messelektronik liefert ein Signal zwischen 0 und 8 Volt. Der Wert von 10 Volt wird als Flag benutzt, um anzuzeigen, dass für eine zuverlässige Messung zuwenig Strahlstrom vorhanden war. Die MATLAB-Befehle

```
iscurrenton = p <= 8
pselect = p(iscurrenton)
```

liefern die gewünschte Auswahl. Der logische Array 'iscurrenton' enthält nur 0 und 1. Wenn ein solcher Array statt des Indexes in der Klammer bei einem Vektor eingesetzt wird, so werden alle Werte ausgewählt, an deren Platz eine Eins steht.

Logische Operatoren

Die Standardoperatoren für logische Verknüpfungen in MATLAB sind AND: '&' , OR: '|' und der unäre Operator NOT '$\sim$' . (Unär bedeutet, dass dieser Operator nicht zwei Objekte miteinander verknüpft, sondern nur auf das eine, nachfolgende Objekt wirkt.) Für die logische Funktion XOR existiert kein Operator, diese Verknüpfung kann aber durch einen Funktionsaufruf `xor(A,B)` realisiert werden.

1.5.2 Das Definieren von Zahlen, Matrizen und Vektoren

Zahleneingabe, Zahlenausgabe

Zahlen können eingegeben werden als ganze Zahlen, Dezimalbrüche (mit Dezimal**punkt**) oder in der üblichen wissenschaftlichen Exponentialnotation `15.3e-08` (mit der Bedeutung $15.3 * 10^{-8}$). Im Fall von positiven Exponenten ist die Eingabe des '+'-Zeichens beim Exponententeil fakultativ. `0.32e6` und `0.32e+6` sind also gleichbedeutend.

Komplexe Zahlen können in logisch einleuchtender Weise durch Angabe einer Summe von zwei Zahlen mit dem Vorfaktor 'i' oder 'j' bei einer der Zahlen eingegeben werden, wie z. B. `1.0+i*2` oder `j*20` . Noch ein wenig kompakter ist die Eingabe durch direktes Anfügen des i oder j hinter dem Imaginärteil, ohne Multiplikationsoperator, wie in `1+2i` oder `20j`.

Jede Zahl, die über den aktuellen Berechnungsschritt hinaus wieder verwendet werden soll, muss einem **Variablennamen** zugewiesen werden. Variablennamen sind empfindlich auf Groß- und Kleinschreibung und müssen mit einem Buchstaben beginnen. An anderen Stellen eines Namens sind auch Zahlen und der Unterstrich _ erlaubt. Die in Filenamen manchmal vorkommenden Bindestriche kann MATLAB in Variablennamen nicht erlauben, da diese als Minuszeichen gelesen werden.
Die maximale Länge für einen Variablennamen beträgt 31 Zeichen.

Die Zahlenausgabe wird mit dem `format`-Befehl vorgewählt und bleibt danach bis auf einen nächsten `format`-Befehl gültig. Die wichtigsten Optionen von `'format'` sind:

```
format short   (der Standard)
format short e  (Exponentialschreibweise)
format short g  (Exponentialschreibweise nur wenn nötig)
format long ,       format long e ,       format long g
```

und für kommerzielle Anwendungen:

```
format bank   (zwei Stellen nach dem Dezimalpunkt:   500.25 )
```

Ein spezielles Format, das beim Verfolgen von Berechnungen interessant sein kann, ist `format rat` (Darstellung wenn möglich als rationale Zahl, also als ganzzahliger Bruch).

Um einen Überblick über große Matrizen gewinnen zu können gibt es das `format +`, welches die Ausgabe auf einen Platz pro Zahl, `+`, `-` oder Leerzeichen reduziert.
Ein ähnliches Hilfsmittel, speziell für Matrizen mit vielen Nullen, ist die grafische Darstellung mit der MATLAB-Plot-Funktion `spy(M)`, welche an allen Postionen mit von Null verschiedenen Werten einen Punkt plottet.

Definition von Matrizen und Vektoren

Die Eingabe von Matrizen erfolgt durch Einschließen der Zahlen in eckige Klammern wie in: `M = [ 1 2 ; 3 4]`. Dabei können die Zahlen, die in der gleichen Zeile stehen, mit Leerzeichen oder mit Kommata getrennt werden. Das Weiterschalten der Zeile erfolgt mit dem Semikolon.

Ein normaler (Spalten-)Vektor kann entweder mit Hilfe des Transpositionsoperators wie in `v = [ 1 2 3]'` oder durch Verlangen einer neuen Zeile für jedes Element definiert werden, wie in `w = [ 1; 2; 3]` .

Die Elemente im Innern dieser eckigen Klammern können beliebige, in MATLAB definierte Objekte sein, also auch Matrizen oder Vektoren. Mit Leerzeichen oder Komma getrennt

werden die Objekte horizontal aneinandergefügt, mit Semikolon getrennt, vertikal übereinander angeordnet. Die beiden Funktionen des Aneinanderfügens in horizontaler bzw. vertikaler Richtung heißen 'horzcat' und 'vertcat'. Diese Namen erscheinen in den Fehlermeldungen, falls beim Aneinanderfügen die benachbarten Dimensionen nicht übereinstimmen.

So kann man z. B. aus den 2x2 Matrizen `Mx` und `My` eine 4x4 Matrix

`Mt = [ Mx zeros(2) ; zeros(2) My ]` herstellen.

Eine andere häufige Anwendung dieser Möglichkeit ist das Nebeneinandersetzen von Spaltenvektoren:

`Mv = [ v1 v2 v3 v4 v5 ]` z. B. mit `v1 = [1 2 5]'` etc.

Die Matrix der nebeneinandergestellten Vektoren `Mv` kann dann wie ein einzelner Spaltenvektor durch Multiplikation von links beliebigen Transformationen unterworfen werden. Damit werden alle enthaltenen Vektoren simultan transformiert.

Die leeren eckigen Klammern ohne eingefügtes Objekt stellen die leere Matrix dar. Dies kann benutzt werden, um Teile von Matrizen zu löschen:
Der Befehl `M(:,3) = []` entfernt z. B. die dritte Spalte aus `M`.

Ein hilfreiches Werkzeug für das Erstellen von Matrizen sind die Funktionen `zeros(n,m)`, `ones(n,m)`, sowie `rand(n,m)` und `randn(n,m)`, welche je eine Rechtecksmatrix erstellen und füllen (mit Nullen, Einsen, gleichverteilten bzw. normalverteilten Zufallszahlen).
Achtung! diese Funktionen können auch mit nur einem Parameter aufgerufen werden, dann erstellen sie eine quadratische Matrix mit den Dimensionen nxn. Für lange Zeilenvektoren also unbedingt die Zeilendimension 1 angeben wie in `zeros(1,10000)`.

Für die Definition einer Einheitsmatrix `I`, der Dimension n, gibt es die Funktion

`I = eye(n).`

Noch eine andere Möglichkeit zum Definieren von langen Zeilenvektoren, meist als Abszissenwerte von Linienplots, ergibt sich mit dem **impliziten Schleifenoperator ':'**

`x = 1:100` ergibt den Zeilenvektor mit den Zahlen von 1 bis 100.

`x = 0:0.01:2` ergibt den Zeilenvektor mit $0,\ 0.01,\ 0.02$ etc. bis 2.

1.5.3 Schleifen und Bedingungen

Mit MATLAB ist das Erstellen von eigentlichen Programmen möglich, dazu sind natürlich Befehle für Schleifen und zum Formulieren von Bedingungen erforderlich.

Einfache Schleifenkonstruktionen

Die Befehlsstruktur zum Programmieren einer einfachen Schleife ist:

```
for k = 1:n
    „Schleifenrumpf“
end
```

Das ist wirklich eine kompakte Formulierung, die dadurch möglich wird, dass die Zeilenstruktur bei diesem Programmstück eingehalten werden muss.
Vorsicht! Die Formulierung `1:n` unterscheidet sich von anderen Programmiersprachen! Dies ist eine häufige Fehlerquelle, indem man aus Gewohnheit `1,n` oder `1;n` schreibt, was sich meist erst viel später als Fehler bemerkbar macht.

Zur Programmierung von Schleifen ohne festgelegte Anzahl Durchläufe kann man in MATLAB die Schleife mit `while` konstruieren:

```
while „berechenbarer Ausdruck“
    „Schleifenrumpf“
end
```

Diese Schleife wird wiederholt, solange der berechenbare Ausdruck einen Wert verschieden von Null ergibt; bei Null wird abgebrochen.

Beide Arten von Schleifenkonstruktionen können mit dem Befehl `break` vorzeitig beendet werden. Durch den Befehl `continue` springt die Ablaufkontrolle von while- oder for-Schleifen auf den nächsten Durchlauf und überspringt so die verbleibenden Befehle bis zur zugehörigen end-Marke.

Programmieren von Bedingungen

Die Formulierung von Bedingungen ist ähnlich kurz:

```
if k == 1
    „Programmschritte bei erfuellter Bedingung“
elseif k < 1
    „Programmschritte bei erfuellter 2. Bedingung“
else
    „Programmschritte andernfalls“
end
```

Darin können das `elseif` und das `else` fehlen, wie in:

```
if k == 1
    „Programmschritte bei erfuellter Bedingung“
end
```

Beachten Sie das Fehlen eines “then” Schlüsselwortes! (Neue Zeile genügt.)

Alle end-Marken für `while`, `for`, `if` und `elseif` enthalten nur das einfache Wort ’`end`’. Daher empfiehlt es sich, bei komplexeren Programmen durch Kommentare und Einzüge die Zugehörigkeit der end-Marken zusätzlich hervorzuheben. Beachten Sie, dass das Schlüsselwort `elseif` ohne Leerzeichen geschrieben werden muss, sonst zählt es als separates `else` und `if`, was eine andere Struktur der end-Marken bedingt!

1.5.4 Mathematische Funktionen

Trigonometrische Funktionen

MATLAB kennt die gängigen Funktionen aus der Familie der trigonometrischen Funktionen wie:

```
sin(), cos(), tan(),
asin(), acos(), atan(),
```

wobei die Argumente immer auch Vektoren oder Matrizen sein können. Bei nicht skalaren Argumenten hat das Resultat dieselbe Struktur wie die eingegebenen Parameter.

Die Funktion `atan2(y,x)` braucht hingegen ein zueinander passendes Paar von identisch strukturierten Parametern.

Beachten Sie, dass MATLAB die kurzen Formen 'acos', 'asin' etc. verwendet und nicht die in vielen Mathematikbüchern verwendeten längeren Schreibweisen wie 'arccos'.

Hyperbolische Funktionen

Die hyperbolischen Funktionen wie
`sinh(), cosh(), tanh(),`
`asinh(), acosh(), atanh(),`
sind ebenfalls in der gleichen Weise definiert.

Weitere mathematische Basisfunktionen

Ohne ausführliche Erklärung seien die folgenden Funktionen erwähnt:
`sqrt(x)` nur Quadratwurzel, höhere Wurzeln mit $R^{(1/n)}$
`exp(z)` für e^z, auch für komplexe z.
`log(x) , log10(x) , log2(x)`
`sign()`

Funktionen für komplexe Zahlen

Für den Umgang mit komplexen Zahlen benötigt man die folgenden Funktionen:

`abs(z)` = Betrag der Zahl (auch für reelle Zahlen)
`angle(z)` = Argument der komplexen Zahl, (Winkel der Polarkoordinaten)
`real(z)` = Realteil von z
`imag(z)` = Imaginärteil von z
`conj(z)` = Komplex-konjugierte Zahl von z (= $\overline{z}$)

Funktionen für den Umgang mit ganzen Zahlen

`round(x)` = Runden auf nächstliegende ganze Zahl
`fix(x)` = Runden in Richtung 0
`floor(x)` = Runden in Richtung $-$ infinity
`ceil(x)` = Runden in Richtung $+$ infinity
`rem(x,m)` = 'remainder' = Restklasse Modulo m
`mod(x,m)` = 'modulus' = Restklasse Modulo m mit Vorzeichen von m
`gcd(a,b)` = 'greatest common divider' = ggT
`lcm(a,b)` = 'least common multiple' = kgV

1.5.5 Einige Hinweise zu Linienplots

Die einfachen Befehle `plot(), axis, hold on, hold off` und `subplot()` erlauben die Produktion einer großen Vielfalt von grafischen Darstellungen.
Das Grundprinzip beim Darstellen von Linienplots beruht immer auf Zahlenreihen, d. h. langen Vektoren, deren Zahlenwerte in einen Polygonzug umgesetzt werden.

Aufrufe von plot(v1) mit einem Datenparameter

Falls beim Aufruf wie bei `plot(v1)` nur ein einziger Vektorparameter eingefügt wird, so gelten diese Zahlen als Ordinaten (y-Werte) des zu zeichnenden x-y-Diagramms. Als zugehörige Abszissen (x-Werte) werden automatisch die Index-Nummern (1 bis Anzahl der Elemente von v1) eingesetzt.

Überrascht werden kann man, falls man nicht daran gedacht hat, dass der Vektor v1 komplexe Zahlen enthalten kann, dann werden nämlich die Imaginärteile versus die Realteile dargestellt, statt wie erwartet die Realteile versus die Indexnummer. Die Abhilfe ist aber einfach: man verlangt `plot(abs(v1))` oder `plot(real(v1))`.

Ist der einzige Datenparameter eine Matrix statt eines Vektors, so werden die Spalten als separate Funktionen gezeichnet, mit wechselnden Farben.

Aufrufe von plot(v1,v2) mit zwei Datenparametern

Der Aufruf von `plot(v1,v2)` mit zwei Vektorparametern ist der Normalfall, dabei werden die Werte von v2 als Ordinaten (y-Werte) gegen die Werte von v1 als Abszissen (x-Werte) aufgezeichnet. Verschiedene Fehlermöglichkeiten ergeben sich bei Fällen, in denen die Dimensionen von v1 und v2 inkompatibel sind. Die dabei resultierenden Fehlermeldungen sind jedoch meist gut verständlich, so dass der Fehler leicht korrigiert werden kann.

Im Fall, dass v2 kein Vektor sondern eine „echte“ Matrix ist, werden die Spalten von v2 je als separater Satz von y-Werten interpretiert und es werden soviele Kurven gezeichnet, wie die Matrix Spalten besitzt (mit variierenden Farben). In diesem Fall muss v1 ein Spaltenvektor mit derselben Zeilenzahl wie die Matrix v2 sein oder eine Matrix mit identischen Dimensionen. Wenn v1 und v2 beide Matrizen sind, so werden die zueinander passenden Paare von Spalten aus den beiden Matrizen als x- und y-Werte der verschiedenen Kurven interpretiert.

Der Zusatzparameter vom Typ Characterstring

Mit Aufrufen vom Typ `plot(v1,v2,'Stringparameter')`, also durch Einsetzen eines Zusatzparameters, der in Apostrophzeichen eingefügte Buchstaben enthält, kann der Linientyp (durchgezogen, punktiert, etc.) und die Farbe der zu zeichnenden Linie, sowie die Art der an den Datenpunkten gezeichneten Markiersymbole (Kreuz, Kreis etc.) bestimmt werden.

Für die Wahl der Farbe stehen die Buchstaben c, m, y (cyan, magenta , yellow), r, g, b (red, green, blue), sowie w, k (white, blac**k**) zur Verfügung.

Die Linientypen sind spärlicher: '-' für durchgezogen, '`--`' für gestrichelt, ':' für punktiert und '`-.`' für strich-punktiert.

Von den Markiersymbolen gibt es dagegen eine ganze Menge +, o, *, . ,x sind selbsterklärend, die anderen bedeuten 's' square, Quadrat, 'd' diamond, Spielkarten-Karo, 'p' pentagon, fünfzackiger Stern, 'h' hexagon, sechszackiger Stern. Dreieckige Marker gibt es in vier Himmelsrichtungen: $\hat{}$, $>$, v , $<$.

Einfache Kontrolle des Grafikfeldes

Anschließend an den Aufruf `plot(..)` ist das Aufrufen der Funktion `axis` möglich, um das Zeichenfeld anzupassen. Die wichtigsten Fälle von axis-Aufrufen sind:

`axis([xmin, xmax, ymin,ymax])` für das Festlegen der Bereiche der Abszissen- und Ordinatenwerte.

`axis square` , um zu erreichen, dass die dargestellte Grafik auf dem Bildschirm (ohne die absolute Größe festzulegen) in einem Quadrat einbeschrieben ist.

Diese beiden Aufrufe werden meist beide nacheinander verwendet.

Weitere wichtige Hilfsmittel zum Erzeugen von Grafiken sind die Befehle `hold on` und `hold off` für das Weiterzeichnen in derselben Grafik und die anschließende Freigabe für ein neues Bild. Mehrere Linienzüge können auch durch Eingabe von mehreren Paaren von x- und y- Vektoren im gleichen Aufruf gezeichnet werden, wie in

`plot(x1,y1,x2,y2)` oder auch `plot(x1,y1,'k',x2,y2,'r')`.

Eine Unterteilung des Zeichenfeldes in mehrere Bereiche bewirkt der Befehl

`subplot(n,m,k)`. Die ersten zwei Parameter steuern die Unterteilung des Zeichenfeldes in der Art einer Matrixdimensionsangabe.

So ergeben z. B. n,m = 1,2 zwei Teilbilder nebeneinander, entsprechend 1 Zeile und 2 Spalten; analog erzeugen n,m = 3,1 drei Teilbilder übereinander. Der dritte Parameter `k` bestimmt die Teilbildnummer, in welche nachfolgende Bildbestandteile gezeichnet werden. Ein Wechsel des zu bearbeitenden Teilbildes wird immer durch `subplot(n,m,k)` eingeleitet mit gleichen `n,m` wie vorher, aber anderen Werten von `k`.

Drucken, Grafikausgabe auf Papier

Die Ausgabe des gerade aktuellen Bildes auf Papier erfolgt mit dem Befehl `print`. Einblick in die vielen möglichen Optionen gibt der Befehl `help print`. Einige wichtige Fälle sind als Beispiele unten angegeben.

Ohne Parameter sendet der bloße Befehl `print` die Grafik an den Standarddrucker.

`print -dpsc` ergibt einen farbigen Ausdruck im PostScript-Format.

`print -dpsc filename`, also z. B. `print -dpsc fitresult03`

speichert die PostScript Information in einem File mit dem Namen 'fitresult03.ps' (bzw. dem angegebenen Namen).

Die Dateinamenserweiterung ' .ps ' wird von MATLAB automatisch hinzugefügt.

Soll die Grafik in einen Bericht eingefügt werden, so ist das Fileformat "Encapsulated PostScript" besonders geeignet:

`print -depsc filename` ergibt ein File 'filename.eps '.

Die anderen Optionen betreffen viele andere mögliche Ausgabeformate wie z. B. tiff, jpeg, HP laserjet und HP-deskjet.

Für MS-Windows Benutzer sind noch die folgenden Optionen von Bedeutung:

`print -dwinc` für den aktuell ausgewählten Windows-Drucker,

`print -dmeta` zum Übertragen eines Windows-Metafiles in die Zwischenablage.

`print -dbitmap` zum Übertragen eines Files im Bitmap-Format in die Zwischenablage.

MATLAB Grundlagen aktivieren

Checkliste zu Kapitel 1

Durch die Arbeit mit diesem Kapitel sollten Sie die Grundkenntnisse für das Einsetzen von MATLAB bei einfachen Problemen erworben haben. Im Detail sollten Sie über die folgenden Kenntnisse und Fähigkeiten verfügen:

- Beliebige Berechnungen mit Zahlen in MATLAB eingeben.
- Die wichtigsten Fachausdrücke der Matrizenmathematik verstehen, wie z. B. Dimension, Zeile, Spalte, Indizierung, Diagonale, Symmetrie, Transponieren, Einheitsmatrix.
- Matrizen in MATLAB eingeben, Matrizenmultiplikation, Matrix–Vektor-Multiplikation in MATLAB formulieren, den Transpositionsoperator einsetzen.
- Die Punkt-Operatoren bzw. die gewöhnlichen arithmetischen Operatoren richtig einsetzen.
- Die Regeln der Matrixmultiplikation kennen und beim Multiplizieren von Matrizen von Hand anwenden.
- Lineare Gleichungssysteme in Matrizenform formulieren und mit MATLAB lösen.
- Den impliziten Schleifenoperator zum Erzeugen von langen Zeilenvektoren verwenden, diese als Argument in mathematische Funktionen einsetzen und einfache Funktionsplots erzeugen.

Übungen zum Kapitel 1

Einfache Berechnungen

10–1 Die Erde in Zahlen

In vernünftiger Näherung beträgt der Erdumfang 40 000 km. Das Meter wurde so definiert, dass ein Kilometer gerade einem 'c', also einem Hundertstel eines Neugrades auf dem Äquator entspricht. Ein 'cc' ist dann also 10 Meter. (Die Seemeile war als eine Bogenminute der alten Grad-Einteilung definiert worden.)
Berechnen Sie die Oberfläche, das Volumen und die durchschnittliche Dichte der Erde (Die Masse der Erde beträgt $5.97 * 10^{24}$ kg.)
Die darin zu verwendenden Formeln sind $V_{\text{Kugel}} = 4/3 \cdot \pi \cdot r^3$, $O_{\text{Kugel}} = 4 \cdot \pi \cdot r^2$, $U = 2 \cdot \pi \cdot r$, $\rho = m/V$.
Verwenden Sie in allen Berechnungen dieses Abschnittes die Befehle `format ...` zur verständlichen Darstellung der Resultate.

10–2 Mond und Sonne

Die mittlere Distanz von der Erde zum Mond beträgt 384 400 km, diejenige zur Sonne 150 Millionen km. Beide Himmelskörper werden von der Erde aus mit dem gleichen totalen Öffnungswinkel von 31 Winkelminuten gesehen. (Bei einer Sonnenfinsternis passen beide genau übereinander.)
Berechnen Sie daraus die ungefähren Durchmesser von Mond und Sonne!

10–3 Der Eiffelturm

Die Gitterkonstruktion des Eiffelturms, erbaut für die Weltausstellung 1889, ist erstaunlich leicht, obwohl die Streben und Knotenelemente aus Stahl sind. Damals standen noch keine Aluminium-, oder Magnesiumlegierungen zur Verfügung und schon gar kein mit Kohlefasern verstärkter Kunststoff. Als Demonstration der leichten Bauweise soll die Masse der Luft in dem, den Turm umfassenden Zylinder berechnet werden und mit der Eisen-Masse des Bauwerks von 7000 t verglichen werden. Die Seitenlänge der Grundfläche beträgt 160 m und die Höhe 320 m, die Dichte von Luft kann als 1.2 kg/m^3 angenommen werden.

10–4 Physikalische Arbeitsleistung eines Bergwanderers

Bei Bergwanderungen wird als Richtwert für die Überwindung einer Höhendifferenz von 300 m eine Stunde eingesetzt. Welche durchschnittliche Leistung in Watt erbringt ein 75 kg schwerer Wanderer für den reinen Höhengewinn? Die Erdbeschleunigung beträgt 9.81 m/s^2.

10–5 Wieviele Sekunden hat ein Jahr?

Durch einfache Multiplikationen erhält man die Zahlen für die Anzahl Sekunden bzw. Minuten pro Tag, Woche, Jahr sowie Stunden pro Woche und Jahr.

10–6 Wie weit kommt ein Tanklaster mit seiner eigenen Ladung?

Von einem Tanklastfahrzeug kann man im Langstreckenbetrieb annehmen, dass es etwa 15 Liter Diesel-Treibstoff pro 100 km verbraucht. Für die volle Ladung kann man einen Wert von ungefähr 20 000 Liter einsetzen. Wie weit kann er also mit der eigenen Ladung fahren? Käme er rund um die Erde, falls es eine Straße gäbe?

10–7 Berechnungs-Spass beim Essen
Versuchen Sie die Gesamtlänge aller Pommes-Frites, bzw. aller Spaghetti auf einem Teller angenähert zu bestimmen, wenn diese in Längsrichtung geradegezogen aneinandergereiht werden. Hinweis: Für eine Portion kann man von etwa 300 Gramm ausgehen, das spezifische Gewicht der meisten Speisen ist nahezu 1 g/cm^3 und die Dicke von Pommes-Frites ist ca. 6 mm während der Durchmesser von gekochten Spaghetti etwa 2 mm beträgt.

Produktschreibweise mit '*'

10–8 Binomische Formeln
Die Binomischen Formeln eignen sich besonders gut zum Einüben der in MATLAB immer verlangten Verwendung des Multiplikationsoperators '`*`' zwischen zwei Faktoren.
Wählen Sie zwei Zahlenwerte für `a` und `b`, z. B. `a = 10, b = 3`. Berechnen Sie nun die Resultate für $(a+b)^n$ und $(a-b)^n$, mit $n = 2, 3, 4$ je auf zwei Arten, nämlich direkt durch Bilden der Summe/Differenz und anschließendes Potenzieren, sowie aufwendiger durch Summieren der einzelnen Terme der binomischen Formeln, wie z. B. $a^3 - 3 * a^2 * b + 3 * a * b^2 - b^3$.

Implizite Schleifen und Summen

10–9 Summen von natürlichen Zahlenreihen
Der Mathematiker Karl Friedrich Gauß sollte als Primarschüler mit der Aufgabe beschäftigt werden, alle Zahlen von 1 bis und mit 100 zusammenzuzählen. Da er sofort das Prinzip herausfand, dass jede Zusammenfassung einer Zahl aus der unteren Hälfte mit einer passenden aus der oberen Hälfte den Wert 101 ergab, und dass es 50 solche Paare gab, fand er sehr schnell das Resultat $5050 = 101 * 50$.

Die allgemeine Formel für die Summe einer Reihe natürlicher Zahlen von a bis b lautet

$$s = 1/2 \cdot (a + b) \cdot (b - a + 1).$$

Testen Sie diese Formel, indem Sie verschiedene Reihen mit dem Befehl `r = a:b` erzeugen und deren Summe mit `sum(r)`, sowie mit der obigen Formel berechnen.

10–10 Summen von allgemeinen arithmetischen Reihen
Eine allgemeine arithmetische Reihe ist definiert durch die Formel für das allgemeine Element: $a_k = a_1 + (k - 1) \cdot d \;\; ; \quad k = 1 \ldots n$.

Die zugehörige Summenformel lautet:

$$s = 1/2 \cdot n \cdot (a_1 + a_n) = 1/2 \cdot n \cdot (2 \cdot a_1 + (n - 1) * d).$$

Verwenden Sie wiederum den impliziten Schleifenoperator `:` (diesmal in der Form `a:d:b`), um verschiedene arithmetische Reihen zu erzeugen und anschließend deren Summe mit sum(r) zu berechnen. Vergleichen Sie jeweils den so bestimmten Summenwert mit dem Resultat der Formelauswertung! (Mögliche Beispielwerte sind $a_1 = 1, 0, 10, -20, 0.1$, $d = 2, 3, -2, 5, 0.1$ und $n = 101, 12, 11, 10, 10$.)
Trotz des Vorfaktors $1/2$ wird das Resultat für ganzzahlige a_1 und d immer ganzzahlig; ein ganz kleines mathematisches Wunder!

10–11 Summenwert von magischen Quadraten
Ein magisches Quadrat der Dimension nxn enthält alle natürlichen Zahlen zwischen 1 und n^2. Der Wert der überall gleichen Zeilen- und Spaltensummen lässt sich aus folgender Überlegung bestimmen: er muss n mal dem Durchschnittswert aller Elemente entsprechen. Berechnen Sie diesen Summenwert für $n = 3, 4, 6, 9$!

Matrixdefinition

10–12 Jedes Element hat seinen Platz!
Geben Sie in MATLAB alle möglichen 2x2 Matrizen ein, welche die vier Zahlen 1 .. 4, aber an verschiedenen Plätzen enthalten. Nennen Sie diese A1, A2 etc. Die 24 Matrizen sind alle verschieden, das können Sie mit `eqtest = A1 == A2` prüfen.
Suchen Sie in diesen Matrizen Paare, welche sich durch die Abbildungen „Transponieren“ (')-Operator, sowie `fliplr()` bzw. `flipud()` ineinander überführen lassen. (Spiegelung an vertikaler bzw. horizontaler Mittellinie.)

10–13 Zeilen und Spalten
Aus den Zahlen 1 bis 12, alle der Reihe nach eingefügt, kann man auf verschiedene Weise Rechtecksmatrizen erzeugen. Diese haben die Dimensionen 1x12, 2x6, 3x4, 4x3, 6x2, 12x1. Geben Sie alle diese Varianten in MATLAB ein, wobei die Definition der Namen E, Z, D, V, S, C ein nachträgliches Abrufen zum Quervergleich der verschiedenen Matrizen erlaubt.
Die Funktion `reshape()` erlaubt, die verschiedenen Matrizen ineinander zu verwandeln. Finden Sie mit der MATLAB help Funktion selbst heraus, wie diese anzuwenden ist.
Testen Sie auch in diesem Fall, was passiert, wenn Sie versuchen zwei solche Matrizen zu addieren, zu subtrahieren oder zu vergleichen!
Wenden Sie die Funktionen `length()` und `[m n] = size()` auf diese Matrizen an und überlegen Sie sich anhand der Resultate die Arbeitsweise dieser Funktionen!

10–14 Matrizen als Bestandteile von Matrizen
Verwenden Sie die Möglichkeit, Matrizen selbst als Elemente in der Definition einer Matrix einzusetzen!
`Q = [ 1 0 ; 0 1 ]` kann in `H= [ Q Q ; Q Q]` verwendet werden. Erweitern Sie das Prinzip um eine 8x8-Matrix mit Schachbrettverteilung von 0 und 1 zu erzeugen!

10–15 Vektoren zu Matrizen zusammenfügen
Erzeugen Sie eine Serie von 4 (bzw. n) Zeilenvektoren mit unterschiedlicher Anzahl von Einsen in der Art `v1 = [ 1 0 0 0]`, `v2 = [ 1 1 0 0]`, `v3 = [ 1 1 1 0]`, `v4 = [ 1 1 1 1]`. Zeigen Sie dass durch Aufeinanderstapeln
`L = [ v1 ; v2 ; v3 ; v4]` der Vektoren eine untere Dreiecksmatrix entsteht.
Wenn Sie die Vektoren einzeln transponieren, so erhalten Sie durch seitliches Aneinanderfügen eine obere Dreiecksmatrix. `R = [ v1' v2' v3' v4']`.

10–16 Matrizen aneinanderfügen
Starten Sie mit einer 6x6 Einheitsmatrix `I6 = eye(6)`.
Erzeugen Sie geeignete Spalten- und Zeilenvektoren mit der Funktion `zeros(n,m)`, die Sie an `I6` anfügen können, um daraus je eine 7x7 Matrix zu erhalten, welche dann die Einsen in der oberen, bzw. unteren Nebendiagonalen aufweist!

10–17 Einmaleins-Tabelle
Erzeugen Sie eine Einmaleins-Tabelle durch untereinander Anordnen von Zeilenvektoren mit impliziter Schleife der Art `k:k:10*k`, mit verschiedenen Werten von k!

10–18 Größter gemeinsamer Teiler, kleinstes gemeinsames Vielfaches
Bestimmen Sie zu den Zahlen 4, 7, 13, 20, 36, 49, 64, 72 eine (symmetrische) Matrix, welche die g.g.T. enthält, sowie eine mit den k.g.V.-Werten. Die beiden Funktionen in MATLAB heißen `gcd(a,b)` (greatest common divider) und `lcm(a,b)` (least common multiple).
Zeigen Sie mit Hilfe der Punkt-Multiplikation dieser Matrizen, dass gilt: `gcd(a,b) .* lcm(a,b) = a .* b`.

Fachausdrücke zur Matrizentheorie

10–19 Welche der folgenden Aussagen sind wahr?

- Eine symmetrische Matrix ist quadratisch.
- Eine antisymmetrische Matrix, addiert zu ihrer Transponierten ergibt die Nullmatrix.
- Eine Matrix mit lauter Nullen ist das Neutralelement der Matrixmultiplikation.
- Jede Diagonalmatrix ist regulär.
- Das Produkt quadratische Matrix mal Spaltenvektor ergibt wieder einen Spaltenvektor, falls die Multiplikation möglich ist.
- Das Produkt einer Rechtecksmatrix mit ihrer Transponierten ist immer möglich und ergibt eine quadratische Matrix.
- Jede symmetrische Matrix ist mit ihrer Transponierten identisch.

10–20 Zerlegung in symmetrischen und antisymmetrischen Anteil
Berechnen Sie aus der Matrix T den Ausdruck $T - T'$! Suchen Sie eine Methode, um den antisymmetrischen A und den symmetrischen Anteil S der Matrix T zu bestimmen, so dass gilt $A + S = T$!

$$T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

10–21 Charakteristische Eigenschaften von speziellen Matrizen
Finden Sie je eine 3x3 Matrix der folgenden Typen: symmetrische Matrix, antisymmetrische Matrix, Nullmatrix, Diagonalmatrix, Einheitsmatrix, obere Dreiecksmatrix, untere Dreiecksmatrix.
Bestimmen Sie von jedem Typ die Anzahl Freiheitsgrade.
Suchen Sie möglichst viele Teilmengenrelationen der Art: „Die Einheitsmatrix bildet eine Teilmenge der Diagonalmatrizen.“

Hilfsfunktionen zum Erzeugen von Matrizen

10–22 zeros(), ones(), eye()
Sehen Sie Sich die Resultate der MATLAB-Funktionen zeros(k), ones(k), eye(k) an für verschiedene quadratische Dimensionszahlen k, z. B. $k = 3, 7, 10$.
Wenden Sie zeros() und ones() zum Erzeugen einer nxm Rechtecksmatrix an.
Erzeugen Sie das Komplement zur Einheitsmatrix mit Nullen auf der Diagonalen und Einsen an den anderen Plätzen.

10–23 diag()
Sehen Sie in der Hilfefunktion nach, wie `diag()` anzuwenden ist. Erzeugen Sie anschließend eine Diagonalmatrix der Dimension nxn z. B. $n = 15$ mit den fortlaufenden ungeraden Zahlen $1, 3, 5$ etc. in der Diagonalen.

10–24 Block-Diagonalmatrizen
Verwenden Sie das Prinzip, dass man Matrizen aus kleineren Matrizen zusammenfügen kann, um quadratische Matrizen der Dimension $k * n \times k * n$ ($k = 1, 2, 3, 4, 5$), ($n = 2, 3, 4$) zu erzeugen, welche auf der Diagonalen und auf k parallelen Linien im Abstand n dazu je Einsen haben und sonst Nullen. (Lauter Matrizen die mit eye(n) erzeugt wurden, werden zusammengefügt.)
Benutzen Sie die Funktion spy(M) zur grafischen Darstellung der von Null verschiedenen Werte einer Matrix. Diese Funktion ist besonders beim Arbeiten mit dünn besetzten Matrizen (englisch: sparse matrices) wertvoll.

10–25 Quadrieren von magischen Quadraten
Lernen Sie die MATLAB-Demo über magische Quadrate (Demo – Matrizen – Matrix-Manipulationen) kennen! Nehmen Sie daraus ein magischer Quadrat 3x3 oder 4x4 und probieren Sie das folgende Rezept aus: Ein magisches Quadrat der Dimension nxn kann man wie folgt quadrieren: In die nxn Teilmatrix am Platz (j,k) setzt man die Originalmatrix ein, plus den „Sockelwert“ $(m_{jk} - 1) * n^2$: dies ergibt ein magisches Quadrat der Dimensionen $n^2 \times n^2$.

Logische Operatoren angewandt auf Matrizen

10–26 Invertiertes Schachbrettmuster
Die in einer früheren Übung erzeugte Matrix mit 0- und 1-Belegung in einem Schachbrettmuster soll durch Anwendung eines der logischen Vergleichsoperatoren ’==’, ’<=’, ’>=’, ’<’, ’>’ mit einer Matrix ones(8) oder zeros(8) verglichen werden. Das Resultat soll die in allen Elementen invertierte Matrix liefern (0 statt 1, 1 statt 0). Logische Operatoren liefern 1 als ’true’ und 0 als ’false’ mit der gleichen Dimension wie die verglichenen Matrizen.

10–27 Negativbild durch logische Operatoren
Mit den Befehlen `E = zeros(7); E(6,3:6)=ones(1,4); E(4,3:5)= ones(1,3); E(2,3:6)= ones(1,4); E(2:6,2) = ones(5,1);` ergibt sich eine Matrix, welche mit der Funktion `spy(E)` einen Buchstaben E zeigt. Durch `N = E == zeros(7)` kann davon ein Negativbild erzeugt werden.

Multiplikation von Matrizen

10–28 Matrixmultiplikation
Berechnen Sie von Hand die Matrizenprodukte

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix} * \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad \text{und} \quad \begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix} * \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

und kontrollieren Sie Ihr Resultat mit Hilfe von MATLAB!

10–29 Indexwertmatrix mal allgemeine Matrix
Multiplizieren Sie mit Bleistift und Papier je eine 2x2 und eine 3x3 Matrix deren Werte die Indizes als zweistellige Zahlen enthalten, jeweils von links und von rechts mit einer allgemeinen Matrix mit den Werten a, b, c, d bzw. a bis i.

10–30 Matrixmultiplikation „von Hand“
Multiplizieren Sie mit Hilfe von Bleistift und Papier eine 2x2 Matrix D mit den Zahlen (1..4) mit sich selbst (d. h. berechnen Sie D*D von Hand). Multiplizieren Sie ebenso die 3x3 Matrix T mit den Zahlen (1..9) mit sich selbst.
Prüfen Sie die Resultate mit MATLAB.

10–31 Formel für die Inverse einer 2x2 Matrix
Multiplizieren Sie mit Bleistift und Papier die zwei allgemeinen 2x2 Matrizen `[a b ; c d]` und `[u v ; w x ]`. Bestimmen Sie aus der Forderung, dass das Resultat die Einheitsmatrix `[1 0; 0 1]` ergeben muss vier Gleichungen, welche die Parameter u, v, w, x als Funktion von a, b, c, d erfüllen müssen.
Durch Auflösen dieser vier Gleichungen nach u, v, w, x erhält man eine geschlossenen formelmäßige Lösung für die Inverse einer 2x2 Matrix.

10–32 Legalitätsüberlegung bei Matrizen- und Vektormultiplikationen
Mit den 4 Matrizen / Vektoren A 2x3, B 3x3, v 3x1, w 2x1 sollen die folgenden Operationen auf ihre Legalität überprüft werden:

A+B A+A B+B v+w w-v
A*B B*A A'*B B*A'
A*v v*A v*A' w*A' v*v' v'*v B*v B*w A'*v
A*A A'*A A*A' B*B

10–33 Legale und illegale Matrixmultiplikationen
Bilden Sie alle möglichen Rechtecksmatrizen, welche die Zahlen 1 bis 6 der Reihe nach enthalten, also E 1x6, Z 2x3, D 3x2 und S 6x1.
Bilden Sie zusätzlich deren Transponierte Et=E', Zt, etc. und suchen Sie alle legalen Multiplikationen, welche zwischen zwei von diesen 8 Matrizen möglich sind! Bestimmen Sie jeweils die Resultat-Dimensionen.

10–34 Legale und Matrizen- und Vektor-Multiplikationen
Erzeugen Sie die vier Matrizen. bzw. Vektoren A(4x3), N(3x3), v(3x1), w(1x4), so dass die darin enthaltenen Zahlen den Index als zweistellige Zahl darstellen (z. B. $a_{12} = 12, n_{33} = 33$). Prüfen Sie mit MATLAB, welche der folgenden Multiplikationen legal sind und überlegen Sie sich jeweils vorgängig, ob Sie den Fall als legal eingestuft hätten: (M' steht für M^T, d. h. M-transponiert)

w*A, w'*A, w*A', w'*A', A*w, A*w', A'*w, A'*w'
w*w, w*w', w'*w, v*v, v'*v, v*v'
N*v, v*N, v'*N, N*v'
A*N, A*N', A'*N, A'*N'

10–35 Nachprüfen einer Matrizengleichung
Zeigen Sie durch Nachrechnen von Hand, dass gilt:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{a*d-b*c} * \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Formulieren von Gleichungssystemen in Matrizenform

10–36 In einzelnen Gleichungen vorgegebenes Gleichungssystem
Die folgenden Gleichungen bilden ein lineares Gleichungssystem. Stellen Sie dieses in Matrizenform dar und lösen Sie es mit MATLAB.

$$r+s+t+u=0, \qquad r-s=1, \qquad u+t-s=-1, \qquad t-r=1.$$

10–37 Gleichungssystem, durch einzelne Gleichungen gegeben
Lösen Sie das untenstehende Gleichungssystem mit MATLAB:

$$x=2*y, \qquad y+z=5, \qquad x-u=2, \qquad x+u=3*y+1.$$

10–38 In einem Schema vorgegebenes Gleichungssystem
Lösen Sie das untenstehende Gleichungssystem mit MATLAB, nachdem Sie es in Matrizenform gebracht haben.

$$\left|\begin{array}{rcrcrcrcl} x_1 & - & x_2 & & & & & = & 5 \\ & & x_2 & - & x_3 & + & x_4 & = & 3 \\ & & & & x_3 & - & x_4 & = & 1 \\ x_1 & & & & & + & x_4 & = & 3 \end{array}\right|$$

10–39 Gleichungssystem in gewohnter Form
Beachten Sie das richtige Einsetzen von Matrixelementen mit den Werten 0, 1 und -1, da diese Zahlen im Gleichungssystem nicht geschrieben werden. Lösen Sie dieses Gleichungssystem mit MATLAB, nachdem Sie es in Matrizenform gebracht haben.

$$\left|\begin{array}{rcrcrcl} 3x & + & y & - & z & = & 7 \\ x & & & + & 2z & = & 3 \\ & & y & - & z & = & 0 \end{array}\right|$$

Einfache Funktionsplots

10–40 Plot von geraden Potenzfunktionen
Erstellen Sie eine Grafik mit den Funktionen $y_1 = x^2,\ y_2 = x^4,\ y_3 = x^8,\ y_4 = x^{16}$ im Bereich $x = 0..2$, $y = 0..2$, alle in derselben Grafik, mit verschiedenen Farben.

10–41 Gerade und ungerade Potenzfunktionen
Stellen Sie die ersten 5 Potenzfunktionen $y_1 = x$, $y_2 = x^2$ etc. in einem gemeinsamen Bild im Bereich $-2 < x < 2$ und $-2 < y < 2$ dar. Wählen Sie für die ungeraden Potenzen die rote Farbe ('r') und für die geraden schwarz ('k').

10–42 Verschiedene Wurzelfunktionen
Erzeugen Sie eine gemeinsame Grafik der Funktionen

$$y_1(x) = x, \qquad y_2(x) = \sqrt{x}, \qquad y_3(x) = \sqrt[3]{x}, \qquad y_4(x) = \sqrt[4]{x}$$

im Bereich $x = (0..4]$

MATLAB-Operatoren und Grundfunktionen

10–43 MATLAB-Operatoren
Stellen Sie alle Ihnen bekannten Operatoren in MATLAB tabellarisch zusammen und geben Sie bei den speziellen Operatoren stichwortartig deren Bedeutung an!

10–44 Vordefinierte Konstanten
Zählen Sie die Ihnen bekannten, in MATLAB vordefinierten Konstanten auf.

10–45 Spiegelungsoperationen an Matrizen
Füllen Sie eine 3x3-Matrix mit den Werten von 1 bis 9. Testen Sie an dieser Matrix die drei „Spiegelungsoperationen“ Transponieren, flipud() und fliplr(). Suchen Sie eine Kombination aus diesen, welche eine „falsche Transposition“ bewirkt, d. h. eine Spiegelung an der von links unten nach rechts oben verlaufenden „falschen“ Diagonale.

Miniprojekte zum MATLAB Einstieg

101 Fadenstern
Zeichnen Sie mit MATLAB einen vierstrahligen „Fadenstern“, indem Sie die Punkte von -10 bis 10 auf der x-Achse mit passenden Punkten auf der y-Achse durch Geraden verbinden.

102 Farbkreis
Benutzen Sie die Funktionen `fill()` oder `patch()`, um in einem Kreis angeordnete farbige Flächen mit den zu ihrer Winkelposition passenden Farben zu füllen. Die RBG-Werte zum Einfügen in die Grafikaufrufe erhalten Sie aus den Werten für Farbwinkel (hue), Sättigung (saturation) und Helligkeit (value) mit der Bibliotheksprozedur `[r,g,b]=hsv2rgb(h,s,v)`.

Selbsttests zum Kapitel 1

Diese Testserien dienen zur Selbstkontrolle der erworbenen Kenntnisse. Falls das erste Kapitel sorgfältig durchgearbeitet wurde, sollte es möglich sein, eine Serie in ca. 60 Minuten vollständig zu lösen.

Testserie 1.1

T111) **Verständnisfragen**

- Für welche arithmetischen Operatoren gibt es in MATLAB zugehörige Punkt-Operatoren und was ist die Bedeutung dieser Punkt-Operatoren?
- Geben Sie die notwendigen Befehle an, um mit einer Bibliotheksprozedur in MATLAB eine 4x4 Einheitsmatrix zu erzeugen.
- Wie erhält man in einer MATLAB-Grafik eine schwarze Linie?
- Wie extrahiert man die ganze 4. Zeile aus einer nxn Matrix (Annahme n ist mindestens 4)?

T112) Erzeugen Sie eine grafische Darstellung der Sinusfunktion über drei ganze Perioden mit gleich großen Einheiten in der x- und y-Richtung!

T113) Geben Sie die nötigen MATLAB-Befehle an, um eine 8x8 Matrix im Schachbrettmuster mit 0 und 1 zu füllen! Starten Sie mit der Eingabe einer 2x2-Matrix, fügen Sie vier davon zu einer 4x4 Matrix zusammen und dann 4 von diesen zur gesuchten 8x8 Matrix.

T114) Erzeugen Sie einen Vektor mit 505 Elementen, welcher 5 Perioden einer Sägezahnfunktion enthält, wobei jedesmal die Werte von -10 bis 10 laufen.

T115) Schreiben Sie die MATLAB-Befehle auf zum Lösen des Gleichungssystems:

$$x + z = 4; \qquad y + z = 2; \qquad 2 * x - 4 * y = 2.$$

Testserie 1.2

T121) **Verständnisfragen**

- Wie erhält man in MATLAB die Lösung x eines in Matrixform gegebenen linearen Gleichungssystems A*x=b?
- Wieviele frei wählbare Größen enthält eine nxn Diagonalmatrix?
- Wie erreicht man, dass nachfolgende plot-Aufrufe in dasselbe Bild gezeichnet werden wie der vorangehende?
- Wie lautet der einzugebende Text, um in der Variablen E3 eine 3x3 Einheitsmatrix direkt einzugeben (ohne Verwendung einer Bibliotheksprozedur)?

T122) Erzeugen Sie eine simultane Grafik der Funktionen $y_q = \sqrt{x}$, $y_c = \sqrt[3]{x}$ und $y_f = \sqrt[4]{x}$, mit dem Intervall $\varepsilon < x < 2$.

T123) Das File 'tempmess.dat' enthalte in ASCII-Form eine Tabelle mit der Tages-Nummer als erster Zahl in jeder Zeile, gefolgt von drei Temperaturmessungen.
Geben Sie die MATLAB-Befehle an, um dieses File einzulesen und anschließend die Temperaturwerte gegen die Messzeit in einem gemeinsamen Zeichenfeld zu plotten.

T124) Programmieren Sie in MATLAB eine Schleife, welche mit einer if-Konstruktion eine mit 201 Werten tabellierte Sinusfunktion bei Werten über 0.6 und unter -0.6 auf die Grenzwerte zurücksetzt (Clipping)!

T125) Berechnen Sie von Hand das nachfolgende Matrizenprodukt!

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 8 \\ 0 & 5 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 3 & 0 \end{pmatrix}$$