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 und Mac's durch einen Doppelklick auf das Icon von MATLAB oder durch Anwählen des 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 „`&`".) Bei neueren 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 den Markennamen MATLAB immer in Großbuchstaben gesetzt sehen.

Darauf erscheint kurz das quadratische MATLAB-Markenzeichen, Banner genannt. Dieses Bild ist die Visualisierung der Schwingung einer L-förmigen Membran. Kurz danach zeigt sich das in drei vertikale Streifen unterteilte Hauptfenster von MATLAB.

Die meisten Aktionen spielen sich im mittleren Streifen im **Kommandofenster** ab (Fenstertitel: Command Window): Eingabe von Berechnungs-Befehlen, Abfrage von Daten und Hilfstexten, sowie Aufruf von Funktionen und Programmen. Ebenfalls im Kommandofenster erfolgt die Ausgabe von Resultaten, kurzen Texten und Fehlermeldungen.

Die Hilfsfenster in den beiden Seitenstreifen vermitteln nützliche Information zum aktuellen Dateipfad und zu bisher ausgeführten Befehlen, sowie zu den dabei erzeugten Daten.

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

MATLAB und Mathematik kompetent einsetzen, 2. Auflage. Stefan Adam.

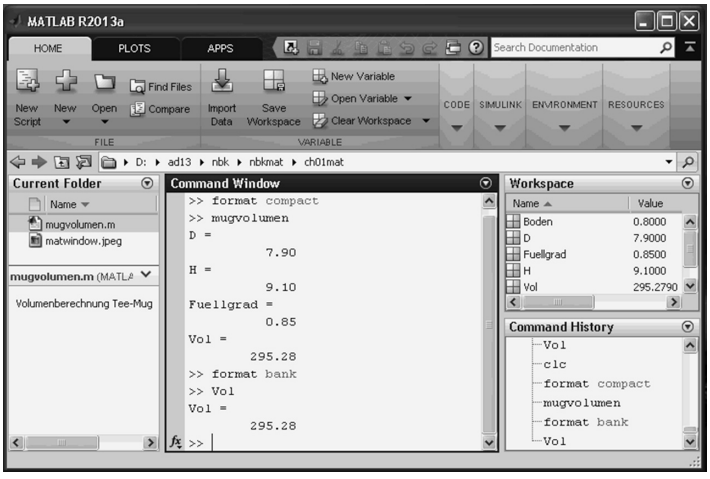

Abb. 1.1 Das Hauptfenster von MATLAB.

Im Kommandofenster, anschließend an die zwei nach rechts zeigenden Winkel >> wartet MATLAB auf die Eingabe des nächsten Befehls. Eventuell muss der Text-Cursor noch mit der Maus dorthin gesetzt werden. Bei einer längeren Berechnung kann sich die Eingabe-Bereitschaft verzögern.

Grafische Darstellungen werden in unabhängigen Fenstern mit der Bezeichnung `figure` (und einer nachfolgenden Zahl) gezeigt.

MATLAB-Befehle bestehen aus direkten Anweisungen zum Ausführen einer Berechnung, Zuweisungen von Daten, Aufrufen von Funktionen oder vielseitigen Kombinationen davon.

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
```

Jede neue Berechnung überschreibt `ans` mit dem neuen Wert.

```
ans =
       17
```

```
>> 7^2                    (die MATLAB-Schreibweise für 7^2, also 7 hoch 2 = 7 · 7)
   ans =
          49              und als Gegensatz
>> 2^7
   ans =
          128             und auch noch
>> 2^-7
   ans =
          0.0078          (hier genügen 4 Dezimalstellen nicht)
>> format long            (verlangt Ausgabe mit 15 Dezimalen)
   ans =
          0.007812500000000
>> format short           (Format zurücksetzen)
```

Wie bei einem Taschenrechner können große oder sehr kleine Zahlen mit einem Exponenten-Zusatz („E“ oder „e“, gefolgt von positiver oder negativer ganzer Zahl) eingegeben werden: (Beispiel Flugzeit des Lichtes von der Sonne zur Erde: 150 Millionen km dividiert durch 300 000 km/s)

```
>> 150E6 / 300e3
   ans =
          500.0000        Resultat in Sekunden
>> tlichtmin = ans/60     geläufiger in Minuten
   tlichtmin =
          8.3333
```

Für die Berechnung der mittleren Dichte der Erde werden mit Vorteil zuerst die Variablen Radius und Masse definiert

```
>> Erad = 6371*1000 ;     inkl. Umwandlung km in m
>> Emass = 5.9736e24;     in kg
>> Edicht = Emass/(4*pi/3*Erad^3)
   Edicht =
          5.5147e+03      in kg/m^3, in bekannten Einheiten 5,5 kg/L
```

Nun noch eine näher liegende Berechnung:

```
>> Vol = (7.9-2*0.3) ^ 2 * pi /4 * (9.1-0.8) ...
          * 0.85
   Vol =
          295.2790
```

In der obigen Volumenberechnung ist es fast unumgänglich, dass man auch den darin vorkommenden Zahlen, und nicht nur dem Resultat, einen Namen zuweist. Das hilft, die durchgeführte Berechnung zu verstehen und zu dokumentieren. Dadurch wird aber die Eingabe mehrzeilig. Am besten schreibt man die Befehle in ein **Skript**.

Skripts sind Dateien mit dem Namen-Zusatz „`.m`“ (sogenannte M-Files), in denen eine Serie von MATLAB-Befehlen als reiner Text aufgelistet ist. Am besten erstellt man M-Files mit dem MATLAB-eigenen Editor. Dessen Fenster erscheint

im Mittelstreifen oberhalb des Command Window, wird aber mit Vorteil abgekoppelt in ein separates Fenster.

Das M-File `mugvolumen.m` hat in unserem Beispiel die folgende Form:

```
%  Volumenberechnung Tee-Mug
D = 7.9 ;
H = 9.1 ;
Wand = 0.3; Boden = 0.8;
Fuellgrad = 0.85;
Vol = (D-2*Wand)^2 *pi/4 * (H-Boden) * Fuellgrad
```

Sobald dieses File unter dem Namen `mugvolumen.m` abgespeichert ist, bewirkt der Befehl `>> mugvolumen` (Filename, aber ohne Zusatz!) die Ausführung der darin aufgezeichneten Befehle. Das Resultat ist dasselbe wie beim einzeiligen Befehl

```
Vol =
       295.2790
```

Der Mug fasst 295 cm^3, also knapp drei Deziliter.

1.1.3 Befehlsstruktur: ein erster Überblick

Aus diesen einfachen Beispielen ersieht man bereits einige Grundprinzipien von MATLAB, welche in der Anwendung immer wieder auftreten:

> Die Zahl $\pi = 3{,}141\,592\ldots$ für die Kreisberechnung ist vordefiniert und unter dem Namen `pi` abrufbar.

Bei **Skript-M-Files** wird eine Reihe von Befehlen in ein Textfile mit dem Filenamen-Zusatz `.m` geschrieben. Der ganze Block gelangt dann durch Eintippen des Filenamens (ohne den Zusatz `.m`) zur Ausführung.

Das Arbeiten mit ganzen Blöcken (Skripts) von Befehlen heißt in der Informatik Makro-Programmierung. (Ausblick: Im Innern eines Blocks können in beliebiger Tiefe auch andere Blöcke aufgerufen werden; Rekursion ist aber nur für Funktionen erlaubt.)

Die Steuerung der **Bildschirm-Ausgabe** erfolgt mit `format ..`
(Beispiel: Feinstruktur-Konstante und Inverse)

Befehl	**Stellen**	α	$1/\alpha$
`format short`	4	0.0073	137.0360
`format long`	15	0.007297352569800	1.370359990743064e+02
`format long e`	15	7.297352569800e-03	1.370359990743064e+02
`format bank`	2	0.01	137.04

Für Zeilenabstände eng/locker mit: `format compact`/`format loose`.

Merkpunkte: Konventionen der Kommandosprache

- **Zeilen:** MATLAB-Befehle stehen meinst einzeln, jeder in einer Zeile. Lange Befehle können durch drei Punkte `...` am Ende der Zeile eine Fortsetzung verlangen (auch mehrfach).
- **Ausgabesteuerung durch Befehls-Abschluss:** Nach der Ausführung eines Befehls wird das dazugehörige Zwischenresultat im Kommandofenster angezeigt.
  Durch einen Strichpunkt am Zeilenende wird diese Ausgabe unterdrückt.
- **Mehrere Befehle in einer Zeile:** Mehrere Befehle können durch Komma oder Strichpunkt getrennt in dieselbe Zeile geschrieben werden. Komma als Trennung bewirkt Zwischenresultat-Ausgabe, Strichpunkt wie bei einzelnen Befehlen dessen Unterdrückung.

Merkpunkte: Verwendung von Namen/Variablennamen

- Die zu verarbeitenden Zahlen, sowie Zwischen- und Schlussresultate in MATLAB sollten **Namen** (= Variablennamen, Bezeichner) erhalten. Diese können später dazu dienen die mathematischen Größen wieder zu verwenden und helfen erst noch die Übersicht zu bewahren.
- Für jede Berechnung kann der Programmbenutzer den Resultatnamen durch die Formulierung `'name' =` vorgeben, wie z. B. in `c2 = a^2 + b^2` (gefolgt von `c = sqrt(c2)` für die Berechnung der Hypotenuse nach Pythagoras).
- **Namen** sind in MATLAB empfindlich auf **Groß- und Kleinschreibung** (englisch: „case sensitive“). Somit ist also `A` etwas anderes als `a`.
  Namen 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.
- 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 (im Kommandofenster) aus.
- Falls der Benutzer bei einer Berechnung 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 vielfältige Variablennamen!

1.1.4 Berechnung oder Formel-Manipulation?

MATLAB kann beides! Für Formel-Manipulation braucht man allerdings den Zusatz „symbolic toolbox" (in Studentenlizenz enthalten).

In der **Berechnungs-Umgebung**, im normalen Arbeits-Modus von MATLAB werden Zahlen verarbeitet, um damit ein **in Zahlen ausdrückbares Resultat** zu erhalten. Den verwendeten Variablen müssen also vor ihrer Verwendung Zahlenwerte zugeordnet werden. Sonst kommt die Fehlermeldung `Undefined function or variable`.

Die verwendeten Namen werden bei ihrer ersten Zuweisung automatisch deklariert. (Im Gegensatz zu vielen anderen Programmiersprachen.)

Im **Symbolic-Modus** (Formel-Modus) müssen alle in einer Formel verwendeten symbolischen Variablen als solche deklariert werden. Das erfolgt z. B. mit dem Befehl `syms s t u` (Variablen durch Leerschläge getrennt).

Das **Ziel des symbolischen Rechnens ist eine Formel.** Darum wird diese Art der Verarbeitung auch Formel-Modus genannt. Eingegebene Formeln werden umgestellt oder vereinfacht, zu gegebenen Ausdrücken werden Ableitungen oder unbestimmte Integrale ermittelt; von Gleichungssystemen oder Differentialgleichungen werden analytische Lösungen bestimmt.

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. Daraus ergibt sich eine glänzende Gelegenheit, die binomischen Formeln aufzufrischen.

Weil dies für beide Arbeits-Modi gilt, ergibt sich ein direkter Quervergleich zwischen Berechnung und Formel-Manipulation.

Binomische Formeln im Symbolic-Modus

Der Symbolic-Modus zeigt diese Formeln mit Hilfe der Funktion `expand()`:

```
% Deklaration von s und t als symbolische Variablen
syms s t
expand( (s-t)^2)
ans = s^2 - 2*s*t + t^2
expand( (s-t)^3)
ans = s^3 - 3*s^2*t + 3*s*t^2 - t^3
expand( (s-t)^4)
ans = s^4 - 4*s^3*t + 6*s^2*t^2 - 4*s*t^3 + t^4
```

und wieder zurück

```
factor(ans)
ans = (s-t)^4
```

und nur so zum Spaß:

```
expand((s-t)^50)
```

Versuchen Sie nun selbst, die ausmultiplizierte Form von `(s-t)^2` und `(s-t)^3` einzugeben. Achten Sie auf die Eingabe aller Multiplikationszeichen.

Testen Sie Ihre Eingabe mit `factor(ans)`.

Binomische Formeln im Berechnungs-Modus

Im Berechnungs-Modus müssen zuerst den Variablen `a` und `b` Zahlenwerte zugewiesen werden.

Der Test, ob Sie die Formel richtig eingegeben haben ergibt sich aus dem Zahlenwert (es muss gelten $p3 = q3$, $p4 = q4$ etc.)

Fehlende Multiplikations-Operatoren ergeben die Fehlermeldung `Unexpected MATLAB expression` oder `Undefined function or variable 'ab'`. Ergänzen Sie die fehlenden Terme bei den Fragezeichen!

```
a = 10 , b = 2
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  ..?
```

Merkpunkt: Der Multiplikations-Operator

- Die **Multiplikation** von zwei nebeneinander stehenden 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. Diese Konvention gilt übrigens bei fast allen Programmiersprachen.

Klammern in algebraischen Ausdrücken

Die Prioritätsregeln für arithmetische Operatoren werden populär mit dem Satz „Punkt-Rechnung kommt vor Strich-Rechnung“ memorisiert (mit Multiplikations-Punkt und Divisions-Doppelpunkt).

Nochmals genau definiert gilt:

- Addition „+“ und Subtraktion „−“ haben tiefste Priorität;
- Multiplikation „∗“ und Division „/“ die nächst höhere und
- Potenzieren „^“ die höchste.

konkret: bei $4 + 5 * 3^2$ muss zuerst $3^2 = 9$ berechnet werden, dann $5 * 9 = 45$ und am Schluss $4 + 45 = 49$.

Um eine Abweichung davon zu erzwingen braucht man Klammern.

In MATLAB sind für die Strukturierung von arithmetischen Ausdrücken **ausschließlich runde Klammern** erlaubt.

Das Horner-Schema im Formel-Modus Das Horner-Schema zur Auswertung von Polynomen liefert einen weiteren, eindrücklichen Quervergleich zwischen Formel-Manipulation und Berechnung. Zuerst die Darstellung im Formel-Modus:

Das Polynom 4. Grades $P4 = q_4 * x^4 + q_3 * x^3 + q_2 * x^2 + q_1 * x + q_0$ mit den Koeffizienten q_4, q_3, q_2, q_1, q_0 kann im Formel-Modus als normales Polynom definiert werden, wie oben dargestellt. In der Schreibweise des Horner-Schemas werden alle Faktoren „x“ einzeln ausgeklammert. Der Formel-Modus enthält eine Funktion zur Erzeugung des Horner-Schemas aus der normalen Polynomdarstellung. Diese wird anschließend aufgerufen.

```
>>syms  q4 q3 q2 q1 q0 x
>>P4 = q4*x^4 +  q3*x^3 + q2*x^2 + q1*x +  q0
>>P4H = (((q4*x + q3)*x + q2)*x + q1)*x + q0
>>P4Hf = horner(P4)
P4Hf =
q0 + x*(q1 + x*(q2 + x*(q3 + q4*x)))
```

Das Horner-Schema im Berechnungs-Modus Um ein Polynom numerisch, im Berechnungs-Modus zu bearbeiten, müssen wir den Koeffizienten konkrete Zahlen zuweisen. Wir nehmen das Beispiel: $P4N = x^4 + x^3 - 6 * x^2 - 4 * x + 8$ und dessen Auswertung am Wert $x = 1{,}5$. (Nicht vergessen, den symbolic-Status von x zu beenden!)

Die Koeffizienten fassen wir in einer Zahlenfolge mit dem Namen `cf` zusammen. Dies geschieht durch Einschließen in eckige Klammern.

```
>>clear x
>>cf = [1  1  -6 -4  8];
>>x = 1.5;
>>pval=(((cf(1)*x +cf(2))*x +cf(3))*x +cf(4))*x +cf(5)
pval =
      -3.0625
```

Berechnung mit einem Funktions-M-File Von da ist es nun ein kleiner Schritt zum Erstellen einer ersten selbst programmierten Funktion. Dazu schreibt man die folgenden Zeilen in ein Textfile mit dem Namen `hornereval.m` (schon die zweite Art von M-File).

```
function pv = hornereval(cv, xv)
% function pv = hornereval(cv, xv)  Polynom-Evaluation
%   cv = Koeffizienten-Folge, absteigend,
%   xv = auszuwertende x-Positionen (Vektoren erlaubt)
nkoef = length(cv);
pv = cv(1);     % Start der Polynomwert(e)
for cnum = 2:nkoef  % in jedem Schritt: (..)*xv + cv(cnum)
   pv = pv.*xv + cv(cnum); % '.*' erlaubt xv-Vektoren
end
```

Sobald dieses File abgespeichert ist, funktioniert der Befehl

```
>>pval = hornereval(cf,1.5)
pval =
      -3.0625
```

Aber beliebige andere Koeffizienten-Folgen und ganze Zahlenfolgen (Vektoren) von x-Werten sind ebenso möglich. Das Prinzip einer Funktion mit Parameterübergabe und Resultatrückgabe erlaubt die vielfache Anwendung eines einmal programmierten Berechnungsablaufes mit immer neuen Datensätzen.

Merkpunkte zum Funktions-Beispiel

- Die Funktionsdeklaration in der ersten Zeile enthält das Codewort `function` gefolgt von den Rückgabeparametern, dem Gleichheitszeichen, dem Funktionsnamen und den Eingabeparametern in runden Klammern.
  Der Filename (ohne `.m`) muss mit dem Funktionsnamen übereinstimmen.
  Mehrere Rückgabeparameter werden in eckige Klammern eingeschlossen. Den Rückgabeparametern müssen im Innern der Funktion Werte zugewiesen werden.
- Der Punkt vor dem Multiplikations-Stern verhindert, dass MATLAB die Multiplikation als Matrizenprodukt einstuft. Punkt-Stern verlangt dagegen elementweise Multiplikation.
- Die Kommentare (nach dem %-Zeichen) bei der Funktionsdeklaration werden beim Befehl `help hornereval` angezeigt, sind also eine Mini-Dokumentation.
- Der Befehl im Innern der for-Schleife wird für die Werte der Schleifenvariablen `cnum` = 2,3,4, etc. bis `nkoef` je einmal ausgeführt.

Die neue Funktion ist bereit zur Anwendung, verbunden mit einer Grafik:

```
xpt = -3:3; xfin = -3.5:0.1:3.5;   % x grob / fein
ppt = hornereval(cf,xpt)    % Funktionswerte zu xpt
pfin = hornereval(cf,xfin); % Funktionswerte zu xfin
%  grafische Darstellung
plot(xpt,ppt,'+') ; hold on ;
plot(xfin,pfin) ; grid on; hold off;
```

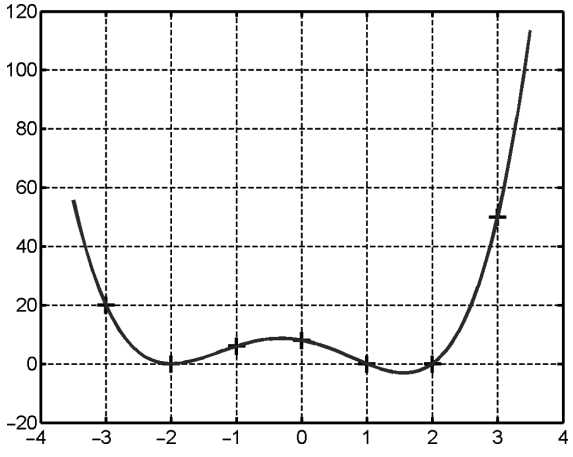

Abb. 1.2 Grafik des Beispiel-Polynoms.

M-Files

Die reiche Vielfalt der Anwendungsmöglichkeiten von MATLAB beruht darauf, dass komplexe Berechnungsaufgaben auf einfachste Weise in Teilaufgaben zerlegt werden können. Die in MATLAB vorhandenen Zerlegungsmechanismen reichen von einfachen Skript-M-Files über Funktions-M-Files bis zu Klassendefinitionen der Objektorientierten Programmierung.

Achtung! Die Bezeichnung M-File verwendet traditionell ein großes „M“, obwohl der Namen-Zusatz „`.m`“ klein geschrieben wird.

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. Für komplexere Anwendungen werden

M-Files mit Klassendefinitionen eingesetzt. MATLAB ist also zu einem großen Teil in MATLAB geschrieben. Durch Erstellen von eigenen Funktionen und Klassen 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 4×6 Matrix mit lauter Nullen.)
`[m,n] = size(M)`	(Eine Liste von mehreren Rückgabeparametern wird mit eckigen Klammern umschlossen.)

1.1.5 Tabellen, Vektoren und Matrizen

Die wenigsten Berechnungen verwenden nur einzelne Zahlenwerte. In den meisten Fällen werden in einer Berechnung ganze Tabellen von Zahlen ausgewertet oder bei einer Simulation erzeugt. Die Zahlenwerte in jeder Spalte einer Tabelle bilden je einen (Spalten)-Vektor. Zusammen bilden sie die Matrix der Tabellenwerte. (Die Text enthaltenden Titelzeilen müssen in einer separaten Variablen gespeichert werden.) Sie passen nicht in die Definition einer **Matrix**:
Eine rechteckige Anordnung von lauter Zahlen.

Die Zahlenwerte einer Tabelle können in einer Matrix unter einem einzigen Variablen-Namen gespeichert werden. Zusätzlich zu allen darin enthaltenen Zahlen gehören zu einer Matrix die Angaben von Zeilen- und Spaltenzahl.

Die Dimensions-Angabe „$\boldsymbol{M}$ ist eine 4×3-Matrix“ oder $\boldsymbol{M}(4 \times 3)$ heißt: $\boldsymbol{M}$ hat 4 Zeilen und 3 Spalten.

Konvention: immer Zeilenangaben zuerst

Eingabe von Vektoren und Matrizen

Eckige Klammern Die eckigen Klammern werden in MATLAB für die Definition von Matrizen und Vektoren gebraucht. Alle Zahlen in einer Matrix oder in einem Vektor (die Elemente), 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.

Beispiele zur MATLAB-Eingabe eines dreidimensionalen Spaltenvektors `v = [2; 0.5; 4]` entsprechend der mathematischen Form

$$\boldsymbol{v} = \begin{pmatrix} 2 \\ 0{,}5 \\ 4 \end{pmatrix}$$

und einer 2×2-Matrix `M = [1 4; 3.5 2]` entsprechend

$$\boldsymbol{M} = \begin{pmatrix} 1 & 4 \\ 3{,}5 & 2 \end{pmatrix} .$$

Weitere Anwendungen der eckigen Klammern finden sich

- beim Funktionsaufruf: Mehrere Rückgabeparameter werden in eckige Klammern eingeschlossen.
  Beispiel: Dimensionszahlen einer Matrix `[nzei,nspa] = size(M)`
- beim Zusammenfügen von Teilen einer Zeichenkette: Beispiel

```
outtext = ['Matrix hat ',num2str(nzei), ' Zeilen und ', ...
     num2str(nspa), 'Spalten']
```

Matrizen als Bündelung von Vektoren Die eckigen Klammern dienen allgemein zum Aneinanderfügen von Zahlen, aber auch von Vektoren und Matrizen. Das wird mit der Bündelung (dem Nebeneinanderstellen) der drei Vektoren $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ zu einer Matrix gezeigt.

```
u = [1 ; 2 ; 3];
v = [4 ; 5 ; 6];
w = [7 ; 8 ; 9];
M = [u v w]
```

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

Selbstverständlich müssen dabei die Teilstücke an den „Klebestellen“ zusammenpassen. Die häufige Fehlerquelle, dass die Teile nicht zusammenpassen, führt zu den Meldungen `Error using vertcat` bzw. `horzcat` (gemeint ist: vertical/horizontal concatenation, d. h. vertikales/horizontales Aneinanderfügen).

Funktionen zum Erzeugen von Matrizen

Die wichtigsten Funktionen zum Erzeugen einer Matrix sind:

- Matrix mit lauter Nullen `Nlnm = zeros(nzei, nspa)`
- Matrix mit lauter Einsen `Einm = ones(nzei, nspa)`
- bzw. `Nlq = zeros(ndim)`, `Eiq = ones(ndim)` (falls quadratisch)
- Einheitsmatrix (quadratisch, Einsen auf Diagonale) `I = eye(ndim)`
- Diagonalmatrix mit Werten des Vektors `dvec` auf Diagonalen `D = diag(dvec)`

Matrix aus Teilmatrizen (Transformation von Ablage und Divergenz bei Teilchenstrahlen.) Die 2×2-Matrizen $\boldsymbol{Mx}$ und $\boldsymbol{My}$ werden definiert und anschießend zu einer 4×4-Matrix ohne x-y-Kopplung zusammengefügt.

```
>>Mx = [1 0.5 ; 0 1];
>>My = [1 0.5 ; 0 1];
>>M  = [ Mx zeros(2) ; zeros(2) My ]

M =
    1.0000    0.5000         0         0
         0    1.0000         0         0
         0         0    1.0000    0.5000
         0         0         0    1.0000
```

Kopieren von Teilmatrizen

Die Funktion `repmat(Tmat,zeimult,spmult)` erzeugt aus der ($m \times n$) Matrix Tmat eine der Dimension (zeimult $*$ $m \times$ spmult $*$ n), indem Tmat entsprechend oft vertikal und horizontal kopiert wird.

```
Mul = repmat([1 2 3], 2,3) % entspricht [ 1 2 3  1 2 3  1 2 3;
                                          1 2 3  1 2 3  1 2 3]
```

Erstellen von Zahlenfolgen

MATLAB kennt zum Erstellen von Zahlenfolgen den **Doppelpunkt-Operator**. Dieser erzeugt eine Aufzählung, also eine implizite Schleife (von .. bis, bzw. von .. Schritt .. bis). `ez = 1:10` erzeugt z. B. die Folge der Zahlen von 1 bis und mit 10, er ist also gleichwertig mit: `ev = [1 2 3 4 5 6 7 8 9 10]`.

Mit nur einem Doppelpunkt wird Schrittweite 1 angenommen.

Mit zwei Doppelpunkten kann zwischen diesen der Wert der Schrittweite gewählt werden. `xs = -1 : 0.2 : 1` erzeugt die Folge der Zahlen von -1 bis 1 mit Schrittweite 0,2 und entspricht damit:

```
xv = [-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1]
```

Es sind auch negative Schrittwerte möglich `xdw = 5:-1:0` erzeugt die Folge der Zahlen von 5 bis 0 mit Schritt -1 und entspricht: `xdwv = [5 4 3 2 1 0]`.

Die mit dem Doppelpunkt-Operator erstellten Zahlenfolgen bilden Zeilen, sind also Zeilenvektoren. Bei Berechnungen mit Vektoren, Tabellen und Matrizen wird aber meist mit Spaltenvektoren gearbeitet. Aus einem Zeilenvektor erhält man durch Transponieren (mit angehängtem Apostroph) einen Spaltenvektor.

```
xspav = xzeiv'
```

Merkpunkte: Definieren von Matrizen

- **Eckige Klammern** dienen zum Definieren von Matrizen und Vektoren durch Einschließen von Elementen, Teilvektoren oder Teilmatrizen.
- Der **Doppelpunkt-Operator** (`start:end` oder `start:step:end`) definiert eine Folge von Zahlen. Diese Zeilenvektoren dienen als Bausteine für Matrizen, als Ordinatenwerte für Grafiken oder als Laufvariablen bei for-Schleifen.
- **Matrizen definieren durch Einlesen** aus Dateien oder durch den **Aufruf von Funktionen** bildet eine wichtige Definitionsart.

1.1.6 Hintergrundinformation und Hilfefunktionen

Workspace – aktive Variablen Eine Übersicht über die momentan definierten Variablen liefert das obere Teilfenster im rechten Seitenstreifen mit dem Titel „Workspace“. Neben den Namen wird der Typ und die Dimension der jeweiligen Variablen angezeigt.

Durch Doppelklicken kann der Inhalt angezeigt werden. Im dabei aufgehenden Zusatzfenster können auch direkt neue Werte gesetzt werden. Mit links-Klick auf eine Variable erscheint ein vielfältiges Menu mit möglichen Aktionen basierend auf dieser Variablen.

Die analoge Information wie im Zusatzfenster Workspace erhält man mit dem Kommando `whos` (für who-size). Eine Kurzversion welche nur die belegten Variablennamen anzeigt wird beim Befehl `who` angezeigt. (Der Funktionsname „who" ist eine Anlehnung an das Lexikon mit persönlichen Daten von gesellschaftlich bedeutsamen Personen „Who is Who".)

Um den Inhalt einer Variablen anzuzeigen muss man nur deren Namen eintippen. **Vorsicht!** bei großen Matrizen kann das einen Schwall von Bildschirm-Ausgaben auslösen.

Der Befehl `clear 'Name'` dient zum Löschen einer Variablen mit dem eingegebenen Namen, `clear` allein löscht den gesamten Workspace.

Weitere Befehle zum Löschen sind `clf` zum Löschen einer Grafik („clear figure"), und `clc` für clear Command Window zum Löschen des Dialogfensters.

Frühere Befehle wiederholen Um einen früher eingetippten Befehl zu wiederholen benützt man am besten ein- oder mehrmals die Pfeiltaste nach oben. Die Pfeiltaste nach unten führt wieder zurück.

Mit den Rechts-/Links-Pfeiltasten kann ein früherer Befehl dann auch modifiziert werden. Diese Line-Edit Funktion arbeitet im Einfüge-Modus: Hineinschreiben mit der Tastatur, Löschen mit der Delete- oder Rück-Taste. Return aktiviert den aktuellen Befehl unabhängig von der Position des Cursors.

Das untere Teilfenster des rechten Seitenstreifens heißt `Command History`. Dieses zeigt die früher eingegebenen Befehle im Überblick. Durch Auswählen einer Befehls-Gruppe in diesem Fenster (Klick beim Start, dann Shift-Klick beim Ende) kann diese in die Zwischenablage kopiert werden. Von dort ist ein Hinein-Kopieren in ein M-File oder direkt in das Command Window mit der Funktion `paste` möglich.

Durch die bequeme History-Information hat das Befehlspaar `diary 'filename'`, zum Starten eines Protokollfiles mit den eingegebenen Befehlen und `diary off`, zu dessen Abschluss, stark an Bedeutung verloren.

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.

Hilfefunktionen Zur großen Erleichterung für alle Anfänger, aber ebenso für die Fortgeschrittenen gibt es in MATLAB gut ausgebaute Hilfefunktionen und Demonstrationsbeispiele. Diese reichen von knappen Hinweisen bis zu ausführlichen Beschreibungen und kurzen Beispielprogrammen.

- Das Kommando `help stichwort` bzw. `kategorie/stichwort` ist die schnellste Nachfragemöglichkeit. Es zeigt die am Anfang des M-Files zum eingegebe-

nen Stichwort eingetragenen Kommentare. Das ermöglicht das Nachschauen des genauen Aufrufformates (der Signatur), 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.
Jeder Anwender kann sich diese Informationsquelle erschließen, indem er kurze Kommentare am Anfang seiner M-Files einfügt.

- 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 der zugehörige Programmcode kann angezeigt werden.

Merkpunkte: Workspace und Hilfefunktionen

- Eine **Übersicht** über die verwendeten Variablen liefern `who, whos` und das Workspace-Fenster rechts oben.
- **Frühere Befehle zurückholen** kann man mit „Pfeil aufwärts". Das aktiviert die Line-Edit Funktion mit „Pfeil links-rechts" im Einfüge-Modus, um den alten Befehl eventuell zu modifizieren.
- Die aufsteigenden Stufen der **Hilfefunktionen** sind `help, lookfor, doc, demo`, sowie die URL www.mathworks.com/matlabcentral/

1.1.7 Datenaustausch mit Files

Für die Anwendung auf größere technische Probleme ist die interaktive Eingabe der Daten im Kommandofenster von MATLAB ungeeignet. Die interaktive Eingabe ist zu aufwändig und 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.

Auch für die effiziente Wiederaufnahme der Arbeit nach einer Unterbrechung eines Arbeitsgangs stehen Funktionen zur Verfügung.

Einlesen und Abspeichern mit load und save

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  oder auch Dmat = load('filename.ext')
```

Mit diesen Befehlen werden die in einem File enthaltenen Zahlen einer Matrix zugeordnet. Der Name der Matrix ist in der ersten Version gerade der Filename (ohne Namenserweiterung), in der zweiten die angegebene Variable.

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 das textartige Ausgabefile.

Ohne den Zusatz `-ascii` dient der Befehl `save filename` der Abspeicherung von Daten im MATLAB-eigenen Binärformat. Die Files erhalten dann die Erweiterung `.mat` und können nur von MATLAB mit `load` wieder gelesen werden. Um dies zu erkennen sollte hier dem Filenamen keine Erweiterung beigefügt werden.

Der Befehl `save filename` ohne Angabe von Variablen ist für eine **Unterbrechung der Berechnung** gedacht und speichert den gesamten Inhalt des Workspace, also alle aktuell belegten Variablen (Zurückholen mit `load filename`).

Die Abspeicherung im Binärformat kann mit `save filename var1,var2` etc. auf eine Gruppe ausgewählter Variablen beschränkt werden.

Wie bei `load` ist auch bei `save` eine Variante in der Art eines Funktionsaufrufs möglich, also `save('filename')`. Dabei muss der Filename in einfache Apostroph eingeschlossen werden.

Die „Import Data“ Funktionen

Beim Anklicken der Schaltfläche „Import Data“ erscheint ein File-Auswahl-Fenster, in welchem alle interpretierbaren Filetypen angezeigt werden. Nach der Auswahl eines Files wird dessen Inhalt in einem Tabellenkalkulations-Schema präsentiert (siehe Abb. 1.3).

Zuerst wird zwischen Files mit vertikaler Ausrichtung und solchen mit Spalten-Separatoren (meist Komma, wie im Beispiel, evtl. auch Tabulator, Semikolon) unterschieden. Dann kann der einzulesende Datenbereich (Range) in gewohnter Weise angewählt werden. Die Zeilennummern laufen von der ersten bis zur letzten Zeile, die Spaltenbezeichnungen von A bis Z (und weiter AA etc.) Im Beispiel zum Importieren des Files `blklima.txt` ist der Daten-Bereich A2 : P13.

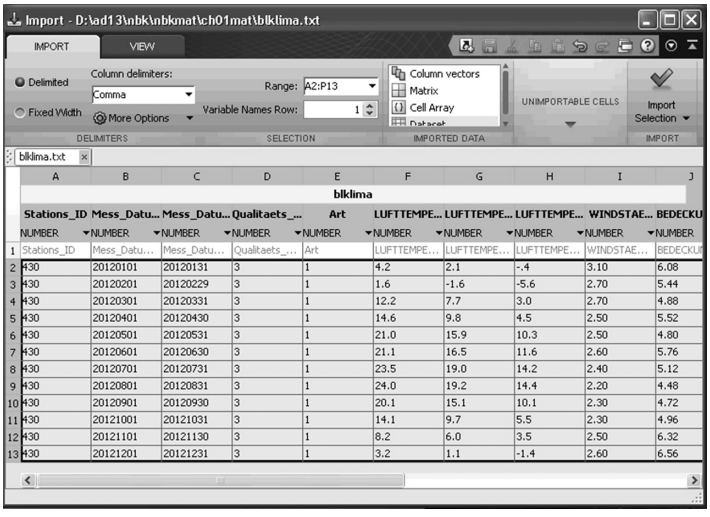

Abb. 1.3 Das „Import Data" Fenster.

Für die Art der Abspeicherung stehen separate Spaltenvektoren, eine Matrix, ein „Cell Array" und ein „Dataset" zur Verfügung.

Ein „Cell Array" ist Matrix-artige Anordnung von Zellen (Boxen, Containern), in denen neben Zahlen auch andere Datentypen enthalten sein können; die wichtigste Anwendung sind dabei Zeichenketten. Im Unterschied zu einer Matrix ist eine Mischung von Datentypen in demselben „Cell Array" erlaubt.

Der neue Datentyp „Dataset" ist eine eigentliche Tabelle: Die Kombination des Tabellen-Inhalts mit den zugehörigen Spalten-Titeln und weiteren Hintergrund-Informationen. Beim Feld „Variable Names Row" des Import-Dialoges kann angegeben werden, in welcher Zeile des Files die Spalten-Titel stehen (im Beispiel = 1).

Dataset ist als Objekt definiert (siehe doc dataset). Von den vielen darauf anwendbaren Methoden (d. h. Funktionen) sind die wichtigsten `display(datasetname)`, `summary(datasetname)`, `size(datasetname)`, sowie `Dmat = double(datasetname)` zum Extrahieren der reinen Zahlen.

Formatiertes Abspeichern und Einlesen

In ein File, das mit dem Befehl

```
filehandle = fopen('filename.ext', 'w')
```

eröffnet wurde, können mit

```
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.

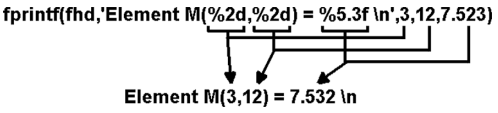

Abb. 1.4 Das Reißverschluss-Prinzip bei fprintf.

Die entsprechenden Sequenzen zum formatierten Einlesen eines Files sind:

```
filehandle = fopen('filename.ext', 'r')
fscanf(filehandle, ' formatstring' , var1, var2, ...)
% ..... ganze Serie von Einlese-Operationen
fclose(filehandle)
```

Die Arbeitsweise dieser formatierten Ein- und 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 zusammengefügte Ausgabezeile sieht dann in diesem Beispiel so aus:

```
Matrixdimensionen: 4 x 6
```

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), sowie c für Einzelzeichen und s für Zeichenketten. Weitere Details erfahren Sie mit `help fprintf`.

Ein weiteres Beispiel zeigt nochmals das Reißverschlussprinzip von `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 $\boldsymbol{M}$ je eine Formatierungsvorschrift im Formatstring (hier alle „%8.4f“). Diese Plätze werden der Reihe nach mit den Werten in der nachfolgenden Variablenliste gefüllt und ergeben die folgende Zeile:

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

Dieselbe von der Sprache C übernommene Formatierung dient bei der Funktion `sprintf` dazu, eine einzelne Zeile, also eine Zeichenkette zu produzieren. Der Formatierungs-Befehl hat dann die analoge Form:

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

Der Parameter mit dem File-Handle fehlt, weil die Zeichenkette direkt in der Variablen `newstring` gespeichert wird.

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.

Wenn es darum geht, die in einer Berechnung verwendeten Abläufe mit den zugehörigen Formeln zu dokumentieren, so kann man unter dem Register PUBLISH im MATLAB Editor eine Fülle von Hilfsfunktionen abrufen, um den Berechnungsverlauf und die Resultate publikationsreif zu formatieren.

Dabei werden die Texte in der Publikation direkt aus den Kommentaren in den bearbeiteten M-Files übernommen. Die Bildung von Abschnitten und das Setzen von zugehörigen Zwischentiteln beruht auf der Verwendung von „Hauptkommentaren“; diese beginnen mit doppeltem Prozentzeichen (%%).

Merkpunkte: Daten Ein- und Ausgabe

- Die einfache **Ein- und Ausgabe** von Daten mit `load` und `save` umfasst das interne .mat-Format und einfach strukturierte Text Formate.
- Das interaktive **Import Data Fenster** bietet vielfältige, an die Tabellenkalkulation angelehnte Eingabe-Möglichkeiten.
- Die **detaillierte Festlegung der Formatierung** von Ein-und Ausgabe ist mit der Funktionsfamilie fscan/fprint möglich.

1.2 Grundlagen der Matrizenrechnung

Matrizen sind mathematische Objekte mit denen wir im täglichen Leben eher 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.

1.2.1 Definitionen und Fachausdrücke

Definition einer Matrix

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.

Die Matrix dieses Beispiels wird als (3×4)-Matrix bezeichnet: Für die Dimensionszahlen gibt man in Klammern die Zeilenzahl, ein „×“ und die Spaltenzahl an.

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 $\boldsymbol{M}$: Da sind einmal die 12 Zahlen selbst, von jeder ist 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 auch noch die **Dimensionszahlen** (Anzahl Zeilen und Anzahl Spalten) zur vollständigen Definition der Matrix $\boldsymbol{M}$. In diesem Fall mit 12 Elementen wären statt 3×4 andere Matrizen mit Dimensionszahlen 2×6, 4×3 und 6×2 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 1×12 entspricht ein Zeilenvektor, den Dimensionszahlen 12×1 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 zugehörigen 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. Für jeden Platz in einer Matrix **muss** eine Zahl definiert sein; dabei ist auch Null erlaubt, fehlende Elemente sind aber illegal.

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.

Abb. 1.5 „Alignements de Kermanio“ bei Carnac, Frankreich.

Das Anordnungsprinzip in einer Matrix

Das in einer Matrix verwendete Anordnungsprinzip ist 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!

Die wichtigsten Begriffe der Matrizenrechnung

In einer Matrix nebeneinander stehende 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 $\boldsymbol{m} \times \boldsymbol{n}$-Matrix, dann hat sie $\boldsymbol{m}$ **Zeilen** und $\boldsymbol{n}$ **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 ($m > n$), bzw. **breite** Matrix ($m < n$).

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** (englisch: „upper triangular matrix“). Dementsprechend hat eine **untere Dreiecksmatrix**, auch **Links-**

Dreiecksmatrix genannt, die von null verschiedenen Elemente nur auf der Diagonalen und unterhalb (links) davon (englisch: „lower triangular matrix“).

Bei beiden müssen also alle Elemente auf der anderen Seite der Diagonalen null sein; auf der mit dem Namen bezeichneten Seite können beliebig von null verschiedene Elemente, aber auch Null-Elemente auftreten.

Beispiele von unteren (Links-) und oberen (Rechts-) Dreiecksmatrizen:

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

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 $1 \times n$ Matrizen entsprechen, und die aus der Vektorgeometrie wohlbekannten **Spaltenvektoren**, meist einfach **Vektoren** genannt, die den $n \times 1$-Matrizen entsprechen.

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

Der letzte hier erwähnte Spezialfall sind die **1×1-Matrizen**, die nur eine Zahl enthalten und in MATLAB automatisch wie einfache normale Zahlen behandelt werden. Sobald bei einer Berechnung eine 1×1-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 1×1-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 $m \times n$ Matrix?
- Wieviele Freiheitsgrade hat eine $n \times n$ Diagonalmatrix?
- Wieviele Freiheitsgrade hat eine obere $n \times n$ 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 $n \times n$-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 ‘×’ 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 $m \times n$ Matrix $\boldsymbol{A}$ also links oben und entsprechend steht a_{mn} 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 4×5-Matrix?

Übung 12–3
Geben Sie die Indizes der Diagonalelemente einer 4×4-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 $\boldsymbol{S}^{\mathrm{T}} = \boldsymbol{S}$ 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.

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

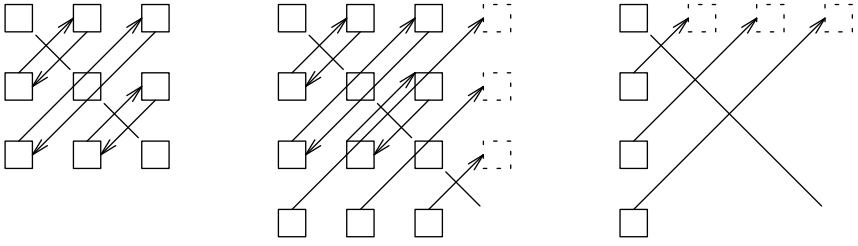

Abb. 1.6 Transponieren einer quadratischen Matrix, einer Rechtecksmatrix und eines Vektors.

Anmerkung Der MATLAB-Operator ’ bedeutet genau genommen 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 ihren Platz, aus 3×4 wird eine 4×3-Matrix):

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

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

1.2.4 Addition und Subtraktion von Matrizen

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

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

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

Die Multiplikation einer Matrix mit einem Skalar

Die Multiplikation einer Matrix $\boldsymbol{A}$ 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 $\boldsymbol{A}'$ der Matrix $\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ und anschließend $\boldsymbol{A}' + \boldsymbol{A}, \quad \boldsymbol{A} - \boldsymbol{A}', \quad (\boldsymbol{A} + \boldsymbol{A}), \quad 2 * \boldsymbol{A}!$

Merkpunkte: Matrizenformen, spezielle Matrizen

- Die Unterscheidung von Matrizen nach ihrer **Form** umfasst hohe, quadratische und breite Matrizen. Nicht quadratische Matrizen heißen Rechtecksmatrizen. Die Extremfälle mit nur einer Spalte/Zeile sind die Vektoren (auch Spaltenvektoren) und die Zeilenvektoren.
- **Spezielle Matrizenformen** von Bedeutung sind die Diagonalmatrizen, die Einheitsmatrizen, die oberen und unteren Dreiecksmatrizen, sowie die Tridiagonalmatrizen.

1.2.5 Das Produkt von zwei Matrizen

Definition des Matrizenproduktes

Das Produkt von zwei Matrizen $\boldsymbol{A} \cdot \boldsymbol{B}$ 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 3×3-Matrizen $\boldsymbol{A}$ und $\boldsymbol{B}$ und zeigen die Entstehung von deren Produktmatrix $\boldsymbol{A} \cdot \boldsymbol{B}$, die wir $\boldsymbol{C}$ nennen.

Vorsicht! Bei der Matrizenmultiplikation ist die Reihenfolge der Faktoren wichtig, $\boldsymbol{A} \cdot \boldsymbol{B}$ ist also im Allgemeinen verschieden von $\boldsymbol{B} \cdot \boldsymbol{A}$.

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

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

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 jedes Mal 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 $\boldsymbol{A}$ d. h. a_{31}, a_{32}, a_{33} mit der gesamten 2. Spalte von $\boldsymbol{B}$, 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 Anzahl Spalten (= 2. Dimensionszahl, = Zeilenlänge) der links stehenden Matrix mit der Anzahl Zeilen (= 1. Dimensionszahl, = Spaltenhöhe) 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 erscheint, zwei Matrizen zu multiplizieren, bei denen diese Bedingung verletzt ist.

Bedingung für die Existenz des Produktes von zwei Matrizen Das Produkt $\boldsymbol{A} \cdot \boldsymbol{B}$ einer $m_1 \times n_1$-Matrix $\boldsymbol{A}$ mit einer $m_2 \times n_2$-Matrix $\boldsymbol{B}$ existiert, falls $n_1 = m_2$ ist. Die innenliegenden Dimensionszahlen müssen also übereinstimmen. Das Resultat er-

hält dann die Dimensionen $m_1 \times n_2$ (d. h. der außenliegenden Dimensionszahlen).

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

Die Resultat-Matrix hat dann die Dimensionszahlen $m_1 \times n_2$, erste Zahl der linken × zweite der rechten Matrix, also die beiden außen liegenden Zahlen.

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: $w'(1 \times 3) \cdot v(3 \times 1) = s(1 \times 1)$

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

Das 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 $r_i(\boldsymbol{A})$ (row(i, A)) schneidet aus der Matrix $\boldsymbol{A}$ (Faktor links) einzelne Zeilen heraus und der Operator $c_j(B)$ (col(j, B)) einzelne Spalten aus der Matrix $\boldsymbol{B}$ (Faktor rechts). Dies ergibt:

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

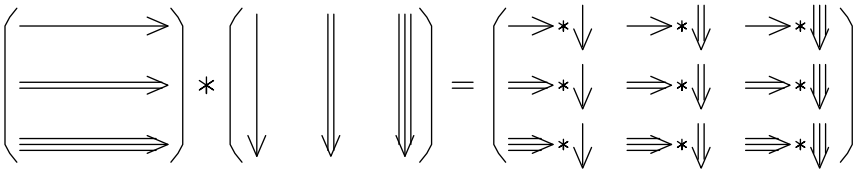

Abb. 1.7 Grafik zum Prinzip der Matrixmultiplikation.

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

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 $(m \times 1)$-Spaltenvektor mit einem $(1 \times n)$-Zeilenvektor ergibt eine $m \times n$-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 jedem Resultat-Element statt der Summe aus mehreren Produkten nur ein einziges Produkt. Dieses besteht aus einem Element des linken Spaltenvektors mal einem des rechten Zeilenvektors. Bildet man das dyadische Produkt eines Vektors mit sich selbst, so entsteht ein Schema analog zu einer Meisterschaftsrunde im Sport mit Hin- und Rückspielen. Nur die Diagonalelemente haben in dieser Analogie keine Bedeutung, weil eine Mannschaft nicht gegen sich selbst spielen kann.

Übung 12–6

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

Merkpunkte: Matrix-Operationen

- Durch **Indizieren** erfolgt der Zugriff auf einzelne Elemente innerhalb einer Matrix. Die Indizes sind die zwei Zahlenangaben von Zeile und Spalte des Elementes.
- **Transponieren** heißt die Operation der Spiegelung einer Matrix an ihrer Diagonalen. Dabei tauschen die beiden Indizes jedes Elementes und die Dimensionszahlen ihre Rollen.
- **Addition und Subtraktion** von Matrizen erfolgt elementweise und setzt daher Matrizen mit identischen Dimensionszahlen voraus.
- Bei der **Multiplikation mit einem Skalar** wird jedes einzelne Element mit dem Skalar multipliziert.
- Das **Skalarprodukt: Zeilenvektor mal Spaltenvektor** liefert die einzelne Zahl mit der Summe aller Produkte aus je einem Element des Zeilenvektors mit dem entsprechenden des Spaltenvektors.
- Das **Matrizenprodukt** enthält alle möglichen Skalarprodukte aus einem Zeilenvektor der linken und einem Spaltenvektor der rechten Matrix. Für die Dimensionszahlen gilt: $(m_1 \times n_1) * (m_2 \times n_2)$ ergibt $(m_1 \times n_2)$ unter der Voraussetzung dass $(n_1 = m_2)$ („inner dimensions must agree“).

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 gibt es also für Matrizen **kein** kommutatives Gesetz.

Im Allgemeinen gilt (Spezialfälle ausgenommen) für zwei Matrizen $\boldsymbol{A}$ und $\boldsymbol{B}$: $\boldsymbol{A} \cdot \boldsymbol{B} \neq \boldsymbol{B} \cdot \boldsymbol{A}$. Das Matrizenprodukt $\boldsymbol{A} \cdot \boldsymbol{B}$ ist i. a. verschieden von $\boldsymbol{B} \cdot \boldsymbol{A}$.

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 2×2 Matrizen lässt sich die Einheitsmatrix relativ einfach bestimmen. Eine 2×2 Matrix mit beliebigen Zahlen a, b, c, d soll durch Multiplikation mit der zu bestimmenden 2×2 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

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

deren Lösungen leicht durch Erraten gefunden werden können.

Es sind dies $x = 1$, $y = 0$, $z = 0$ und $u = 1$.

Im Symbolic-Mode ist diese Bestimmung mathematisch exakt möglich:

```
syms a b c d   x y z u
A = [a b ; c d]  % allgemeine symbolische Matrix
N = [x y ; z u]  % unbekanntes Neutralelement, symbolisch
Gls = A*N == A   % Gleichungssystem (== für "ist identisch")
Nsol = solve(Gls, x,y,z,u) % Nsol ist Strukturvariable
Nsval = [Nsol.x Nsol.y Nsol.z Nsol.u] %Anzeige der Werte
%  liefert   Nsval =  [ 1, 0, 0, 1]
```

Hier sehen wir ein Beispiel einer Strukturvariablen: Der analytische Gleichungs-Löser `solve` speichert mehrfache Lösungen von Gleichungssystemen oder Gleichungen höheren Grades in einer Struktur. Deren Bestandteile sind mit dem Prinzip `hauptname.subname` zugänglich.

Die 2×2 Einheitsmatrix ist also:

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

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

Das Prinzip gilt jedoch ganz allgemein: Die $n \times n$ 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 $n \times n$ 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 durch Matrizen dividieren?

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

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 $\boldsymbol{B}$ 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. Dies ergibt die Definition der „Rechts“-Division und der „Links“-Division:

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

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

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 $\boldsymbol{A}^{-1}$ umschrieben.

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

Die Schreibweise $\boldsymbol{A}\backslash\boldsymbol{B}$ benutzt den in MATLAB für die Links-Division neu eingeführten Operator \, 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 $\boldsymbol{A} \cdot x = b$ hat die Lösung $x = \boldsymbol{A}^{-1} \cdot b$ 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:

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

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 diese 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 $n \times n$-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.

Merkpunkte: Matrix-Division

- Eine **Division durch eine Matrix** $\boldsymbol{A}$ ist eine formale Umschreibung der Multiplikation mit der Inversen $\boldsymbol{A}^{-1}$. Weil die Matrix-Multiplikation nicht kommutativ ist, braucht es zwei verschiedene Operatoren:
  Rechts-Division $\boldsymbol{B}/\boldsymbol{A} = \boldsymbol{B} \cdot \boldsymbol{A}^{-1}$ und Links-Division $\boldsymbol{A}\backslash\boldsymbol{B} = \boldsymbol{A}^{-1} \cdot \boldsymbol{B}$
- Die Linksdivision hat die viel größere Bedeutung durch ihre Anwendung bei der Lösung von linearen Gleichungssystemen:
  `A * x = b` hat die Lösung `x = A \ b`

1.3 Matrizenrechnung mit MATLAB

Matrizenrechnung ist die eigentliche Stärke von MATLAB – die grundlegenden mathematischen Objekte in MATLAB sind Matrizen. Der Name ist ein Akronym für MATrix LABoratory. Benutzen Sie diese Tatsache, um sich durch die Anwendung von MATLAB mit der Matrizenrechnung näher vertraut zu machen!

1.3.1 Einstieg in die Matrizenrechnung mit MATLAB

Für die folgenden einfachen Beispiele arbeiten Sie am besten mit Skripts. D. h. Sie schreiben die Befehle in ein Textfile mit dem Namenszusatz „.m“ und verlangen die Ausführung durch Eintippen des Filenamens (ohne den Zusatz) oder durch das „run“-Icon (grüner Pfeil in der Kopfleiste des MATLAB Editors). Das Einfügen von Kommentarzeilen, die mit `%` beginnen, erleichtert die Übersicht. Mit speziellen Kommentarzeilen, welche mit `%%` beginnen, können Abschnitte gebildet werden, die einzeln ausführbar sind.

Addition und Subtraktion von Matrizen

Geben Sie die 4 Matrizen $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ und $\boldsymbol{D}$ ein (alle 2×2):

```
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 $\boldsymbol{A} + \boldsymbol{B}$, … $\boldsymbol{B} + \boldsymbol{D}$, … $\boldsymbol{A} + \boldsymbol{B} + \boldsymbol{C} + \boldsymbol{D}$!

Ebenso sind die Summen $\boldsymbol{A} + \boldsymbol{A}$, $\boldsymbol{C} + \boldsymbol{C} + \boldsymbol{C}$ 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 $\boldsymbol{A} + \boldsymbol{B} + \boldsymbol{C} + \boldsymbol{D}$ aus dem vorherigen Beispiel soll der Variablen S zugewiesen werden. Es gilt also `S = [1 2 ; 3 4]`. 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 ..`). Das rechte obere Teilfenster zeigt an, welche Variablen im „workspace“ verfügbar sind, ebenso die Kommandos `who` und `whos`. Den Inhalt einer Variablen lässt man sich anzeigen, indem man ihren Variablennamen eintippt (natürlich ohne Semikolon am Ende!).

Matrizenmultiplikation – möglich oder nicht?

Erzeugen Sie neben der 2×2-Matrix $\boldsymbol{S}$ aus dem vorherigen Beispiel noch eine 3×2-Rechtecksmatrix $\boldsymbol{W}$ und einen 2×1-Vektor $\boldsymbol{v}$. Benutzen Sie den Transpositionsoperator **()'**, um den vorerst als Zeile eingegebenen Vektor $\boldsymbol{v}$ 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 nun aus, vielleicht erst nachdem sie sich überlegt haben, was das Ergebnis sein sollte, welche der folgenden Operationen legal sind, und welche Dimensionszahlen die Resultate im erlaubten Fall aufweisen.

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

Erzeugen einer Diagonalmatrix

Mit dem Befehl `Dia = zeros(5)` können sie als Start eine 5×5-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.

Für diese Aufgabe existiert eine Bibliotheksfunktion `diag()`. Diese erwartet für die Werte entlang der Diagonalen einen Vektor oder Zeilenvektor der passenden Länge: z. B. `diag(1:5)` oder `diag(1:n)` (mit vorher definiertem `n`).

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. Beachten Sie, dass hier mit `rand(1,5)` nur ein 1×5 Vektor und keine Random-Matrix erzeugt werden muss.

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 $\boldsymbol{A} \cdot \boldsymbol{A}^{\mathrm{T}}$ ergibt sich durch das Transponieren aus der Spaltenzahl der links stehenden Matrix die Zeilenzahl der rechts stehenden. Bei $\boldsymbol{A}^{\mathrm{T}} \cdot \boldsymbol{A}$ ergibt sich durch das Transponieren bei $\boldsymbol{A}^{\mathrm{T}}$ die Zahl von Spalten, die der Zahl von Zeilen bei $\boldsymbol{A}$ entspricht. In beiden Fällen ist die Bedingung für das Multiplizieren der beiden Matrizen erfüllt. (Die innen aneinanderstoßenden Dimensionszahlen sind gleich.)

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 $\boldsymbol{A} \cdot \boldsymbol{A}^{\mathrm{T}}$ als auch $\boldsymbol{A}^{\mathrm{T}} \cdot \boldsymbol{A}$, 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:

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

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 $\boldsymbol{v}^{\mathrm{T}} \cdot \boldsymbol{v}$; es ergibt sich das Skalarprodukt von $\boldsymbol{v}$ mit sich selbst. Die andere Reihenfolge der Faktoren, $\boldsymbol{v} \cdot \boldsymbol{v}^{\mathrm{T}}$ liefert bei Vektoren der Länge n eine $n \times n$ Matrix, das dyadische Produkt von $\boldsymbol{v}$ mit sich selbst.

Das Neutralelement der Addition von Matrizen

Das neutrale Element bezüglich einer Operation (hier der Addition) lässt eine beliebige Matrix unverändert, wenn sie mit diesem verarbeitet wird: $\boldsymbol{A} + N_{\mathrm{add}} = \boldsymbol{A}$. Für gewöhnliche Zahlen ist das Neutralelement der Addition bekannt, es ist die Null. Weil die Addition von Matrizen elementweise erfolgt, besteht das Neutralelement der Addition aus lauter Nullen, es ist die Nullmatrix.

Ein einfaches Skript kann dazu einige Beispiele vorführen:

```
syms a b c d    % fuer symbolische 2x2 Matrix
S = [a b ; c d] % Definition der symbolischen Matrix
Na = zeros(2)   % Nullmatrix
Sp = S + Na     % Summe S + Nullmatrix gibt wieder S
Z = rand(2)
Zp = Z + Na     % Summe Z + Nullmatrix gibt wieder Z
ndim = 5        % Waehle irgend eine Dimensionszahl
```

```
M = rand(ndim)
Mp = M + zeros(ndim) % Summe M + zeros(ndim) = M
```

Neutralelemente der Multiplikation bei 2 × 2 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

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

- und für das Rechts-Neutralelement der Multiplikation

$$\boldsymbol{A} * N_{\text{mult-R}} = \boldsymbol{A}: \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix} * \begin{pmatrix} p & q \\ r & s \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Führen Sie in beiden Fällen die Matrixmultiplikation von Hand aus und bestimmen Sie jeweils die vier Gleichungen zur Bestimmung von x, y, z, u bzw. p, q, r, s. (Das Ausführen von Matrizenmultiplikationen von Hand verstärkt Ihr Verständnis für diesen meist noch nicht vertrauten 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 wurde bereits erwähnt: Die Neutralelemente $N_{\text{mult-L}}$ und $N_{\text{mult-R}}$ sind dieselbe Einheitsmatrix (Diagonalmatrix mit lauter Einsen auf der Diagonalen).

$$N_{\text{mult-L}} = N_{\text{mult-R}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Allgemein besitzt die $n \times n$ 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.

Auch bei quadratischen Matrizen höherer Dimension ist die links- und rechts-Neutralität der Einheitsmatrix für die Multiplikation mit einem einfachen Skript demonstrierbar.

```
ndim = 5        % oder eine andere Dimensionszahl
I = eye(ndim)   % Einheitsmatrix (ndim x ndim)
M = rand(ndim)  % (ndim x ndim) Zufallszahlen
MtestL = I*M    %  I von links multiplitiert
MtestR = M*I    %  I von rechts multipliziert
MtestL - MtestR % muss Nullmatrix ergeben
```

Bei Rechtecksmatrizen müssen die Einheitsmatrizen für die Multiplikation von links und von rechts her verschiedene Dimensionszahlen aufweisen, aber natür-

lich beide quadratisch sein, wie im folgenden Beispiel:

```
M = [ 1 2 3 4 5 ; 1 0 2 0 3]
IdL = eye(2)
IdR = eye(5)
MtL = IdL * M
MtR = M * IdR
```

Merkpunkte: Matrix-Multiplikation

- **Matrixmultiplikationen mit illegalen Dimensionen** ergeben eine Fehlermeldung „inner dimensions must agree". Häufig, vor allem bei Vektoren, hat man vergessen den Punkt-Operator `(.*)` zu verwenden oder eine Transposition zu verlangen.
- **Einheitsmatrizen**, also Diagonalmatrizen mit lauter Einsen, sind immer sowohl Links- als auch Rechts-Neutralelement der Multiplikation.
- Das **Produkt Matrix mal ihre Transponierte** ist immer möglich und liefert eine symmetrische quadratische Matrix.

1.3.2 Indizieren in MATLAB

Die Indizierung eines einzelnen Matrix-Elementes erfolgt durch die Angabe des Paars von Indexwerten, also von Zeilen und Spaltennummer.

Die mathematische Formulierung M_{34} entspricht in der MATLAB Befehlssprache der Angabe `M(3,4)`. Die beiden Indizes stehen in runden Klammern, getrennt durch Komma.

Index-Bereiche

Um das Arbeiten mit Matrizen, Tabellen und Vektoren zu erleichtern gibt es in MATLAB eine Reihe von Möglichkeiten ganze Index-Bereiche zu definieren:

Formulierung	Bedeutung
`M(1:2,1:2)`	1. und 2. Zeile aus 1. und 2. Spalte
`M(:,3)`	nur Doppelpunkt = alle Zeilen in Spalte 3
`M(5,:)`	nur Doppelpunkt = alle Spalten in Zeile 5
`U=M(3:5,:)`	Zeilenbereich 3 bis 5, alle Spalten
`M(2:end,3)`	2. bis letzte Zeile in Spalte 3
`M(:,1:end-1)`	alle Zeilen, 1. bis zweitletzte Spalte

Mit dieser Konstruktion ist es auch möglich, einzelne Zeilen oder Spalten durch Zuweisung einer leeren Matrix an diesem Bereich zu löschen, wie in:

```
A(3,:) = []    oder    A(:,5) = []
```

Darin wurde die Formulierung `[]` (für leere Matrix) verwendet.

Dieselbe Angabe von Index-Bereichen funktioniert auch bei Vektoren, für die nur ein Index-Wert anzugeben ist. Als Beispiel werden die Differenzen der aufeinanderfolgenden Werte eines Vektors berechnet:

```
dx = x(2:end) - x(1:end-1)
```

Index-Vektoren

An der Stelle eines Index-Wertes sind neben der Angabe einer einzelnen Zahl, oder eines Bereiches auch ganze Vektoren von Indexwerten möglich. (Die Angabe des Vektors erfolgt in gewohnter Weise in eckigen Klammern oder mit einer vorher definierten Variablen.) In den Index-Vektoren dürfen Indexwerte mehrfach vorkommen.

Ein einleuchtendes Beispiel ist das Schließen eines Linienzuges:

```
xcl = x([1:end 1]) ; ycl = y([1:end 1])
```

Etwas komplexer ist das Zeichnen eines Quadrates mitsamt seinen Diagonalen als durchgehender Linienzug:

```
qxl = qx([1:end 1 3 4 2]) ; qyl = qy([1:end 1 3 4 2])
```

Eine spezielle Art von Index-Vektoren sind die **logischen Index-Vektoren**. Dabei wird am Platz des Index-Wertes ein Vektor aus logischen Variablen angegeben. Dies selektiert alle Elemente, an deren Platz der Wert „true“ im logischen Vektor steht. Ein logischer Vektor wird durch eine Vergleichs-Operation erzeugt, kann aber auch aus einer Kette von numerischen 0 und 1 Werten durch `vlog = logical(vnum)` gewonnen werden. Die folgenden Befehle finden alle Zahlen von 1 bis 100, welche sowohl durch 3 als auch durch 5 teilbar sind:

```
vo = 1:100
iv = (mod(vo,3) == 0 ) & (mod(vo,5) == 0)
v35 = vo(iv)
```

Merkpunkte: Index-Bereiche

- Durch die Angabe von ganzen **Index-Bereichen** statt einzelnen Indexwerten können Teilmatrizen oder Vektoren aus einer Matrix herausgeholt werden, z. B. `A(:,2)` ganze 2. Spalte, `W(3,:)` ganze 3. Zeile, `v(1:end-1)` ganzer Vektor außer letztes Element.
- Anstelle eines Wertes kann bei der Index-Angabe ein **Indexvektor** stehen, eine Aufzählung aller anzusprechenden Indizes.

1.3.3 Beispiele zur Schleifenprogrammierung

Spezielle Matrizen ergeben gute Beispiele für die Programmierung von Schleifen.

Eine Einheitsmatrix erzeugen Natürlich ist in MATLAB die Definition einer $n \times n$ Einheitsmatrix am einfachsten mit der Funktion `eye(n)` möglich. Die detaillierte Definition mit einem einfachen Schleifenprogramm ist aber recht instruktiv:

```
I = zeros(ndim)  % quadratische Nullmatrix
for k=1:ndim     % Laufvariable k von 1 bis ndim
  I(k,k) = 1;    % Eins setzen an Zeile k und Spalte k
end
I                % Kontrollausgabe
```

Obere Dreiecksmatrix Für die Aufgabe, eine obere Dreiecksmatrix mit Einsen zu füllen, sind bereits einige Überlegungen erforderlich:

```
R = zeros(ndim);
% der erste (Zeilen-)Index muss von 1 bis ndim
%    alle Zeilen durchlaufen
for zeil = 1:ndim
%   der Spaltenindex beginnt jeweils erst in der Diagonalen
%   das entspricht in der zeil-ten Zeile dem Wert 'zeil';
%   von dort laeuft er bis zu 'ndim', dem rechten Rand.
   for spal = zeil:ndim
      R(zeil,spal) = 1;
   end
end
R  % Kontrollausgabe nach erfuellter Aufgabe
```

Übung 13–1 Diagonalmatrix
Im Beispiel der Einheitsmatrix war nur eine einfache Schleife notwendig, ebenso genügt eine einfache Schleife zum Definieren einer Diagonalmatrix mit den Werten 1,2, …, ndim in der Diagonalen. Versuchen Sie es und testen Sie Ihr Mini-Programm mit verschiedenen Dimensionen!

Übung 13–2 Indexwertmatrix
Ein kleines Doppelschleifenprogramm, das sehr nützlich ist zu Demonstrationszwecken, ist das folgende:
Füllen Sie eine Matrix mit Zahlenwerten, welche gerade dem Indexpaar entsprechen, aufgefasst als zweistellige Dezimalzahl. (Dieses Programm wird bei $n > 9$ keine sinnvollen Werte mehr liefern.) Die so definierte spezielle Matrix wird im Folgenden unter dem Namen Indexwertmatrix mehrfach verwendet.

1.3.4 Turmmatrizen (Permutationsmatrizen)

Für Matrizen zur Permutation (Vertauschung) von Zeilen bzw. Spalten habe ich die Bezeichnung „Turmmatrizen“ gehört, vor vielen Jahren bei Prof. Eduard Stiefel an der ETH Zürich. Damit wird beschrieben, dass die Einsen in diesen Matrizen so angeordnet sind, dass sich Türme an diesen Plätzen auf einem Schachbrett gegenseitig nicht bedrohen könnten. (In jeder Zeile und in jeder Spalte gibt es nur eine einzige Eins.) Sonst enthalten diese Matrizen lauter Nullen.

Im Beispiel

$$\boldsymbol{T} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

werden durch die Multiplikation $\boldsymbol{T} \cdot \boldsymbol{M}$ einer beliebigen Matrix $\boldsymbol{M}$ mit $\boldsymbol{T}$ von links her die Zeilen von $\boldsymbol{M}$ wie folgt vertauscht:

- die ursprünglich 2. Zeile wird in die 1. Zeile verschoben,
- die ursprünglich 3. Zeile wird in die 2. Zeile verschoben,
- die ursprünglich 1. Zeile wird in die 3. Zeile verschoben.

Ganz allgemein sind Matrizen dieses Typs Permutationsmatrizen, d. h. Matrizen, welche Vertauschungen hervorrufen.

Bei der Multiplikation von **links** her mit einer Turmmatrix werden **Zeilen** vertauscht.

Wird eine gegebene Matrix von **rechts** her mit einer Turmmatrix multipliziert, so werden deren **Spalten** vertauscht.

Spechtmatrix Um die produzierte Vertauschung leichter verständlich zu machen, kann man sich die Turmmatrizen zusammengesetzt denken aus lauter „Specht“-Matrizen. Diese haben lauter Nullen, außer einer einzigen Eins. Spechtmatrizen nenne ich sie, weil sie eine einzige Zeile, bzw. Spalte aus der mit ihr multiplizierten Matrix herauspicken und an einem neuen Ort platzieren. Außerhalb der am neuen Ort platzierten Werte sind alle Elemente der Resultatmatrix bei Multiplikation mit einer Spechtmatrix Null.

Eine Rezension der ersten Auflage hat diesen Namen als unwissenschaftlich abgelehnt. Die bildliche Vorstellung des Spechtes, der mit spitzem Schnabel einen Wurm aus der Rinde herauspickt, und diesen andernorts seinem Nachwuchs füttert, trägt jedenfalls zum Verständnis dieser elementaren Matrixmultiplikation bei. Darum bleibe ich bei dieser Bezeichnung.

Die Wirkung der Spechtmatrix zur ersten Zeile des obigen Beispiels ergibt somit:

$$\begin{pmatrix} m_{21} & m_{22} & m_{23} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$$

In die erste Zeile des Resultats (= Zeile der „Eins“) wird bei Multiplikation mit der Spechtmatrix von links her die zweite Zeile der Matrix $\boldsymbol{M}$ platziert (entsprechend der Spaltennummer der „Eins“).

Analog verläuft das Herauspicken und Neuplatzieren von Spalten bei der Multiplikation einer Matrix mit einer Spechtmatrix von rechts her: die Zeile, in der die Eins steht, bestimmt, welche Spalte herausgepickt wird, während die Spalte mit der Eins die neue Platzierung definiert.

Zusammengefasst Bei Multiplikation von links (Zeilen vertauschen) bestimmt die Spalte der Eins, welche Zeile herausgepickt wird und die Zeile der Eins wohin die herausgepickte Zeile platziert wird.

Beim Vertauschen von Spalten, der Multiplikation von rechts, bestimmt die Zeile der Eins das Herauspicken und die Spalte das Platzieren.

Allgemein: Der innenliegende Index der Eins bestimmt das Herauspicken, der außenliegende das Platzieren.

Hier ergibt sich ein ganz neuer Zugang zur Einheitsmatrix: Jede Eins auf der Diagonalen platziert eine Zeile/Spalte dort wo sie herausgepickt wurde.

Potenzieren von Turmmatrizen

Für alle Turmmatrizen der Dimension n gilt: Nach höchstens n-fachem Potenzieren erscheint wieder die Einheitsmatrix. $\boldsymbol{T}^k = \boldsymbol{I}$, wobei ($k \leq$ ndim). Dies kann einfach begründet werden: Nach einer genügenden Anzahl von aufeinander folgenden Vertauschungen landen die Zeilen wieder an ihrem ursprünglichen Platz. Für den trivialen Fall der Identität $\boldsymbol{I}$, welche auch eine Turmmatrix ist, gilt $\boldsymbol{T}^1 = \boldsymbol{I}^1 = \boldsymbol{I}$.

Einen interessanten Spezialfall von Turmmatrizen stellen die elementaren Vertauschungen eines Zeilenpaares (k, l) dar: Außer T_{kk} und T_{ll} sind alle Diagonalelemente Eins. T_{kk} und T_{ll} sind Null, aber dafür sind T_{kl} und T_{lk} Eins. Für elementare Vertauschungen gilt $\boldsymbol{T}^2 = \boldsymbol{I}$; diese Matrizen sind gerade ihre eigenen Inversen.

Die Indexwertmatrix aus der Übung zum Schleifenprogrammieren eignet sich hervorragend, um alle diese Herauspick- und Umplatzierungseffekte bei Multiplikation mit Specht- oder Turmmatrizen am Bildschirm zu verfolgen.

Weitere interessante Beispiele von Turmmatrizen sind die „scroll-up“ und „scroll-down“ Matrizen. Bei diesen werden die Zeilen einer Matrix um eine Zeile nach oben bzw. unten verschoben und die herausfallende Zeile wird am anderen Ende wieder eingefügt. Versuchen Sie solche „scroll-up“ und „scroll-down“ Matrizen mit einem kleinen Programm zu erstellen (Minimal-Dimension 3, besser 4). Anschließend können Sie deren Wirkung wieder mit der Indexwertmatrix prüfen.

Zu allen Turmmatrizen ist die zugehörige Inverse besonders einfach zu finden: Weil diese Matrizen orthogonal sind, ist ihre Transponierte gerade ihre Inverse.

1.3.5 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 $\boldsymbol{A}\cdot x = b$ durch $x = \boldsymbol{A}^{-1}\cdot b$, wobei die Multiplikation mit $\boldsymbol{A}^{-1}$ 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
```

Lineare Gleichungssysteme in Matrixform bringen

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

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

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.

Merkpunkte: Lineare Gleichungssysteme in Matrizenform

- In der **Matrizenform** $\boldsymbol{A} \cdot \boldsymbol{x} = b$ entspricht die Matrix $\boldsymbol{A}$ den Koeffizienten der Unbekannten; $\boldsymbol{x}$ ist der Vektor mit allen Unbekannten, und $\boldsymbol{b}$ ist der Vektor mit den konstanten Termen (rechte Seiten).
- **Voraussetzung** zur Bestimmung der Matrix ist die Ausrichtung der Unbekannten in Spalten. Eine fehlende Unbekannte führt dann zu einem Matrix-Element Null und eine Unbekannte ohne Vorfaktor zu einem Element 1 oder -1, je nach deren Vorzeichen.

1.3.6 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 $n \times 2$-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 53×7-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 bei der 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. Datensätze aus dem Programm Excel der Firma Microsoft sind sehr einfach mit der Importfunktion in MATLAB zu importieren.

Die häufigste Verarbeitung dieser Art von Daten ist die Bildung von Zeilen- und Spaltensummen (abgesehen von grafischen Darstellungen).

Die Funktion `spsvec = sum(M)` liefert einen Zeilenvektor mit allen Spaltensummen. Um einen Spaltenvektor mit den Zeilensummen zu erhalten muss bei der Anwendung der Summenfunktion zweimal der Transpositions-Operator angewandt werden. `zsvec = sum(M')'`.

```
M = [ 11 12 13 14; 21 22 23 24 ; 31 32 33 34]
spsvec = sum(M)
zsvec = sum(M')'
sum(sum(M))
sum(spsvec)
```

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, und damit einer eigentlichen antisymmetrischen Matrix, 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 $\boldsymbol{d}_{jk} = -\boldsymbol{d}_{kj}$ (bzw. $\boldsymbol{D}^{\mathrm{T}} = -\boldsymbol{D}$). 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: $\boldsymbol{d}_{kk} = -\boldsymbol{d}_{kk}$ hat nur die Lösung $\boldsymbol{d}_{kk} = 0$.

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 13–3 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 Schritte zum eigenen Programm

1.4.1 Skript-M-Files und Funktions-M-Files

MATLAB-Programme (Skripts, Funktionen, Klassendefinitionen) werden in sogenannten M-Files festgehalten. Das sind reine Textfiles mit „.m“ als Namenszusatz (Namenserweiterung). Der Namenszusatz wird klein geschrieben, obwohl die Bezeichnung „M-Files“ ein großes M enthält.

Der Hauptteil des Dateinamens darf Groß- und Kleinbuchstaben, sowie Zahlen (aber keine Bindestriche! und keine Leerschläge) enthalten.

Der Name muss mit einem Buchstaben beginnen. Bei einer Funktion und einer Klasse muss der Aufruf auch bezüglich Groß- und Kleinschreibung exakt mit der Deklaration in der `function`-Zeile bzw. `classdef`-Zeile 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. Benutzen Sie diese Option, um die Hilfefunktion auch für Ihre eigenen M-Files zu aktivieren!

Kommentarzeilen, welche mit zwei Prozentzeichen **%%** beginnen, dienen zur Bildung von Abschnitten in einer umfangreichen Berechnung.

Skript M-Files

Als einfaches Beispiel für ein Skript-M-File dient der Geradenfit: Durch eine Anzahl gegebene Punkte, ausgedrückt durch x-y-Koordinatenpaare, soll eine möglichst gut passende (englisch: well fitting) Gerade gelegt werden. Die dabei verwendete Bibliotheksprozedur `polyfit(xvec,yvec,pgrad)` kann neben einer Geraden (pgrad = 1) auch Polynom-Kurven beliebigen Grades durch eine Schar gegebene Punkte legen. Zum Vergleich wird auch noch eine kubische Parabel (Polynom 3. Grades) gefittet.

```
%% Geraden- und Polynomfit
%% synthetische Punkte aus sinus
x = (-1:0.1:1)*pi/2;
y = sin(x);
%% zeichne x,y in leere Figur 1, Bild bleibt stehen
figure(1); clf; plot(x,y,'o-'); hold on
format compact    % Ausgabe eng auf Bildschirm
%% Geradenfit und Polynomfit 3. Grades
pc1 = polyfit(x,y,1)   % Fit 1. Grades = Gerade
yg1 = polyval(pc1,x) ;  plot(x,yg1,'k'); % Farbe blac'k'
pc3 = polyfit(x,y,3)    % fit 3. Grades
yp3 = polyval(pc3,x) ;  plot(x,yp3,'r'); % Farbe 'r'ed
hold off
```

Darin sehen wir mehrere Anwendungen von Funktionen aus der MATLAB Bibliothek. Auf dem Überblick über die verfügbaren Funktionen und dem Verständnis von deren Arbeitsweise gründet der erfolgreiche MATLAB Einsatz.

Der Eingabeparameter für Sinus ist ein Winkelwert in Bogenmaß, das Resultat ist der zugehörige Sinusfunktionswert. In MATLAB-Funktionen kann statt einem einzelnen Wert meistens ein ganzer Vektor von Werten eingegeben werden. Als Resultat wird dann ein passender Vektor von Funktionswerten zurückgegeben.

Die Punktepaare von Winkel- und Sinuswerten spielen in diesem Beispiel die Rolle der zu analysierenden Daten. In Ihrer Anwendung ersetzen Sie diese einfach durch Ihre eigenen Daten, die Sie von einem File mit `load` oder mit der Import-Funktion einlesen.

Der Aufruf von `pc1=polyfit(x,y,1)` liefert die zwei (= pgrad + 1) Koeffizienten der gefitteten Geraden. Diese Gerade hat die Funktionsgleichung `yg = pc1(1)*x + pc1(2)`. Anschließend liefert `yg1=polyval(pc1,x)` die Funktionswerte der Geraden (des „Polynoms" `pc1`) an allen Werten des $\boldsymbol{x}$-Vektors. Analog liefert `pc3=polyfit(x,y,3)` vier Koeffizienten des gefitteten Polynoms 3. Grades, und `yp3=polyval(pc3,x)` berechnet die Werte dieser Kurve 3. Grades (auch kubische Parabel). Die Gerade wird schwarz („k") und die Kurve 3. Grades rot („r") in die mit `hold on` offen gehaltene Figur hinein gezeichnet.

Eigene Funktions-M-Files

Viele Probleme sind viel klarer lösbar, indem man eigene Funktionen erzeugt, sogenannte **Funktions-M-Files**. Diese können dann in einer Serie von Aufrufen mit verschiedenen Eingabeparametern die jeweils analoge Berechnung durchführen.

Konventionen für Funktions-M-Files

1. 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.
2. Der Funktionsname muss mit dem Filenamen übereinstimmen (ohne die Erweiterung .m).
3. Den Rückgabeparametern müssen im Innern der Funktion Werte zugewiesen werden.

Als Beispiel berechnet die folgende Funktion die Oberfläche und das Volumen einer Kugel:

```
  function [oberfl, vol] = Kugeldaten(R)
% Diese ersten Kommentarzeilen erscheinen bei 'help Kugeldaten'
% Beispiel einer Funktion mit 2 Rueckgabe-Parametern
% function [oberfl,vol] = Kugeldaten(R)
% So kann man die Signatur kurz ueberpruefen, weiss also wie
% Aufrufen, und welche Ein- Ausgabe-Parameter einsetzen
    oberfl = 4*pi*R.^2;
    vol = 4*pi*R.^3/3;
  end  % end zu 'function' fakultativ aber empfohlen
```

In MATLAB ist es möglich, mehrere Rückgabeparameter zu verwenden. Um diese Möglichkeit auszunutzen muss eine Gruppe von Rückgabeparametern, durch Kommata getrennt, bei der Deklaration in der „function"-Zeile durch ein Paar von eckigen Klammern zusammengefasst werden. Ebenso müssen die Rückgabeparameter beim Funktionsaufruf, also deren Anwendung in eckigen Klammern stehen.

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, ...)
```

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 Zwischenresultate 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 Rückgabeparameter sind wieder Variablennamen, die der Arbeitsoberfläche bekannt sind.

Ein Kugelbehälter als Beispiel

Erarbeiten der Formel Die Flüssigkeitsmenge in einem kugelförmigen Behälter in Abhängigkeit vom Füssigkeitspegel ergibt ein lehrreiches Beispiel für ein selbst konstruiertes Funktions-M-File. Zuerst wird mit einem Skript im symbolischen Mode von MATLAB die zugehörige Formel bestimmt. Das Erarbeiten einer Formel ist eine typische Anwendung des symbolischen Modes.

```
%% symbolisches Skript teilvolformel.m
  % zum Erarbeiten der Formel Teilvolumen(Fluessigkeitspegel)
syms R  z  h % Kugelradius, Hoehe der Schnittebene, Pegel
%% A) mit bestimmtem Integral
  rho = sqrt(R^2 - (R-z)^2)  % Kreisradius in Schnittebene
  %  bestimmtes Integral ueber Kreisflaeche(z)*dz, von 0 bis h
  tvol = int(rho^2*pi,0,h)   % Formel fuer das Teilvolumen(h)
%% B) stereometrische Herleitung
  okseg = 2*pi*R*h           % Oberflaeche Kugelsegment
  % Kugelsektor-Volumen = Oberflaeche * R/3
  ksekt = okseg*R/3
  % ergaenzender Kegel um Kugelsegment zu erhalten
  % Radius bei Schnitthoehe h
  rhoh = subs(rho, z, h)     % ersetze z durch h in rho(z)
  keg = pi*rhoh^2 * (h-R)/3
  %  Kugelsegment = Kugelsektor + Kegel
```

```
  tvolster = ksekt + keg
  tvssimp = simplify(tvolster)% Vereinfachen zum Vergleich
%% Test fuer Halbkugel und volle Kugel h=R und h=2*R
  halbvol = subs(tvol, h, R)  , gesvol = subs(tvol, h, 2*R)
```

Die Bildschirmausgabe beim Aktivieren des obenstehenden symbolischen Skripts enthält die folgenden Resultatformeln:

```
tvol =
(pi*h^2*(3*R - h))/3
tvolster =
(2*pi*R^2*h)/3 - (pi*(R^2 - (R - h)^2)*(R - h))/3
tvssimp =
(pi*h^2*(3*R - h))/3
halbvol =
(2*pi*R^3)/3
gesvol =
(4*pi*R^3)/3
```

Umsetzen in eine Funktion Damit ist die Formel für das Teilvolumen in Abhängigkeit vom Pegel h bekannt;

$$t_{\text{vol}} = \pi \cdot h^2 \cdot \frac{3R - h}{3}$$

Die Umsetzung als Funktions-M-File ist an die drei Regeln gebunden:

1. Die erste Zeile mit der function-Deklaration enthält den Rückgabeparameter teilvol, das „=“, den Namen und die beiden Eingabeparameter Kugelradius R und Flüssigkeitspegel h.
2. Die Funktion hat den Namen `kugelteilvol`, das bedingt, dass das M-File `kugelteilvol.m` heißen muss.
3. Dem Rückgabeparameter tvol wird im Inneren der aus R und h nach der Formel berechnete Wert zugewiesen.

Wir halten uns an die „best practice“, die Signatur und die prinzipielle Funktionsweise der Funktion mit Kommentaren ganz am Anfang zu dokumentieren. Diese Kommentare sind mit `help kugelteilvol` abrufbar.

Das Vermeiden der Bildschirm-Ausgabe von Zwischenresultaten mit dem Strichpunkt am Ende der Zeile ist bei Funktionen üblich, aber nicht zwingend.

Durch die Verwendung der Punkt-Operatoren `.^` und `.*` bei den Operationen, die `h` enthalten kann für `h` ein ganzer Vektor von Höhenwerten eingegeben werden. Das Resultat ist dann auch ein Vektor mit den zugehörigen Volumenwerten.

Die Zeile mit `end` am Schluss ist fakultativ, wird aber dringend empfohlen.

```
function [ teilvol ] = kugelteilvol( R, h )
%  function [ teilvol ] = kugelteilvol( R, h )
%  berechnet das Teilvolumen der Fluessigkeit in
%  einer Kugel mit Radius R unterhalb des Pegels h
teilvol = pi*h.^2.*(3*R - h)/3;
%  Zusatz zur robusten Programmierung: Melden von
%   sinnlosen h-Werten außerhalb 0 < h < 2*R
```

```
    if min(h) < 0 | max(h) > 2*R
      warning('Aufruf kugelteilvol mit h außerhalb Bereich')
    end
end
```

Anwenden der Funktion Die fertige Funktion soll nun auf ein konkretes Beispiel angewandt werden. Als Längeneinheit wird der wenig gebräuchliche Dezimeter verwendet; damit erhält man das Volumen in Kubikdezimetern, also in Litern. Ein typischer kleiner Milchtransporter hat einen Durchmesser von 1,5 m, also einen Radius von 7,5 dm. Für die Höhenwerte wird ein fein unterteilter Vektor erzeugt. Damit kann das Teilvolumen als Funktion der Füllhöhe anschließend gezeichnet werden.

```
% Skript kugelbehaelter.m zeigt Volumen(Fuellhoehe)
% R, h in Dezimetern ergibt Volumen in Litern
  R = 7.5
  hvec = (0:0.02:2)*R;
  teilvolvec  = kugelteilvol( R, hvec );
  plot(hvec, teilvolvec)
```

Merkpunkte: Skript- und Funktions-M-Files

- Bei Skripts und bei Funktionen erfolgt der **Aufruf mit dem Filenamen**, aber ohne die Erweiterung „`.m`“
- Funktions-M-Files müssen die **drei Bedingungen** erfüllen: Namensgleichheit Funktionsname mit Filename ohne „`.m`“ Deklarationszeile mit Codewort „function“, Rückgabeparameter „=“ Funktionsname (Eingabeparameter).
  Zuweisung von Werten an die Rückgabeparameter im Innern.
- Die Kurzdokumentation durch Kommentarzeilen, die mit % starten, kann mit „help filename“ angezeigt werden und ist daher von großem Nutzen.
- Im Rechenablauf von Funktionen sollten keine Zwischenresultate auf dem Bildschirm erscheinen. Daher sind die Befehle mit Strichpunkt abzuschließen.

Einzeilige Funktionen

Berechnungen, die nur eine einzige Befehlszeile benötigen, sind als sogenannte „anonymouns functions“ mitten im Programmtext definierbar. Sie können anschließend mit beliebigen aktuellen Parametern aufgerufen werden. Die Berechnung des Radius aus den x- und y-Koordinaten dient hier als Beispiel:

```
radius = @(x,y) sqrt(x^2+y^2) % Deklaration
r1 = radius(1,1) ; r2 = radius(3,4) ; % Auswertungen
```

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, Aufruf: nfakultaet = fakurekur(n)
% 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.

1.4.2 Objekt-Orientiertes Programmieren

Ein erster Einblick in das Konzept des Objekt-Orientierten Programmierens ist auch für den reinen Anwender wichtig. Die Handhabung von komplexen Anwendungen und ganz besonders das Erzeugen von grafischen Darstellungen wird dadurch leichter verständlich.

Mit steigender Komplexität erreicht das Konzept von Funktionen seine Grenzen. Man stelle sich einen Funktionsaufruf vor, bei dem man sechs oder gar zehn Eingabeparameter angeben muss! Unpraktisch und fehleranfällig. Die Antwort der Informatik war das Konzept von Objekten. Die meisten, für eine Aktion, einen

Funktionsaufruf benötigten Parameterwerte werden dem Objekt bei dessen Erzeugung bereits übergeben (oder als Standard festgelegt) und bleiben im Inneren gespeichert. Ein Objekt erlaubt es auch verschiedene Aktionen, mit jeweils eigenen Funktionsaufrufen, auf denselben Daten durchzuführen.

Die Welt der Objekt-Orientierten Programmierung verwendet eine Anzahl von neuen Fachausdrücken. Das stellt eine unnötige Hemmschwelle dar. Wenn man sich einige Schlüsselworte merkt, sind die Konzepte leicht zu verstehen.

- **Klassendefinition (classdef)** nennt man den Quellcode (das M-File) welcher die Eigenheiten und das Funktionieren eines Objektes festlegt.
- Die **properties** bezeichnen die im Innern eines Objektes gespeicherten **Daten**.
- **Methoden (methods)** heißen die altbekannten Funktionen, von denen ein Objekt immer mehrere anbietet, von außen aufzurufen, oder für rein interne Anwendung.
- Mit **Konstruktor (constructor)** bezeichnet man die spezielle Methode, mit der ein Objekt erzeugt wird. Ein Objekt mit den in der Klasse festgelegten Eigenschaften wird in die Welt gesetzt (instantiiert). Dabei wird das Objekt von anderen Objekten derselben Klasse unterscheidbar. Der Konstruktor ist eine Funktion (im Block methods, mit dem codewort function) welche den gleichen Namen hat wie die Klasse (d. h. das M-File ohne den Zusatz .m).
- Durch **Vererbung (definition as subclass)** wird eine Klasse als Unterklasse einer anderen (base class oder superclass) definiert. Damit müssen nur noch jene internen Daten und Methoden programmiert werden, die anders sind als in der übergeordneten Klasse. So wird meist eine ganze Baumstruktur von Klassen und Unterklassen definiert.

Als Beispiel für dieses Programmier-Prinzip habe ich eine Computerblume gewählt. Da braucht es keine Vorkenntnisse um die gewählten Eigenschaften und Funktionsweisen zu verstehen, und man kann sich ganz auf die Konventionen des Programmierens konzentrieren.

Die Klasse `ooblume` hat die internen Daten Liniendicke, Wurzelposition (xpos, ypos), Stiellänge (stiellg), Farbe, Länge und Anzahl der Blütenblätter (farbe, petallg, numpetal), sowie den halben Öffnungswinkel des Blütenblätter-Kranzes (petalhw). Die meisten Grundwerte werden gerade bei der Deklaration im `properties`-Block definiert.

Die internen Daten linhdls sind die gesammelten Grafik-handles der beim Plotten erzeugten Linien. Diese werden benötigt, um die Grafik löschen (radieren) zu können, so dass das Objekt weiterlebt und später neu gezeichnet werden kann.

```
classdef ooblume < handle
    % Grafik einer Blume in objektorientierter (OO) Syntax
    %   Konstruktor: blobjnam = ooblume(xpos, ypos, blgroesse)
    %   Die meisten Parameter sind als Default vorgegeben, mit
    %   blobjnam.propnam = ... koennen sie aber veraendert werden
    properties    % allgemein zugaengliche interne Daten
        farbe = 'y';
        liniendicke = 3;
        xpos = 0;   ypos = 0;
```

```
        stiellg = 4; petallg = 1;
        numpetal = 21; petalhw = 10;
    end
    properties (SetAccess = private, GetAccess = private)
        linhdls;   % abgeschirmte interne Daten
    end
    methods
        function blobj = ooblume(xposi,yposi,farbi)
          blobj.xpos  =  xposi;
          blobj.ypos = yposi;
          if nargin >2 blobj.farbe = farbi; end
        end
        function  zeichne(blobj)
           [petalx, petaly] = makepetal(blobj);
           try   blobj.radiere;  end  % loeschen falls vorhanden
           blobj.linhdls = [];
           stielhd = plot([blobj.xpos blobj.xpos],...
               [blobj.ypos blobj.ypos+blobj.stiellg],'g',...
                 'linewidth',blobj.liniendicke);
           blobj.linhdls = [blobj.linhdls ; stielhd];
%
           blthd=plot(petalx, petaly,'color',blobj.farbe,...
                  'linewidth',blobj.liniendicke);
           blobj.linhdls = [blobj.linhdls ; blthd];
        end    % end function zeichne
        function radiere(blobj)
            delete(blobj.linhdls)
        end
        function delete(blobj)
            radiere(blobj)
        end
    end   % end methods
end   % end class ooblume
function [petx,pety] = makepetal(blobj) % rein interne Funktion
  for k = 1:blobj.numpetal
    dw = 2*blobj.petalhw/(blobj.numpetal-1);
    w = 90-blobj.petalhw + dw*(k-1);
    petx(:,k) = [0 ; cosd(w)*blobj.petallg];
    pety(:,k) = [0 ; sind(w)*blobj.petallg];
  end
  petx = petx+blobj.xpos;
  pety = pety+blobj.ypos + blobj.stiellg;
end  % function makepetal
```

Nachdem ein ooblumen-Objekt mit dem Konstruktor-Aufruf erzeugt wurde, kann man seinen Properties (internen Parametern) neue Werte zuweisen. Dazu verwendet man die Formulierung Objektname.Propertyname = Neuwert.

Die Angabe von Name.Subname wird auch bei Strukturvariablen angewandt.

Anwendungs-Beispiel Bl1 ist eine rote Standardblume, Bl2 hat separate `[r,g,b]` Farbwahl (orange), einen halben Öffnungswinkel von 80° und nur 0,4 cm lange

Blütenblätter. Bl3, Standard: gelb hat einen kürzeren Stiel von 3 cm und petalhw = 180° ist also kreisrund.

```
figure(1); clf;
axis([-7 7 -1 8]) ; hold on; axis equal; axis off
Bl1 = ooblume(4,0,'r');  Bl1.zeichne ; % Bl1 zeichnen
Bl2 = ooblume(0,1); Bl2.petalhw = 80; Bl2.petallg = 0.4;
Bl2.farbe = [1 0.6 0] ; Bl2.zeichne;   % Bl2 zeichnen
Bl3 = ooblume(-1,0); Bl3.petalhw = 180; Bl3.stiellg = 3;
Bl3.zeichne ; pause(1.8)               % Bl3 zeichnen
Bl1.petalhw = 100;  Bl1.zeichne; pause(1.8) % Bl1 modifiziert
Wbl = dynablume(-4,0.2,'m'); Wbl.petwmax = 180; Wbl.wachsstart;
Vbl = dynablume(1.5,0.1); Vbl.petwmax = 90; Vbl.wachsstart;
```

Wbl und Vbl sind Objekte aus einer neuen Klasse `dynablume`. Als Unterklasse von `ooblume` erbt sie alle properties (Daten) und Methoden (Funktionen) der übergeordneten Klasse. Diese animierten Grafiken brauchen als Zusatz-Daten `wzeit, oeffzeit` und `petwmax`, sowie eine neue Methode `wachsstart`.

```
classdef dynablume < handle & ooblume
    %  Unterklasse von ooblume
    %  waechst und oeffnet sich dynamisch
    properties
       wzeit = 2.5; oeffzeit = 4; petwmax = 120;
    end
    methods
        function dblo = dynablume(xposi,yposi,farbi)
          if nargin < 3  farbj = 'y'; else  farbj = farbi; end
          dblo@ooblume(xposi,yposi,farbj);
        end
        function wachsstart(dblo)
            dblo.petalhw = dblo.petwmax*0.1;
            for k=0:10  % Stiel waechst zuerst
              dblo.stiellg = 2+k*0.2;
              dblo.zeichne;  pause(0.1*dblo.wzeit)
            end
            for k=0:10  % dann oeffnet sich die Bluete
              dblo.petalhw = dblo.petwmax*(0.1+0.09*k);
              dblo.zeichne;  pause(0.1*dblo.oeffzeit)
            end
        end
    end
end
```

Mit diesem Beispiel konnte die Welt der Objektorientierten Programmierung in MATLAB nur ganz am Rande gestreift werden. Puristen der OO-Programmierung werfen MATLAB vor, dass es einige Optionen gar nicht anbietet, wie z. B. Signatur-Overloading (anhand der Eingabeparameter-Liste erfolgt die Auswahl der auszuführenden Funktion). Aber auch von den in MATLAB vorhandenen Optionen blieben viele unerwähnt. Einen detaillierten Überblick liefert die Dokumentation in „Comparing MATLAB with other OO Languages“.

Das Ziel dieses Abschnittes ist auch nicht Ihre Ausbildung zum OO-Programmierer. Das Kennen der einfachsten Grundlagen ermöglicht es bereits, komplexen Aufgaben eine geeignete Struktur zu geben, durch Bündelung von Daten und zugehörigen Funktionen. Zugleich werden die zum Erzeugen von anspruchsvollen Grafiken nötigen Aufrufe leichter verständlich. Die „handle graphics" genannten Grafikfunktionen von MATLAB sind vollständig in Objektorientierter Programmierung realisiert (siehe Abschn. 1.5).

Detail-Information über Objekte und deren Klassendefinition liefern die folgenden Befehls-Sequenzen:

- Die Eingabe des Objektnamens (im obigen Beispiel `Wbl`) liefert die Anzeige aller zum externen Lesen offenen Properties mitsamt deren aktuellen Werten. Der unterstrichene Klassenname ergibt einen Link zur Klassendefinition.
- `properties(Objname)` bzw. `properties('Klassenname')` liefern die zugänglichen internen Daten und `methods(Objnam)` bzw. `methods('Klassenname')` liefern die aufrufbaren Funktionen. (Beachten Sie die Apostroph-Einfügung, welche nur bei Angabe einer Klasse nötig ist.)

Merkpunkte: Grundprinzipien für OO-Programme in MATLAB

- Die Daten (properties) und Funktionen (methods) von Objekten werden im classdef M-File festgelegt. (Das classdef-File entspricht einer Gussform, mit der beliebig viele Objekte derselben Art erzeugt werden können.)
- Der Objekt-Name, der Rückgabeparameter beim Konstruktor-Aufruf, ist ein „handle" (Handgriff, eine Adressmarke) bei allen Klassen, welche die handle-Klasse als Überklasse angeben (das sind fast alle!).
- Properties ohne Access-Vermerk sind offen und können mit der Syntax `Objname.Propname` angezeigt und mit `Objname.Propname = Wert` modifiziert werden.
- Aufrufe von Methoden (Funktionen) eines Objektes erfolgen mit der Syntax `Objname.Fktname(Aufrufparameter)`.

1.5 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.

1.5.1 Funktionsdarstellungen

In MATLAB baut eine Darstellung von Funktionen grundsätzlich auf **Tabellen** auf. Zwischen den Tabellenpunkten werden Geradenstücke gezeichnet; die Tabellierung muss daher fein genug gewählt werden. 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.

Von einer analytisch definierten Funktion im symbolischen Modus erhält man eine Grafik mit der Funktion `ezplot(fctdef)`. Die Definition kann auch direkt als Zeichenkette eingegeben werden: `ezplot('x^2')`. Dabei erfolgt die Auswahl des Darstellungs-Bereiches und die Tabellierung automatisch.

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'; % Spaltenvektor durch Transponieren
plot(xv,yv)
```

sollte es genauso funktionieren.

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

```
plot(y)
```

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 Pluszeichen 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])    ( allg. axis([xmin,xmax,ymin,ymax]) )
```

nach dem `plot()` Aufruf) 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:

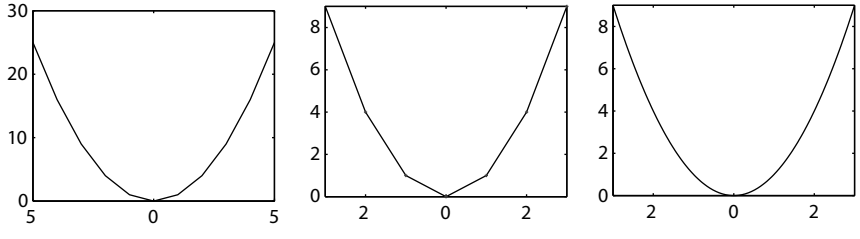

Abb. 1.8 Parabelplots mit automatischer und vorgegebener Wahl des Anzeigebereiches, mit grober und feiner Tabellierung.

Demonstration des spaltenweisen Zeichnens

Wenn die drei Funktionen $z1 = x$, $z2 = x^2$ und $z3 = x^3$ als Zeilenvektoren, übereinander zur Matrix $\boldsymbol{P}$ zusammengefügt wurden, so ergibt der Aufruf `plot(P)` nicht die Darstellung der drei Funktionen. Erst der Transposition von $\boldsymbol{P}$, also mit `plot(P')` erscheinen die Funktionen je in einem Spaltenvektor

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

Eine ausführliche Variante, mit der einzelne Zeilen statt Spalten aus einer Matrix ausgewählt und geplottet werden können, bietet die Angabe eines Index-Bereiches: `P(2,:)` bedeutet z. B. 2. Zeile von $\boldsymbol{P}$, alle Spalten, also den 2. Zeilenvektor aus $\boldsymbol{P}$.

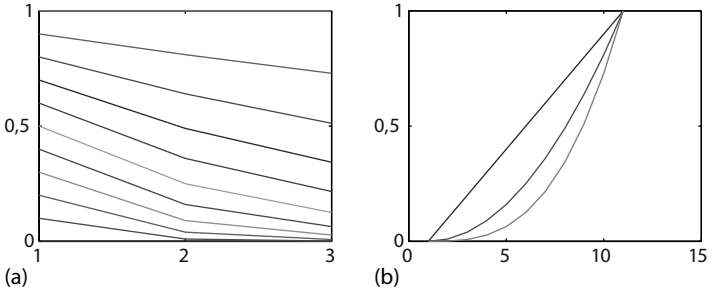

Abb. 1.9 Demonstration der Konvention, dass beim Zeichnen der Werte einer Matrix die Spalten als Funktionsvektoren gelten. (a) `plot(P)`, (b) `plot(P')`.

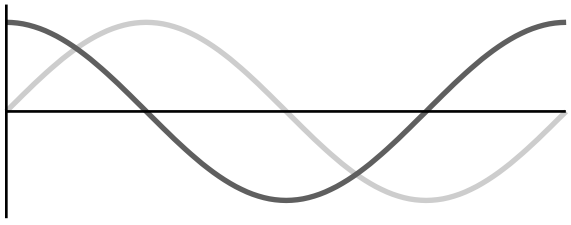

Abb. 1.10 Kosinus- und Sinusfunktion über eine Periode der Länge 2π.

Merkpunkte: Einfache Linienplots

- Bei **einem Datenparameter** wählt `plot()` die Elementnummer als Abszisse. Bei **zwei Datenparametern** sind die Kombinationen Vektor-Vektor, Vektor-Matrix und Matrix-Matrix möglich.
- Bei **Matrizen-Eingabe** in `plot()` wird jeder Spaltenvektor als separate zu zeichnende Linie behandelt.
- Der **Zeichenkettenparameter** (string) ist eine Kombination von Farb-Buchstabe, Linientyp und Punkt-Marker.
- Der **Wertebereich des Plotfensters** wird mit `axis([xmin, xmax, ymin, ymax])` festgelegt. Gleiche Längen in x und y erreicht man mit `axis equal`.
- **Weitere Linien in dieselbe Grafik** erreicht man durch `hold on` (beenden mit `hold off`).

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 (rad) definiert. Die meisten Leser sind aber viel besser vertraut mit der Gradeinteilung. Die trigono-

metrischen Funktionen für Argumente in Grad erhalten alle ein „d“ angehängt, für „degree“, also `sind()`, `cosd()` etc.

```
w = (0:0.01:1)*2*pi;   si = sin(w);  co = cos(w);
plot(w,si);   hold on;   plot(w,co,'r');  hold off
```

1.5.2 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:

```
wg = 0:5:360;   si = sind(wg);   co = cosd(wg);
plot(co,si)
```

Dabei ergibt sich leider nur eine Ellipse. Die nötige Anpassung der Achsenskalierung kann, (nach dem Aufruf von `plot()`) auf zwei Arten erfolgen: mit

```
axis equal    oder mit dem Befehlspaar
axis([-1.2 1.2 -1.2 1.2]) ;   axis square
```

Wem an diesem Kreis doch noch zu stark die Ecken sichtbar sind, der kann das-

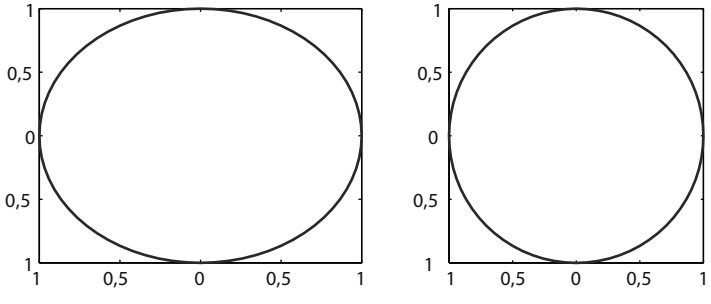

Abb. 1.11 Erst nach Anpassung der Achsenskalierung erscheint ein Kreis.

selbe mit einer feineren Winkeleinteilung versuchen, z. B. mit `w = 0:1:360` und anschließend dieselben Befehle verwenden wie oben.

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 und Sterne

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`.

Die Anzahl Ecken muss vorher definiert werden: so z. B. `neck = 4`.

Daraus ergibt sich der Winkelvorschub und damit die Tabellen der Winkel, der x- und der y-Koordinaten für die zu zeichnenden Punkte:

```
wg = 0:360/neck:360 ;
x = cosd(wg) ;  y = sind(wg) ;
plot(x, y)
```

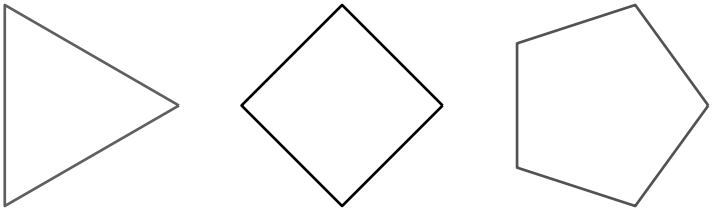

Abb. 1.12 Durch Kosinus- und Sinustabellen gezeichnete Polygone.

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.

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 benutzt man den Zusammenhang zwischen Umkreisradius und Seitenlänge: Das gleichschenklige Dreieck über der Seite wird in zwei rechtwinklige geteilt: $s/2 = r * \cos(2\pi/2n)$

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°):

```
w = (0:2*72:2*360) + 90 ;
si = sind(w) ;   co = cosd(w);
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 ;   w = (0:2*360/neck:360) + 90 ;
xa = cosd(w) ;    ya = sind(w);
xb = cosd(w+360/neck) ;  yb = sind(w+360/neck);
plot(xa,ya,'k', xb, yb,'k')
```

1.5.3 Flächen malen

Mit dem Aufruf `fill(xlin,ylin,farbdef)` oder `patch(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. Die Möglichkeiten zur Definition der Farbe werden anschließend erklärt.

Ein kurzer Test, wie das Pentagramm als auszumalende Figur aussieht (mit `fill(co,si,'g')` statt `plot()`)

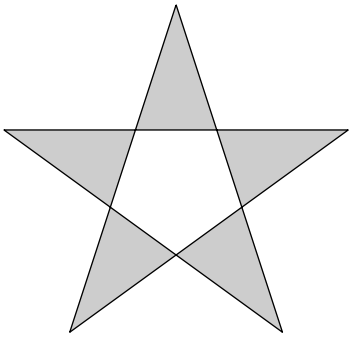

Abb. 1.13 Pentagramm, übergeben an die Funktion `fill()`.

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.

Farbdefinitionen

Leitbuchstabe Die einfachste Farbdefinition ist die Auswahl der reinen Farben durch ihren charakteristischen Buchstaben. Diese sind „r“, „g“, „b“, „y“, „c“, „m“ für rot, grün, blau, gelb, cyan, magenta, sowie „w“ und „k“ für weiß und schwarz (blac„k“).

Farbindex als Verweis in eine colormap Für Grafiken mit vielen Farben, meist mit Farb-Übergängen, verwendet MATLAB eine aktuelle „colormap“. Einige von deren Namen sind „parula“ (vordefiniert), „jet“, „hsv“, „hot“, „cool“, „summer“, „copper“. Das sind Tabellen, häufig mit 64 Zeilen und mit drei Spalten. Jede Zeile enthält die Definition einer Farbe, das sind drei Zahlen zwischen 0 und 1, welche die Intensität der Grundfarben Rot, Grün und Blau beschreiben.

Die Farbe jedes Objektes in der Grafik wird durch dessen Farbindex bestimmt, der die Zeilennummer in der aktuellen colormap angibt. Durch einfaches Auswechseln der colormap mit dem Befehl `colormap(name)` kann dann das Erscheinungsbild einer Grafik mit Farb-Übergängen modifiziert werden. Weitere Details erhalten Sie mit dem Befehl `doc colormap`.

Das Prinzip der r,g,b-Farbdefinition Durch die Angabe von drei Zahlen zwischen 0 und 1 (in gewissen Computern auch zwischen 0 und 255) können alle Farben definiert werden. Ausgenommen sind dabei Transparenz- und Glanzeffekte.

Weil die Definition einer Farbe durch die hsv-Angabe (hue, saturation, value, Farbwinkel, Sättigung, Helligkeit) leichter verständlich ist, wird hier der Zusammenhang zwischen den zwei Systemen beschrieben.

Die zugehörigen Farbwinkel zu den reinen Farben und deren rgb-Werte sind rot 0° `[1 0 0]`, gelb 60° `[1 1 0]` , grün 120° `[0 1 0]`, cyan 180° `[0 1 1]`, blau 240° `[0 0 1]`, und magenta 300° `[1 0 1]`. Für die Zwischenwerte des Farbwinkels wird die obige Tabelle linear interpoliert. Orange z. B. liegt zwischen rot und gelb, hat also die rgb-Werte `[1 0.5 0]`.

Um Graustufen-Anteile (Sättigung kleiner als Eins) zu erzeugen erhalten die rgb-Werte einen Weiß-Sockel vom Wert `(1-s)`, der alle drei Grundfarben in der gleichen Stärke enthält. Dafür werden die durch den Farbwinkel bestimmten rgb-Werte mit dem Faktor `s` reduziert.

Am Schluss werden alle rgb-Werte mit dem Helligkeits-Faktor multipliziert.

Zum tieferen Verständnis dieser Definitionen empfehle ich die genaue Betrachtung der untenstehenden Funktion, die mit der Bibliotheksfunktion `hsv2rgb` abgeglichen wurde (Unterschied: `h` von 0° bis 360° Grad statt 0 bis 1).

```
function [ r,g,b ] = hsvtorgb( h,s,v )
% function [ r,g,b ] = hsvtorgb( h,s,v ) h=0..360, s,v= 0..1
% r,g,b aus Farbwinkel h(Grad), Saettigung s und Helligkeit v
rp=[1 1 0 0 0 1 1]; gp=[0 1 1 1 0 0 0]; bp=[0 0 0 1 1 1 0];
% hi ganzzahlig  1 bis 6, f ist Zwischenwert-Bruchteil
hreal=h/60; hint=floor(hreal); f=hreal-hint; hi=mod(hint,6) + 1;
% mit h interpolierte rgb Basiswerte
  rb = rp(hi).*(1-f) + rp(hi+1).*f;
  gb = gp(hi).*(1-f) + gp(hi+1).*f;
  bb = bp(hi).*(1-f) + bp(hi+1).*f;
%  Saettigung ergibt Weiss-Sockel vom Wert (1-s)
  rs = s.*rb + (1-s);  gs = s.*gb + (1-s);  bs = s.*bb + (1-s);
% Helligkeit ergibt allgemeine Reduktion
  r = rs.*v; g = gs.*v; b = bs.*v;
end
```

Merkpunkt: Farbdefinition

- Die drei Arten der Farbdefinition sind **Buchstabe der Grundfarbe, Index in eine Farbtabelle** oder ein **rbg-Tripel** von Zahlen zwischen 0 und 1.

1.5.4 Properties von grafischen Objekten

Die Bestandteile einer Grafik in MATLAB sind ineinander verschachtelte Objekte der Klasse Handle-Graphics. Die Grundlage bildet das Objekt „figure“, worauf die zugehörigen „axes“-Objekte aufbauen. (Der Objektname „axes“ darf nicht mit der Funktion axis() verwechselt werden, welche diverse Parameter in „axes“ modifiziert.) Beim plot-Aufruf wird ein Handle als Rückgabeparameter geliefert. Dieses dient z. B. zum selektiven Löschen einer Linie.

Die Möglichkeiten zum Anpassen einer Grafik mit Hilfe der Handle-Struktur sind vielfältig:

- Die Handles von „figure“ und „axes“ können mit den Funktionen `gcf` (get current figure), `gca` (get current axes) abgefragt werden. Nach dem Anklicken einer Teilgrafik sind Handles von untergeordneten Teilen mit `gco` (get current object) abfragbar.
  Die Funktion `get()` (z. B. `get(gca)`) zeigt die ganze Fülle der zugehörigen Properties. Um eine Eigenschaft zu modifizieren dient dann die Konstruktion `set(ghdl,'propname',Wert)`. (Beispiele: `set(gcf,'color','w')` oder `set(gca,'FontSize',16)`

Property	Werte	Bedeutung
Figure-Properties	get/set(gcf,...)	
Color	`[r g b]`-Werte	Figur-Hintergrund
Position	[xlo ylo xwid ywid]	Position und Dimension
Units	pixels	Einheiten von „Position“
Axes-Properties	get/set(gca,...)	
Color	`[r g b]`-Werte	Achsenflächen-Hintergrund
FontName	z. B. Helvetica	Buchstabenformen
FontSize	10, besser 16	Schriftgröße der Texte
FontUnits	points	Maßeinheit für Größenangaben
FontWeight	normal, bold	Schrift normal/fett
LineWidth	0,5 bis 2	Achsenstrichdicke
DataAspectRatio	`[dx/pix dy/pix dz/pix]`	Achsenskalierung
nextPlot	add replace	hold on/hold off Effekt

- Für die interaktive Modifikation von Grafik-Details gibt es den Property Editor der mit dem Befehl `propedit` oder der Schaltfläche „edit“ auf dem Grafikfenster gestartet wird. Ein ganz großes Plus dieser interaktiven Anpassung ist deren Umwandlung in eine Programmsequenz. Falls weitere Grafiken in analoger Art angepasst werden sollen, so muss die Serie von Klicks und Daten-Eingaben nicht mehr wiederholt werden.

Die Namen der Properties müssen im set-Aufruf in Apostroph eingeschlossen sein, die Groß-Kleinschreibung ist aber in diesem Fall unwichtig. Eine ganz kleine Auswahl aus der Fülle dieser Properties sind in vorstehender Tabelle beschrieben.

1.6 Übersicht über die wichtigsten Grundbefehle in MATLAB

1.6.1 In MATLAB definierte Operatoren und Grundbefehle

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

Eingabe-Konventionen

- Die Eingabe eines Kommandos gilt am Zeilenende als abgeschlossen, wenn nicht eine Markierung mit drei Punkten ... am Zeilenende die **Fortsetzung** verlangt.
- Nach jedem Berechnungsschritt werden die momentanen Zwischenresultate auf dem Bildschirm ausgegeben, außer man **unterdrückt die Ausgabe** mit einem **Semikolon (also mit ;) am Zeilenende**.
- Mehrere Befehle in derselben Zeile werden mit Komma oder Semikolon getrennt. Auch hier bewirkt das Semikolon die Unterdrückung der Bildschirm-Ausgabe.
- **Kommentare** können hinter einem **Prozentzeichen %** eingefügt werden. Kommentare hinter doppelten Prozentzeichen gelten als Titel der damit markierten Abschnitte.
- Vorherige Befehle können mit der Taste „Pfeil aufwärts“ wieder herangeholt werden. Dann ist mit den rechts-links Pfeiltasten ein „line edit“ Modus aktiviert, mit dem die Zeile modifiziert werden kann. Mit „return“ wird der Befehl anschließend ausgeführt.
  Im rechten unteren Teilfenster „Command History“ können vorherige Befehle durch Doppelklick aktiviert oder ganze Blöcke selektiert und in andere Fenster kopiert werden.

Variablennamen

- Bei der Form `'Name' = 'Berechnung'` wird das Resultat unter „Name“ abgespeichert, ohne Zuweisungs-Operation unter der einzigen Variablen `ans`.
- **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. Ebensowenig sind Leerschläge erlaubt.
  Die maximale Länge für einen Variablennamen beträgt 31 Zeichen.
- Im Berechnungs-Modus (Normalfall) werden Variablen bei ihrer ersten Zuweisung automatisch deklariert. Für das symbolische Rechnen ist eine Deklaration der verwendeten Variablen in der Form `syms r s t` (Abstand, kein Komma) notwendig.
- Für große Matrizen, in denen bei der Berechnung auf einzelne Elemente zugegriffen wird, empfiehlt sich die vorgängige Deklaration der zu erwartenden Maximaldimension in der Art von: `M = zeros(zmax,spmax)`.
- Eine Kontrolle der Dimensionen der bisher verwendeten Variablen ist mit dem Befehl `whos` (who-size) möglich; `who` zeigt die Liste der benutzten Variablen. Dieselbe Information liefert das rechte obere Teilfenster „Workspace“.
- 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 (d. h. Multiplikation mit der Inversen) von rechts steht.

$\boldsymbol{A}/\boldsymbol{B}$ bezeichnet die Operation $\boldsymbol{A} * \boldsymbol{B}^{-1}$.

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: $\boldsymbol{A}\backslash\boldsymbol{B} = \boldsymbol{A}^{-1} * \boldsymbol{B}$.

Zusätzlich zu den Elementar-Operationen gibt es auch den **Potenzierungsoperator** $\hat{}$.

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.

Tabelle der arithmetischen Operatoren

Operator	Elementweise Operation	Matrix-Operation
Addition, Subtraktion	`+ -`	`+ -`
Multiplikation, Division	`.* ./`	`* /`
Linksdivision	`.\`	`\`
Potenzieren	`.^`	`^`

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 $\boldsymbol{A}^{\mathrm{T}}$ für die zu $\boldsymbol{A}$ 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. Der Apostroph wirkt auf eine Matrix, und nicht auf eine Funktion.

Achtung! Genau genommen bedeutet in MATLAB der Apostroph „transponieren und gleichzeitig komplex konjugieren“ (Fachausdruck: Adjungieren), was für reelle Matrizen identisch ist mit Transponieren. Deshalb wird in diesem Kurs normalerweise nur der 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:

„<“, „<=“, „>“, „>=“, „==“, „~=“

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 V. Der Wert von 10 V wird als Flag benutzt, um anzuzeigen, dass für eine zuverlässige Messung zu wenig 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 „`~`“. (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.

1.6.2 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 $15{,}3 \cdot 10^{-8}$). 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`.

Die Eulersche Form $r \cdot e^{jw}$ wird als `r*exp(j*w)` eingegeben.

Die Zahlenausgabe wird mit dem `format`-Befehl vorgewählt und bleibt danach bis auf einen nächsten `format`-Befehl gültig.

`format short / short e / short g`	für 4 Mantissen,
`format long / long e / long g`	für 15 Mantissen und
`format bank`	für zwei Kommastellen, und noch:
`format compact / loose`	für engen/normalen Zeilenvorschub.

Ein spezielles Format, ist `format rat` für das Verfolgen von Berechnungen. (Darstellung möglichst 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 Positionen 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 der 3×3 Transformationsmatrix `T`, dem Translationsvektor `tr = [tx; ty; tz]` und der Basiszeile `b = [0 0 0 1]` eine 4×4-Matrix der homogenen Koordinatentransformation herstellen: `Thg = [[T tr] ; b ]`.

Eine andere häufige Anwendung dieser Möglichkeit ist das Nebeneinandersetzen von Spaltenvektoren von homogenen Koordinaten
`Mhv = [ v1 v2 v3 v4 v5 ]` z. B. mit `v1 = [1 2 5 1]'` etc.

Die Matrix der nebeneinandergestellten Vektoren `Mhv` kann dann wie ein einzelner Spaltenvektor durch Multiplikation von links beliebigen Transformationen unterworfen werden. Damit werden alle enthaltenen Vektoren simultan transformiert, wie in `Mthv = Thg*Mhv`.

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 $n \times n$. 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 ':'**, auch Aufzählungs-Operator genannt.

`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.

Einen analogen Vektor liefert die Funktion `linspace(start, ziel, npt)`, mit der Gesamtzahl der Punkte `npt`, d. h. `npt-1` Abschnitte.

1.6.3 Schleifen und Bedingungen

Mit MATLAB ist das Erstellen von komplexen Programmabläufen 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! Das gewohnte `1,n` ist auch erlaubt, aber wirkt anders.

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 (logisch „false“) 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.

Achtung! Das Schlüsselwort `elseif` muss ohne Leerzeichen geschrieben werden, sonst gelten `else` und `if` als separate Schlüsselwörter, was eine andere Struktur der end-Marken bedingt!

1.6.4 Mathematische Funktionen

Trigonometrische Funktionen

MATLAB kennt die gängigen Funktionen aus der Familie der trigonometrischen Funktionen wie: `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`, sowie `sind()`, `cosd()`, `tand()`, `asind()`, `acosd()`, `atand()`, für Winkel in Grad. Dabei können die Argumente immer auch Vektoren oder Matrizen sein. Bei nicht skalaren Argumenten hat das Resultat dieselbe Struktur wie der eingegebene Parameter.

Die Funktionen `atan2(y,x)` und `atan2d(y,x)` brauchen 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 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 $R^{(1/n)}$
`log(x)`, `log10(x)`, `log2(x)`
`sign()`
`exp(z)` für e^z, die Exponentialfunktion. Diese ist auch für komplexe z wichtig, in der Form `exp(-i*w)` für e^{-iw}.

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.

1.6.5 Grundfunktionen im symbolischen Modus

Den Variablen im Berechnungs-Modus muss man vor ihrer Verwendung einen Wert zuweisen. Die **symbolischen Variablen** müssen vor ihrer Verwendung **deklariert** werden. z. B. mit `syms y x z r v0 a` (zur Trennung nur Leerschläge, keine Kommata).

Anschließend kann man mit Formulierungen der Art

```
radius = sqrt(x^2+y^2+z^2)
```

beliebig komplizierte Funktionen definieren. So definierte Funktionen gelten automatisch als symbolisch, sobald sie mindestens eine symbolische Variable enthalten (mit `whos` kontrollierbar).

Für das Manipulieren von symbolischen Funktionen und von deren Kombinationen und Verschachtelungen gibt es eine Vielzahl von Hilfsfunktionen wie z. B.:

Funktion	Bedeutung
fneu = collect(forig)	Gleiche Potenzen sammeln
fneu = expand(forig)	Ausmultiplizieren
fneu = factor(forig)	Ausklammern
polyneu = horner(polyorig)	Polynom in Horner-Form
fneu = simplify(forig)	Sucht möglichst kurze Formel
fnserie = simple(forig)	Viele verschiedene Methoden
simplify(forig,ntry)	für kurze Formel durchprobieren
fneu = subs(forig, varalt, varneu)	Ersetzen varalt durch varneu
fwert = subs(forig, var, Zahl)	Einsetzen Zahlenwert in Formel

Zum Umstellen einer Formel (Auflösen nach einer internen Variablen) und allgemein zum Lösen von Gleichungen und Gleichungssystemen dient die Funktion `solve(eqn,var)`. Als Eingabe werden eine oder mehrere Gleichungen erwartet, sowie die freizustellende Variable, bzw. die Variablen nach denen das Gleichungssystem aufzulösen ist. Dazu zwei einfache Beispiele:

```
syms x y z r a b
rform = sqrt(x^2+y^2+z^2) % Punkt auf Kugel
eq1 = rr == rform % = definiert, == fuer Gleichung
xf = solve(eq1,x) % Formel für x aus r,y,z
% ergibt     xf =  (r^2 - y^2 - z^2)^(1/2)
% 2 Loesungen     -(r^2 - y^2 - z^2)^(1/2)
% Gleichungssystem, Gleichungen direkt eingegeben
sol = solve('y==a*x','y==-x+b',x,y); % 2 Geraden
solvec = [sol.x, sol.y] % Struktur mit sol.x, sol.y
% Resultat: solvec = [ b/(a + 1), (a*b)/(a + 1)]
```

1.6.6 „struct"- und „cell"-Variablen

Die „struct" Deklaration

Seit einigen Jahren hat MATLAB von anderen Programmiersprachen die Möglichkeit zum Erzeugen von **strukturierten Variablen** übernommen.

Mit dem Befehl `vnam=struct` bzw. `vnam=struct('subnam',subwerte)` wird die Variable mit dem Namen „vnam" als „struct"-Variable deklariert.

Dann können mit einem Punkt dazwischen beliebige Sub-Variablen angehängt werden und deren Daten definiert werden. Als Inhalte der Sub-Variablen sind Zahlen, Vektoren, Matrizen, Zeichenketten, „cell"-Arrays und ganze Kaskaden von weiteren Strukturvariablen möglich. Als Beispiel ein Achsenkreuz-Dreibein:

```
% 3D Achsenkreuz-Dreibein als struct-Variable
drb = struct('name', 'Achsenkreuz-Dreibein');
drb.x = [1 0 0 0 0]; drb.y = [0 0 1 0 0]; drb.z = [0 0 0 0 1];
drb,  plot3(drb.x, drb.y, drb.z) % anzeigen, anwenden
```

Dieselbe Notation mit Hauptname.Subname erscheint in zwei Varianten in der objektorientierten Programmierung: Bei der Zuweisung Objekt.Property und beim Aufruf mit Objektname.Methode.

„cell" Arrays

Den Variablen-Typ **„cell" Array** stellt man sich am besten vor als ein Büchergestell mit nummerierten Kartonschachteln. Die Nummerierung (Indizierung) kann eindimensional wie bei einem Vektor oder zweidimensional wie bei einer Matrix angelegt sein. Über den Inhalt der einzelnen Zellen ist nichts festgelegt, er kann von Zelle zu Zelle variieren.

Die „cell" Arrays dienen zum Zusammenbinden von gemischten Informationen, die sowohl Text-Teile als auch Zahlen enthalten. Eine Tabelle kann z. B. bei der Import Data Funktion als 'cell' Array eingelesen werden. Bei Analysieren eines Textes wird oft jedes Wort in einer Zelle platziert. (Die Form einer Matrix von Buchstaben ist ungeeignet, weil die Wörter verschieden lang sind.)

Durch den Befehl z. B. `mixdat = cell(nzeile,nspalte)` wird eine **Variable als „cell" Array deklariert** und deren Dimension vorgemerkt. Für eindimensionale Indizierung unbedingt `cell(1,ndim)` verwenden, denn `cell(ndim)` bedeutet `cell(ndim,ndim)`!

Die direkte Eingabe von Daten in einen „cell“ Array mit simultaner Deklaration erfolgt durch Einschließen der Daten in geschweifte Klammern.

Es gibt zwei Arten auf eine einzelne Zelle zuzugreifen:

Mit **Indizes in runden Klammern** zielt man auf die entsprechende Zelle:
schreiben: `mixdat(zeilind,spaltind) = zellobj` (Index in runden Klammern), verlangt ein Objekt des Typs „cell“, bzw. eine in geschweifte Klammern eingefügte normale Variable.
lesen: `zelement = mixdat(zeilind,spaltind)` (Index in runden Klammern), definiert ein Objekt des Typs „cell“.

Mit **Indizes in geschweiften Klammern** meint man den Inhalt der Zelle:
schreiben: `mixdat{zeilind,spaltind} = zinhalt` (Index in geschweiften Klammern), platziert normale Variablen im „cell“ Array.
lesen: `zinhalt = mixdat{zeilind,spaltind}` (Index in geschweiften Klammern), extrahiert den Inhalt der Zelle und liefert die eingespeicherte Variable mit ihrem eigenen Typ (z. B. double oder string).

1.6.7 Grafische Darstellungen

Die Registerkarte PLOTS

Sobald man eine Variable angewählt hat, eröffnet sich in der Registerkarte PLOTS eine große Auswahl von Möglichkeiten, diese Größe grafisch darzustellen. Dies erlaubt, mit wenigen Klicks verschiedene grafische Darstellungen auszuprobieren. Alle interaktiven Aktionen werden dabei gespeichert, so dass der eingeschlagene Verarbeitungs-Weg später im Kommando-Modus wieder abspielbar ist.

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.

Linienplots und andere Grafiken analytisch definierter Funktionen im symbolic Modus können mit der Funktionsfamilie `ezplot`, `ezplot3` , `ezsurf` etc. erstellt werden. Auch diese verwenden für das Plotten tabellarische Daten, deren Feinheit allerdings automatisch bestimmt wird.

Die Datenparameter des plot()-Aufrufs

- `plot(yvec)` Ein einzelner Datenparameter-Vektor wird als Ordinate gezeichnet, die zugehörige Abszisse entspricht den Indexwerten.
- `plot(ymat)` Bei einer Matrix als Datenparameter wird für jeden Spaltenvektor eine separate Linie gezeichnet.

- `plot(xvec,yvec)` Im Standardfall mit zwei Parametern wird yvec als Funktion von xvec gezeichnet.
- `plot(xvec,ymat)` Mehrere Linien, entsprechend den Spaltenvektoren von ymat, können auf denselben Abszissenvektor bezogen werden.
- `plot(xmat,ymat)` Zwei analoge Matrizen ergeben mehrere Linien mit individuellen Abszissenwerten.

Der Zusatzparameter vom Typ Characterstring

Mit Aufrufen vom Typ `plot(v1,v2,'Stringparameter')`, also durch Einsetzen eines Zusatzparameters, der in Apostrophzeichen eingefügte Buchstaben enthält, können 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: ^, >, 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.

Für Linienplots mit geometrischer Bedeutung kann mit `axis equal` eine in x und y analoge Achsenskalierung erzwungen werden.

Jeder neue plot-Aufruf löscht im Normalfall die aktuelle Figur. Der Aufruf von `hold on` erlaubt das Weiterzeichnen in derselben Grafik, bis mit `hold off` diese Option gestoppt wird.

Mehrere Linienzüge können auch durch Eingabe von mehreren Paaren von $\boldsymbol{x}$- und $\boldsymbol{y}$-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 (Teilbilder) 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 Bildbe-

standteile 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 Farbe wird mit dem „c“ (color) aktiviert.

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` Windows-Metafile in Zwischenablage einfügen.
`print -dbitmap` File im Bitmap-Format in die Zwischenablage einfügen.

1.7 MATLAB Grundlagen aktivieren

Checkliste zu Kapitel 1

Die Arbeit mit diesem Kapitel soll Ihnen die Grundkenntnisse für das Einsetzen von MATLAB bei einfachen Problemen vermitteln. Im Überblick umfasst dies die folgenden Kenntnisse und Fähigkeiten:

- Beliebige Berechnungen mit Zahlen in MATLAB eingeben.
- Einfache Formel-Umwandlungen im Symbolischen Modus von MATLAB vornehmen.
- 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.
- Die wichtigsten Ein- und Ausgabe-Methoden zum Datentransfer mit Dateien anwenden.
- Einfache Skript- und Funktions-M-Files erstellen und anwenden und das Grundprinzip von internen Daten (Properties) in MATLAB-Objekten verstehen.

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 1 km gerade einem „c“, also einem Hundertstel eines Neugrades auf dem Äquator entspricht. Ein „cc“ ist dann also 10 m. (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 $5{,}97 \cdot 10^{24}$ 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. Da-

mals 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 $1{,}2\,\mathrm{kg/m^3}$ angenommen werden.

10–4 Physikalische Arbeitsleistung eines Bergwanderers
Bei Bergwanderungen wird als Richtwert 1 h für die Überwindung einer Höhendifferenz von 300 m eingesetzt. Welche durchschnittliche Leistung in Watt erbringt ein 75 kg schwerer Wanderer für den reinen Höhengewinn? Die Erdbeschleunigung beträgt $9{,}81\,\mathrm{m/s^2}$.

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 L Diesel-Treibstoff pro 100 km verbraucht. Für die volle Ladung kann man einen Wert von ungefähr 20 000 L 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 g ausgehen, das spezifische Gewicht der meisten Speisen ist nahezu $1\,\mathrm{g/cm^3}$ 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. $a^3 - 3 * a^2 * b + 3 * a * b^2 - b^3$.
Geben Sie diese Formeln im symbolischen Modus ein und testen Sie die Eingaben durch Anwenden der Funktion factor(formel).

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 $5050 = 101 \cdot 50$. Die allgemeine Formel für die Summe einer Reihe natürlicher Zahlen von a bis b lautet

$$s = \frac{1}{2} \cdot (a + b) \cdot (b - a + 1) \, .$$

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: $a_k = a_1 + (k-1) \cdot d$; $k = 1, \ldots, n$.
Die zugehörige Summenformel lautet:

$$s = \frac{1}{2} \cdot n \cdot (a_1 + a_n) = \frac{1}{2} \cdot n \cdot [2 \cdot a_1 + (n-1) * d] \, .$$

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 $n \times n$ 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 2×2-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 1×12, 2×6, 3×4, 4×3, 6×2, 12×1. 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 8×8-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 6×6 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 7×7-Matrix zu erhalten, welche dann die Einsen in der oberen, bzw. unteren Nebendiagonalen aufweist!

10–17 Würfelkoordinaten

Erstellen Sie eine 3×8-Matrix $\boldsymbol{W}$ durch Nebeneinanderstellen der Eck-Koordinaten eines Würfels ABCD EFGH mit `A=[0 0 0]'`, `B=[1 0 0]'`, `C=[1 1 0]'`, und `E=[0 0 1]'` etc. Setzen Sie die 3×16-Matrix $\boldsymbol{Wl}$ mit der Abfolge ABCDA E FBF GCG HDH E in `plot3(Wl(1,:),Wl(2,:),Wl(3,:))` ein!

10–18 Einmaleins-Tabelle
Erzeugen Sie eine Einmaleins-Tabelle durch untereinander Anordnen von Zeilenvektoren mit impliziter Schleife der Art `k:k:10*k`, mit verschiedenen *k*-Werten!

10–19 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

10–20 Welche der folgenden Aussagen sind wahr?

- Eine symmetrische Matrix ist quadratisch.
- Eine antisymmetrische Matrix, addiert zur 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.

10–21 Zerlegung in symmetrischen und antisymmetrischen Anteil
Berechnen Sie aus der Matrix $\boldsymbol{T}$ den Ausdruck $\boldsymbol{T} - \boldsymbol{T}'$! Suchen Sie eine Methode, um den antisymmetrischen A und den symmetrischen Anteil S der Matrix $\boldsymbol{T}$ zu bestimmen, so dass gilt $A + S = \boldsymbol{T}$!

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

10–22 Charakteristische Eigenschaften von speziellen Matrizen
Finden Sie je eine 3×3-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

10–23 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 $n \times m$ Rechtecksmatrix an.
Erzeugen Sie das Komplement zur Einheitsmatrix mit Nullen auf der Diagonalen und Einsen an den anderen Plätzen.

10–24 diag()
Sehen Sie in der Hilfefunktion nach, wie `diag()` anzuwenden ist. Erzeugen Sie anschließend eine Diagonalmatrix der Dimension $n \times n$ z. B. $n = 15$ mit den fortlaufenden ungeraden Zahlen 1, 3, 5 etc. in der Diagonalen.

10–25 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.

10–26 Quadrieren von magischen Quadraten
Lernen Sie die MATLAB-Demo über magische Quadrate (Demo – Matrizen – Matrix-Manipulationen) kennen! Nehmen Sie daraus ein magisches Quadrat 3×3 oder 4×4 und probieren Sie das folgende Rezept aus: Ein magisches Quadrat der Dimension $n \times n$ kann man wie folgt quadrieren: In die $n \times n$-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

10–27 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.

10–28 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.

Übungen zum Programmieren von Schleifen

10–29 Indexwert-Dreiecksmatrix
Füllen Sie eine obere Dreiecksmatrix mit Zahlen welche gerade das eigene Indexpaar als zweistellige Zahl anzeigen (dies funktioniert nur für $n < 10$), also z. B. $a_{11} = 11$, $a_{34} = 34$, etc.

10–30 Füllen einer Dreiecksmatrix
Erstellen Sie ein M-File mit einer Doppelschleife, welches eine obere (rechte) Dreiecksmatrix mit 1 füllt für die allgemein vorgegebene Dimension „n".

10–31 Tridiagonalmatrix
Erstellen Sie ein M-File, das eine Tridiagonalmatrix mit den Werten 2 auf der Diagonalen und den Werten −1 auf den beiden Nebendiagonalen erzeugt.

10–32 Streifenmuster
Erstellen Sie ein MATLAB-Skript, welches eine Matrix der Dimension n mit folgendem Muster füllt:
Auf der Diagonale Einsen, auf den Nebendiagonalen Nullen, anschließend auf den nachfolgenden zu der Diagonalen parallelen Linien wieder Einsen, dann wieder Nullen etc.!

10–33 Spezielle untere Dreiecksmatrix
Füllen Sie eine untere Dreiecksmatrix mit den Werten, welche der Summe der beiden Indizes entsprechen.

„Turm"- und „Specht"-Matrizen

10–34 „Spechtmatrizen" – Effekt der Matrixmultiplikation
Füllen Sie eine 4×4 Matrix $\boldsymbol{M}$ der Reihe nach mit den Zahlen 1 .. 16.
Erzeugen Sie eine andere 4×4 Matrix $\boldsymbol{S}$ mit lauter Nullen, außer einer einzigen Eins. Beobachten Sie den Effekt, den die Multiplikationen $\boldsymbol{S} * \boldsymbol{M}$ und $\boldsymbol{M} * \boldsymbol{S}$ produzieren für verschieden gewählte Positionen der Eins innerhalb von $\boldsymbol{S}$!

10–35 Spechtmatrizen – Zeilenselektion
Erstellen Sie als Testobjekt $\boldsymbol{T}$ eine 5×5 Indexwert-Matrix (mit zweistelligen Zahlen die den Indizes entsprechen, z. B. $a_{11} = 11$, $a_{34} = 34$).
Multiplizieren Sie diese mit der geeigneten Spechtmatrix von links her, mit dem Ziel, die erste Zeile von $\boldsymbol{T}$ in die 1., 2., 4. und 5. Zeile zu platzieren.
Bestimmen und testen Sie ebenso die notwendigen Spechtmatrizen für eine analoge Platzierung der 4. und der 5. Zeile von $\boldsymbol{T}$.

10–36 Spechtmatrizen – Spaltenselektion
Verwenden Sie wieder als Testobjekt $\boldsymbol{T}$ eine 5×5 Indexwert-Matrix. Multiplizieren Sie diese Matrix mit der geeigneten Spechtmatrix von rechts her, mit dem Ziel, die erste Spalte von $\boldsymbol{T}$ in die 1., 2., 4. und 5. Spalte der Resultatmatrix zu platzieren.
Bestimmen und testen Sie ebenso die notwendigen Spechtmatrizen für eine analoge Platzierung der 4. und der 5. Spalte von $\boldsymbol{T}$.

10–37 Durch Permutationsangabe definierte Turmmatrix
Erstellen Sie ein MATLAB-Programm (als Funktions-M-File), das einen Spaltenvektor mit den Zahlen von 1 bis n in beliebiger Reihenfolge als Eingabe benötigt, und daraus diejenige Turmmatrix produziert, welche die im Eingabevektor enthaltene Permutation erzeugt. Als Test kann die Multiplikation der Turmmatrix von links an einen Vektor mit der natürlichen Abfolge der Zahlen $1, 2, 3, \ldots, n$ dienen. Durch diese Multiplikation soll der vorgegebene Vektor erzeugt werden.

10–38 Scroll-up- und Scroll-down-Matrizen
Spezielle Turmmatrizen sind die Scroll-up- (bzw. Scroll-down)-Matrizen (fast alle Einsen direkt oberhalb/ unterhalb der Diagonalen). Bei der Multiplikation von links schieben Sie alle Zeilen der rechts stehenden Matrix (alle Werte eines rechts stehenden Vektors) um einen Platz nach oben (bzw. unten). Wird das am Rand herausfallende Element am anderen Rand eingefüllt, so nennt man die Scroll-up/down-Abbildung zyklisch. Fällt das Element am Rand weg, so sind die Matrizen keine eigentlichen Turmmatrizen mehr.
Erstellen Sie für die Dimension 5 alle 4 Typen: zyklische und nicht zyklische Scroll-up- und Scroll-down-Matrizen und testen Sie deren Wirkung an einem Spaltenvektor mit den Zahlen 1:5.
Überlegen Sie sich, welchen Rang diese 4 Matrizen haben und kontrollieren Sie Ihr Resultat mit der Funktion `rank()` von MATLAB.

10–39 Die Wirkung von Turmmatrizen
Testen Sie die Wirkung einiger Turmmatrizen (insbesondere der Scroll up/down Matrizen) beim Multiplizieren der Index-Anzeige-Matrix von links her und von rechts her.

10–40 Die Wirkung von Auswahl- und Permutationsmatrizen
Bestimmen Sie die Matrizen $\boldsymbol{Pl}$ und $\boldsymbol{Pr}$ so, dass für beliebige a_k .. d_k gilt:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ a_4 & a_3 & a_2 & a_1 \\ d_4 & d_3 & d_2 & d_1 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \boldsymbol{Pl} \cdot \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{pmatrix} \cdot \boldsymbol{Pr}$$

10–41 Turmmatrizen zu elementaren Permutationen

Turmmatrizen, welche sich nur an zwei Stellen von einer Einheitsmatrix unterscheiden: Beinahe-Einheitsmatrizen $\boldsymbol{B} = \boldsymbol{I}$, außer $b_{jk} = 1$, $b_{kj} = 1$ und $b_{jj} = 0$, $b_{kk} = 0$ bewirken nur eine einzige Permutation zwischen j und k. Aus Produkten von solchen Elementarpermutationen kann man jede Turmmatrix herstellen. Für jede $\boldsymbol{B}$-Matrix gilt $\boldsymbol{B}^2 = \boldsymbol{I}$. Testen Sie diese beiden Eigenschaften an einfachen Beispielen, z. B. an der Permutation `[4 3 2 1]` aus `[1 2 3 4]`, bzw. an den Paaren $j, k = 1, 2$, oder 1, 4, oder 2, 4.

Multiplikation von Matrizen

10–42 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!

10–43 Indexwertmatrix mal allgemeine Matrix

Multiplizieren Sie mit Bleistift und Papier je eine 2×2- und eine 3×3-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.

10–44 Matrixmultiplikation „von Hand“

Multiplizieren Sie mit Hilfe von Bleistift und Papier eine 2×2-Matrix $\boldsymbol{D}$ mit den Zahlen (1 .. 4) mit sich selbst (d. h. berechnen Sie $\boldsymbol{D} * \boldsymbol{D}$ von Hand). Multiplizieren Sie ebenso die 3×3-Matrix $\boldsymbol{T}$ mit den Zahlen (1 .. 9) mit sich selbst.

10–45 Formel für die Inverse einer 2×2-Matrix

Multiplizieren Sie mit Bleistift und Papier die zwei allgemeinen 2×2-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 2×2-Matrix.
Im symbolischen Modus kann dieses Gleichungssystem als Matrizengleichung definiert werden.

10–46 Legalitätsüberlegung bei Matrizen- und Vektormultiplikationen
Mit den 4 Matrizen/Vektoren $\boldsymbol{A}$ 2×3, $\boldsymbol{B}$ 3×3, $\boldsymbol{v}$ 3×1, $\boldsymbol{w}$ 2×1 sollen die folgenden Operationen auf ihre Legalität überprüft werden:

$$\begin{array}{lllll} \boldsymbol{A}+\boldsymbol{B} & \boldsymbol{A}+\boldsymbol{A} & \boldsymbol{B}+\boldsymbol{B} & \boldsymbol{v}+\boldsymbol{w} & \boldsymbol{w}-\boldsymbol{v} \\ \boldsymbol{A}*\boldsymbol{B} & \boldsymbol{B}*\boldsymbol{A} & \boldsymbol{A}'*\boldsymbol{B} & \boldsymbol{B}*\boldsymbol{A}' & \\ \boldsymbol{A}*\boldsymbol{v} & \boldsymbol{v}*\boldsymbol{A} & \boldsymbol{v}*\boldsymbol{A}' & \boldsymbol{w}*\boldsymbol{A}' & \boldsymbol{v}*\boldsymbol{v}' \quad \boldsymbol{v}'*\boldsymbol{v} \quad \boldsymbol{B}*\boldsymbol{v} \quad \boldsymbol{B}*\boldsymbol{w} \quad \boldsymbol{A}'*\boldsymbol{v} \\ \boldsymbol{A}*\boldsymbol{A} & \boldsymbol{A}'*\boldsymbol{A} & \boldsymbol{A}*\boldsymbol{A}' & \boldsymbol{B}*\boldsymbol{B} & \end{array}$$

10–47 Legale und illegale Matrixmultiplikationen
Bilden Sie alle möglichen Rechtecksmatrizen, welche die Zahlen 1 bis 6 der Reihe nach enthalten, also $\boldsymbol{E}$ 1×6, $\boldsymbol{Z}$ 2×3, $\boldsymbol{D}$ 3×2 und $\boldsymbol{S}$ 6×1. Bilden Sie zusätzlich deren Transponierte $\boldsymbol{Et}=\boldsymbol{E}'$, $\boldsymbol{Zt}$, etc. und suchen Sie alle legalen Multiplikationen, welche zwischen zwei von diesen 8 Matrizen möglich sind! Bestimmen Sie jeweils die Resultat-Dimensionen.

10–48 Legale und illegale Matrizen- und Vektor-Multiplikationen
Erzeugen Sie die vier Matrizen. bzw. Vektoren $\boldsymbol{A}(4\times3)$, $\boldsymbol{N}(3\times3)$, $\boldsymbol{v}(3\times1)$, $\boldsymbol{w}(1\times4)$, 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: ($\boldsymbol{M}'$ steht für $\boldsymbol{M}^{\mathrm{T}}$, d. h. M-transponiert)

$$\begin{array}{l} \boldsymbol{w}*\boldsymbol{A},\ \boldsymbol{w}'*\boldsymbol{A},\ \boldsymbol{w}*\boldsymbol{A}',\ \boldsymbol{w}'*\boldsymbol{A}',\ \boldsymbol{A}*\boldsymbol{w},\ \boldsymbol{A}*\boldsymbol{w}',\ \boldsymbol{A}'*\boldsymbol{w},\ \boldsymbol{A}'*\boldsymbol{w}' \\ \boldsymbol{w}*\boldsymbol{w},\ \boldsymbol{w}*\boldsymbol{w}',\ \boldsymbol{w}'*\boldsymbol{w},\ \boldsymbol{v}*\boldsymbol{v},\ \boldsymbol{v}'*\boldsymbol{v},\ \boldsymbol{v}*\boldsymbol{v}' \\ \boldsymbol{N}*\boldsymbol{v},\ \boldsymbol{v}*\boldsymbol{N},\ \boldsymbol{v}'*\boldsymbol{N},\ \boldsymbol{N}*\boldsymbol{v}' \\ \boldsymbol{A}*\boldsymbol{N},\ \boldsymbol{A}*\boldsymbol{N}',\ \boldsymbol{A}'*\boldsymbol{N},\ \boldsymbol{A}'*\boldsymbol{N}' \end{array}$$

10–49 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

10–50 In einzelnen Gleichungen vorgegebenes Gleichungssystem
Die folgenden Gleichungen bilden ein lineares Gleichungssystem. Stellen Sie dieses in Matrizenform dar und lösen Sie es mit MATLAB.

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

10–51 Gleichungssystem, durch einzelne Gleichungen gegeben
Lösen Sie das untenstehende Gleichungssystem mit MATLAB im Berechnungs- und im symbolischen Modus:

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

10–52 In einem Schema vorgegebenes Gleichungssystem
Lösen Sie das untenstehende Gleichungssystem mit MATLAB, nachdem Sie es in Matrizenform gebracht haben.

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

10–53 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

10–54 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.

10–55 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“).

10–56 Verschiedene Wurzelfunktionen
Erzeugen Sie eine gemeinsame Grafik der Funktionen

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

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

MATLAB-Operatoren und Grundfunktionen

10–57 MATLAB-Operatoren
Stellen Sie alle Ihnen bekannten Operatoren in MATLAB tabellarisch zusammen und geben Sie bei den speziellen Operatoren stichwortartig deren Bedeutung an!

10–58 Spiegelungsoperationen an Matrizen
Füllen Sie eine 3×3-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.

Erarbeiten einer Funktion

10–59 Winkelfunktionen in Grad
Schreiben Sie selbst Funktions-M-Files `sindeg()`, `cosdeg()`, `tandeg()`, welche gleich funktionieren wie `sind()`, `cosd()`, `tand()`. Multiplizieren Sie dazu die Eingabewerte mit pi/180 und übergeben Sie das Resultat an die normalen Winkelfunktionen. Testen Sie die Anwendung von Matrizen und Vektoren als Eingabewerte.

10–60 Direkt aufrufbare Potenzfunktionen
Erzeugen Sie einige function-M-Files mit den Namen „pow2“, „pow3“, „pow4“, etc., welche die Funktionen `x.^2`, `x.^3`, etc. realisieren.

10–61 Teilvolumen in Pyramiden-Trichter
Suchen Sie die Formel für das Teilvolumen als Funktion des Flüssigkeitspegels in einem Trichter mit Pyramidenform. Die Neigung der Seitenwände soll 45° betragen.

10–62 Teilvolumen in liegendem Zylinder
Suchen Sie die Formel für das Teilvolumen als Funktion des Flüssigkeitspegels in einem liegenden Zylinder. Die Fläche des Kreisabschnittes in Funktion von h kann entweder geometrisch als Sektor minus Dreieck oder als

```
'Integral( Sekantenlänge(y)*dy )'
```

bestimmt werden.

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 min 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 4×4 Einheitsmatrix zu erzeugen.
- Wie erhält man in einer MATLAB-Grafik eine schwarze Linie?
- Wie extrahiert man die ganze 4. Zeile aus einer $n \times n$ 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 8×8 Matrix im Schachbrettmuster mit 0 und 1 zu füllen! Starten Sie mit der Eingabe einer 2×2-Matrix, fügen Sie vier davon zu einer 4×4 Matrix zusammen und dann 4 von diesen zur gesuchten 8×8 Matrix.

T114
Erzeugen Sie einen Vektor mit 505 Elementen, welcher 5 Perioden einer Sägezahnfunktion enthält, wobei jedes Mal die Werte von –10 bis 10 laufen.

T115
Schreiben Sie die MATLAB-Befehle auf zum Lösen des Gleichungssystems im Berechnungs-Modus und im symbolischen Modus.

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

Testserie 1.2

T121 Verständnisfragen

- Wie erhält man in MATLAB die Lösung x eines in Matrixform gegebenen linearen Gleichungssystems $A \cdot x = b$?
- Geben Sie die Befehle an zum Erzeugen der ausmultiplizierten Formel für $(a - b)^5$ im symbolischen Modus.
- 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 3×3 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!

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