

# Einführung in MATLAB

**Peter Hertel**

**Fachbereich Physik  
Universität Osnabrück**

Dieser kurze Text führt in die Benutzung von MATLAB ein. Kenntnisse in Mathematik auf Abiturniveau reichen dafür aus. Komplexe Zahlen sind ausgespart, Grundkenntnisse über Matrizen werden aber vorausgesetzt.

Wir lernen, wie man den Kommandozeilen-Interpreter bedient, wie man Zahlen, Zahlenreihen und Zahlenblöcke (Matrizen) erzeugt und verarbeitet. Damit man dieselben Kommandozeilen nicht immer wieder eintippen muss, kann man sie zu Programmen zusammenfassen und dann als Paket ablaufen lassen. Funktionen sind Programme, die einen oder mehrere Datensätze aufnehmen, diese verarbeiten und ein oder mehrere Ergebnisse abliefern. Dabei wird der Speicherplatz für die Zwischenergebnisse automatisch wieder freigegeben.

Eine Einführung in MATLAB wie diese sollte gleich zu Beginn des Physikstudiums durchgearbeitet werden. Numerische Übungen mit MATLAB werden das gesamte Studium begleiten. Schritt für Schritt können Sie sich so die in dem Programmpaket enthaltenen fast unerschöpflichen Möglichkeiten verfügbar machen.

Diese Einführung kann nur der Anstoß dazu sein. Wir bauen darauf, dass die angehenden Physiker am besten anhand von gut gewählten Beispielen lernen. Man muss sie nur so lange führen, bis sie sich anhand der Dokumentation und anderswie selber weiterhelfen können.

## Inhaltsverzeichnis

<b>1</b>	<b>Kommandozeile</b>	<b>3</b>
<b>2</b>	<b>Matrizen</b>	<b>5</b>
<b>3</b>	<b>Elementenweise Operationen</b>	<b>7</b>
<b>4</b>	<b>Matrixoperationen</b>	<b>8</b>
<b>5</b>	<b>Programme</b>	<b>10</b>
<b>6</b>	<b>Funktionen</b>	<b>13</b>
<b>7</b>	<b>Vermischtes</b>	<b>15</b>
7.1	Zugriff auf eine Matrix . . . . .	15
7.2	Wahr und Falsch . . . . .	15
7.3	Winzig, Unendlich und Unsinn . . . . .	16
7.4	Einfache Graphik . . . . .	17
7.5	Schreiben und Lesen von Dateien . . . . .	17

# 1 Kommandozeile

Wir gehen davon aus, dass Sie vor einer funktionierenden MATLAB-Installation sitzen. Für gewöhnlich<sup>1</sup> werden drei Fenster angezeigt, nämlich *Workspace*, *Command History* und *Command Window*. Das erste zeigt den Arbeitsspeicher, das zweite führt Protokoll über die bisher abgesetzten Befehle, und das dritte empfängt und verarbeitet neue Befehle. Mit dem Cursor sollten Sie das *Command Window* aktivieren.

```
>>
```

und ein blinkender Cursor zeigt an, dass MATLAB auf einen Befehl wartet.

Geben Sie

```
>> x=12.5
```

```
=
```

ein. Das System antwortet, dass  $x$  den Wert 12.500 hat. Zugleich kann man im *Workspace*-Fenster sehen, dass es eine Matrix  $x$  gibt, die den Wert 12.5 hat und die Klasse *double*. Es handelt sich also um eine reelle Zahl doppelter Genauigkeit. Das ist der Standard: 64 bit bzw. 8 byte.

Weil es noch keine Variable  $x$  gab, wird sie durch den Zuweisungsbefehl angelegt. Man kann der Variable nun durch

```
>> x=3.141592654
```

einen neuen Wert zuweisen. Wiederum antwortet MATLAB mit der Feststellung, dass  $x$  den Wert 3.1416 hat. Offensichtlich wurde bei der Ausgabe auf fünfstellige Genauigkeit gerundet. Der Befehl

```
>> format long
```

```
format long
```

ändert das. Die Antwort auf

```
>> x
```

sollte nun 3.14159265400000 sein. Wahrscheinlich war  $\pi$  gemeint, was als vorab definierte Variable<sup>2</sup> zur Verfügung steht:

```
>> x=pi
```

```
pi
```

ergibt 3.14159265358979. Mit

```
>> format short
```

```
short
```

kann man wieder auf grobe Genauigkeit (nur in der Anzeige!) umschalten.

Ein Semikolon als Abschluss eines Befehls unterdrückt das Echo. Nach

```
>> x=1.4142;
```

```
;
```

---

<sup>1</sup>Voreinstellung

<sup>2</sup>Vorsicht: kann undefiniert werden.

erfolgt keine Reaktion im *Command Window*, obgleich ein Blick in das *Workspace*-Fenster den veränderten Wert anzeigt.

Sehr wahrscheinlich war übrigens

sqrt

```
>> x=sqrt(2);
```

gemeint. Überzeugen Sie sich durch einen Blick in das *Workspace*-Fenster über die Wirkung. `sqrt`, die Quadratwurzel, ist eine von hunderten von eingebauten Funktionen. Der Goldenen Schnitt  $(\sqrt{5} - 1)/2$  beispielsweise hat den Wert

+ - \* /

```
>> gs=(sqrt(5)-1)/2;
```

MATLAB-Namen beginnen mit einem Buchstaben und können weitere Buchstaben, Ziffern und den Unterstrich enthalten, so wie `XY_fun12`. Große und kleine Buchstaben gelten als verschieden.

Mit den einmal erzeugten Variablen lässt sich weiterrechnen. Z. B. kann man sich überzeugen, ob der Goldene Schnitt tatsächlich die Lösung der quadratischen Gleichung  $x^2 + x - 1 = 0$  darstellt:

^

```
>> gs^2+gs-1
```

sollte 0 zurückgeben.  $x^2$  ist dasselbe wie `x*x`. Es sind aber auch reellwertige Exponenten zugelassen.

Im *Command History*-Fenster können Sie jeden bisher erteilten Befehl anklicken, er wird dann wieder ausgeführt.

## 2 Matrizen

Unter einer Matrix versteht man bekanntlich eine rechteckige Anordnung von Zahlen. Von links nach rechts durchläuft man eine Zeile, von oben nach unten eine Spalte. In MATLAB werden Zeilen und Spalten durch ganze Zahlen numeriert, beginnend mit 1. Damit steht MATLAB in der Tradition von FORTRAN (und der gesamten Literatur über Numerik), während in C und in den davon abgeleiteten Sprachen C++ und Java die Indizes mit 0 anfangen.

Intern wird eine Matrix als Vektor gespeichert, wobei der Zeilenindex schneller läuft als der Spaltenindex. Wir machen das am besten anhand einer  $2 \times 3$ -Matrix klar:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \quad (1)$$

wird intern als

$$A = \begin{pmatrix} a_1 & a_3 & a_5 \\ a_2 & a_4 & a_6 \end{pmatrix} \quad (2)$$

gespeichert. Die Matrix hat zwei Zeilen und drei Spalten<sup>3</sup>, daher  $R = 2$  und  $C = 3$ . Für  $a_{jk} = a_m$  gilt  $m = j + (k - 1)R$ . Das letzte Matrixelement wird durch  $m = R + (C - 1)R = CR$  indiziert, wie es sein muss.

Wir erzeugen durch den folgenden Befehl

```
>> A=[1,2,3;4,5,6]
```

eine  $2 \times 3$ -Matrix. Die eckigen Klammern fassen die Daten zusammen, mit dem Komma wird von links nach rechts zusammengestellt, mit dem Semikolon von oben nach unten. Auf die Datenelemente kann man entweder gemäß

```
>> x=A(1,3)
```

oder als

```
>> x=A(5)
```

zugreifen. In beiden Fällen sollte übrigens 3 angezeigt werden.

$1 \times N$ -Matrizen heißen Zeilenvektoren,  $N \times 1$ -Matrizes sind Spaltenvektoren. Beispielsweise kann man

```
>> x1=[1,2,3];
```

```
>> x2=[4,5,6];
```

---

<sup>3</sup>englisch *rows* und *columns*

schreiben und dann zu der obigen Matrix montieren,

```
>> A=[x1;x2]
```

Die Größe einer Matrix kann man durch

`size`

```
>> [R,C]=size(A)
```

abfragen. `size` gibt einen Zeilenvektor mit zwei Elementen zurück, dessen Komponenten wir mit den Variablen `R` und `C` belegt haben.

`A'` ist die zu `A` transponierte Matrix. Überprüfen Sie das durch

`'`

```
>> [R,C]=size(A')
```

Mit `linspace(a,b,N)` erzeugt man einen Zeilenvektor von  $N$  gleichmäßig im Intervall  $[a, b]$  verteilten Stützstellen. Hier ein Beispiel:

`linspace`

```
>> x=linspace(-pi,pi,128);
```

`ones`  
`zeros`  
`eye`

`ones(R,C)` und `zeros(R,C)` erzeugen mit Einsen oder Nullen besetzte  $R \times C$ -Matrizen. `eye(N)` liefert die  $N \times N$ -Einheitsmatrix<sup>4</sup>.

---

<sup>4</sup>Lautmalerisch englisch für I, Symbol der Identität

### 3 Elementenweise Operationen

Matrizen können auf einen Schlag mit einer Zahl multipliziert werden. Mit dem  $A$  von oben schreiben wir

```
>> B=0.5*A
```

\*

um alle Elemente zu halbieren. Genauso hätte man

```
>> B=A/2
```

/

schreiben können. Ebenso kann man auf einen Schlag zu einer Matrix eine Zahl  $z$  addieren oder von ihr subtrahieren:

```
>> B=A-1+z
```

+ -

Matrizen der gleichen Größe, wie  $A$  und  $B$ , kann man addieren und subtrahieren. Sehen Sie sich

```
>> B-A
```

+ -

an. Matrizen der gleichen Größe kann man auch punktweise multiplizieren, wie in

```
>> D=A.*A
```

.\*

Sie sollten eine  $2 \times 3$ -Matrix von Quadratzahlen sehen. Auch

```
>> D./A
```

./

ist möglich, wenn kein Eintrag in die Matrix  $A$  verschwindet. Übrigens ist Multiplizieren mit einer Zahl, Addieren und Subtrahieren immer elementen- oder punktweise gemeint. Da Mißverständnisse nicht möglich sind, ist der Punkt bei diesen punktweisen Operationen wegzulassen.

Man kann einen Datensatz punktweise quadrieren wie oben oder auch mit

```
>> D=A.^2
```

.^

Die Matrix  $A$  gewinnt man durch

```
>> sqrt(D)
```

zurück. Wichtig: die eingebaute Funktion `sqrt` kann nicht nur auf Zahlen, sondern auch auf Matrizen angewendet werden.

Wenn  $x$  das Intervall  $[-\pi, \pi]$  approximiert, wie oben, dann ist

```
>> y=sin(x);
```

sin

ein Zeilenvektor der entsprechenden Sinus-Werte. Mit

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

plot

kann man sich den entsprechenden Graphen ansehen. Wir erörtern später, wie man daraus ein schöneres Bild macht.

## 4 Matrixoperationen

Wir ordnen erst einmal

```
clear all
```

```
>> clear all;
```

an, um den Arbeitsspeicher völlig zu löschen.

Matrizen beschreiben lineare Abbildungen. Beispielsweise wird im dreidimensionalen Raum ein Spaltenvektor  $z$  durch die  $3 \times 3$ -matrix  $M$  in den Spaltenvektor  $y = M z$  abgebildet, nämlich gemäß

$$y_j = \sum_{k=1}^3 M_{jk} z_k \text{ für } j = 1, 2, 3 . \quad (3)$$

Mit

```
>> z=[-1;2;4];
```

und

```
>> M=[-1,3,0.5;-0.2,0.9,0.1;0,0.3,1.2];
```

dürfen wir in MATLAB einfach

\*

```
>> y=M*z;
```

schreiben. Das Ergebnis ist ein Spaltenvektor  $y$  mit den Einträgen 9.0, 2.4 und 5.4.

Nun kann man die Frage stellen: Gegeben sei der Spaltenvektor  $y$  und die Matrix  $M$ . Welcher Vektor  $x$  wird mit  $M$  in  $y$  abgebildet? Anders formuliert, man soll das lineare Gleichungssystem  $y = M x$  nach  $x$  auflösen. In MATLAB bewerkstelligt man das durch

\

```
>> x=M\y;
```

Man beachte den Unterschied zum gewöhnlichen Divisions-Operator.

In der Tat stimmen das ursprüngliche  $z$  und  $x$  überein—beinahe. Zwar sehen auch im langen Format  $x$  und  $z$  gleich aus, die Differenz  $z - x$  jedoch ist von der Größen  $10^{-15}$ , das sind ein oder zwei bit in der letzten Stelle. Man kann das durch

norm

```
>> norm(z-x)
```

feststellen. Dass  $z$  in  $y = M z$  und  $x$  in dem linearen Gleichungssystem  $y = M x$  nicht exakt übereinstimmen, hat eine einfache Erklärung. Dezimalzahlen mit endlicher Genauigkeit können im Binärsystem im Allgemeinen nicht mit endlich vielen Stellen dargestellt werden. Daher gibt es Rundungsfehler.

Wir können hier noch nicht erklären, wie die Norm  $\|A\|$  einer Matrix berechnet wird. Sie verschwindet jedenfalls dann und nur dann, wenn  $A$  eine Null-Matrix ist, alle Einträge Nullen sind.

Unser Beispiel kann man verallgemeinern. Die Matrizenmultiplikation  $C = B A$  ist immer dann wohldefiniert, wenn die Zahl  $N$  der Zeilen von  $A$  mit der Zahl der Spalten von  $B$  übereinstimmt.  $C$  hat soviel Zeilen wie  $B$  und soviel Spalten wie  $A$ . Die  $2 \times 3$ -Matrix  $A$  kann mit der  $4 \times 2$ -Matrix  $B$  gemäß  $C = B A$  multipliziert werden.  $C$  ist dann eine  $4 \times 3$ -Matrix. In MATLAB schreibt man `C=B*A`. Das steht für

\*

$$C_{jk} = \sum_{n=1}^N B_{jn} A_{nk} . \quad (4)$$

Die Länge eines Spaltenvektors  $x$  ist

```
>> sqrt(x'*x),
```

während man für einen Zeilenvektor

```
>> sqrt(x*x')
```

schreiben muss. In beiden Fällen stimmt diese Länge mit `norm(x)` überein.

## 5 Programme

Nacheinander auszuführende Befehle kann man in eine Datei schreiben, die die Endung `.m` haben muss. Der Name dieser Datei—ohne Endung—ist für den Kommandozeilen-Interpreter ein Befehl. Allerdings muss die Datei auch gefunden werden. MATLAB sucht im Dateisystem auf einem Suchpfad, und darin muss der Speicherort der `.m`-Datei vorkommen.

`path`            `>> path`  
zeigt den Suchpfad an. Mit

`help`            `>> help path`  
können Sie sich informieren, wie man den Suchpfad verändern kann, sowohl vorübergehend als auch dauerhaft. Mit

`helpdesk`        `>> helpdesk`  
rufen Sie die gesamte Dokumentation zu MATLAB auf. Das ist natürlich auch direkt über die Benutzeroberfläche möglich.

`edit`            `>> edit`  
aktiviert den Editor. Hier ein Beispiel.

```
1  % this file is eml_1.m
2  z=[-1;2;4];
3  M=[-1,3,0.5;-0.2,0.9,0.1;0,0.3,1.2];
4  y=M*z;
5  x=M\y;
6  norm(z-x)
```

`%`

Text, der nach einem Prozent-Zeichen kommt, gilt als Kommentar. Die Zeilennummern gehören nicht zum Programm. Das Programm wird durch die Eingabe

```
>> eml_1
```

ausgeführt. Mit dem Programm erweitert man den Wortschatz von MATLAB um einen neuen Befehl. Es kann vom Kommandozeilen-Interpreter verarbeitet oder in anderen Programmen verwendet werden.

In dem folgenden Beispielprogramm erzeugen wir zwei riesige Matrizen aus Zufallszahlen und multiplizieren diese. Einmal mit dem Blockbefehl `*`, einmal mit geschachtelten `for`-Schleifen, die jeweils durch `end` beendet werden.

`rand`  
`for`  
`:`  
`end`

```
1  % this file is eml_2.m
2  clear all
```

```

3  A=rand(500,1000);
4  B=rand(2000,500);
5  tic;
6  C=B*A;
7  toc
8  clear C;
9  tic;
10 for j=1:2000
11     for k=1:1000
12         sum=0;
13         for n=1:500
14             sum=sum+B(j,n)*A(n,k);
15         end;
16         C(j,k)=sum;
17     end;
18 end;
19 toc

```

`tic` schaltet eine interne Stoppuhr ein, `toc` liest die gestoppte Zeit ab. Auf meinem Rechner waren das 0.75 s und etwa 150 s. `tic`  
`toc`

Mit dem Beispielprogramm sollte gezeigt werden, dass man in MATLAB auch herkömmlich programmieren kann und: dass Blockbefehle viel effizienter sind. Allerdings läßt sich der prozedurale Programmierstil nicht immer vermeiden. Es stehen dafür die üblichen Konstrukte zur Verfügung:

- `if-else-elseif` für die von einer auszuwertenden Bedingung abhängige Programmverzweigung
- `switch-case-otherwise` für die Fallunterscheidung
- `for-end` für eine feststehende Anzahl von Wiederholungen
- `while-end` für die durch eine Bedingung gesteuerte Wiederholung
- `continue-break` für den sofortigen Übergang zur nächsten Wiederholung bzw. zum vorzeitigen Ausstieg aus der Wiederholungsschleife
- `try-catch` um den Normalfall und die Reaktion auf Fehler zu beschreiben
- `return` um eine Funktion vorzeitig zu beenden

Wir werden diese Möglichkeiten zur Steuerung des Befehlsflusses immer dann besprechen, wenn sie wirklich benötigt werden.

Um endlich einmal eine sinnvolle Anwendung vorzuführen, soll die kürzeste Entfernung zwischen zwei Orten auf dem Globus berechnet werden.

Die Orte nennen wir  $\mathbf{s}$  (für *start*) und  $\mathbf{d}$  (für *destination*). Deren Position auf dem Globus wird durch Breite und Länge charakterisiert. Die Breite  $\theta$  ist die Winkelentfernung vom Äquator, wobei nördlich positiv und südlich negativ gerechnet wird. Die Länge  $\phi$  ist die Winkelentfernung von Greenwich, wobei östlich positiv und westlich negativ gerechnet wird. In den Atlanten werden Länge und Breite in Graden angegeben, intern benutzen wir das Bogenmaß:  $360^\circ$  sind  $2\pi$  im Bogenmaß. Zu jedem Ort gehört der Einheitsvektor

$$\mathbf{n} = (\cos \theta \sin \phi, \cos \theta \cos \phi, \sin \theta) . \quad (5)$$

Der Kosinus des Winkels  $\alpha$  zwischen zwei Einheitsvektoren  $\mathbf{n}_1$  und  $\mathbf{n}_2$  ist durch das Skalarprodukt gegeben,

$$\mathbf{n}_1 \cdot \mathbf{n}_2 = \cos \alpha . \quad (6)$$

Dem Winkel  $\alpha$  zwischen zwei Orten entspricht auf dem Globus die Entfernung  $R\alpha$ , wobei  $R$  die Entfernung zum Erdmittelpunkt ist (die wir hier als konstant annehmen). Bekanntlich beträgt der Erdumfang gerade 40000 km.

```

1  % this file is globe.m
2  R=40000/2/pi;
3  slat=input('start latitude(degrees) : ')/180*pi;
4  slon=input('start longitude (degrees) : ')/180*pi;
5  dlat=input('dest latitude(degrees) : ')/180*pi;
6  dlon=input('dest longitude (degrees) : ')/180*pi;
7  sv=[cos(slat)*sin(slon),cos(slat)*cos(slon),sin(slat)];
8  dv=[cos(dlat)*sin(dlon),cos(dlat)*cos(dlon),sin(dlat)];
9  alpha=acos(sv*dv');
10 fprintf('distance is %.1f km\n', alpha*R);

```

`input`  
`acos`  
`fprintf`

`input` fordert zu einer Eingabe von der Tastatur auf. `acos` ist die Umkehrfunktion zum Kosinus (arcus cosinus). Mit `fprintf` wird formatiert ausgegeben. Für Einzelheiten des Formatierungsstrings befrage man die Hilfe.

## 6 Funktionen

Funktionen sind Unterprogramme. Sie nehmen ein oder mehrere Argumente auf und geben ein oder mehrere Ergebnisse zurück. Als Beispiel schreiben wir das Programm zur Berechnung der Großkreisentfernung in eine Funktion um.

```
1 % this file is distance.m
2 function d=distance(p,q);
3 % p=[longitude,latitude] in degrees, q likewise
4 p=pi/180*p; q=pi/180*q;
5 % now in radians
6 R=40000/2/pi; % earth radius in km
7 pv=[cos(p(2))*sin(p(1)),cos(p(2))*cos(p(1)),sin(p(2))];
8 qv=[cos(q(2))*sin(q(1)),cos(q(2))*cos(q(1)),sin(q(2))];
9 d=R*acos(pv*qv'); % distance in km
```

Auch Funktionen werden in `.m`-Dateien abgelegt. Sie beginnen mit dem Schlüsselwort `function` und einem symbolischen Aufruf, aus dem die Argumente (hier `p` und `q`) hervorgehen und der Rückgabewert (hier `d`). Funktionsname und Dateiname (ohne die `.m`-Endung) müssen gleich sein. `function`

Die beiden Argumente sind Zweiervektoren aus Länge und Breite in Graden. Mit

```
>> FRA=[8.57,50.03];
>> PEK=[116.58,40.05];
>> distance(FRA,PEK)
```

berechnet man nun die kürzeste Flugstrecke von Frankfurt am Main nach Beijing (Peking): 7785 km.

Lokale Variable, die innerhalb einer Funktion definiert werden (hier `R`, `pv` und `qv`), sind nach der Ausführung nicht mehr vorhanden. Davon kann man sich im *Workspace*-Fenster überzeugen. Umgekehrt kann man innerhalb der Funktion nur die übergebenen Variablen sehen. Dieser Schutz lässt sich allerdings mit dem `global`-Kommando aushebeln. `global`

Funktionen selber können Argumente sein. Beispielsweise hängt ein Integral von der zu integrierenden Funktion, von der unteren Grenze und von der oberen Grenze ab. Die eingebaute Funktion `quad` ruft man gemäß `quad`

```
>> quad('cos',0,pi/2)
```

auf, um das Integral

$$\int_0^{\pi/2} dx \cos(x) = \sin(\pi/2) - \sin(0) = 1 \quad (7)$$

numerisch zu ermitteln. Die Funktion wird dabei durch ihren Namen (eine Zeichenkette in einfachen Anführungszeichen) gekennzeichnet. Auch die fest eingebaute Kosinusfunktion verhält sich so, als ob sie in einer Datei `cos.m` definiert wäre.

Der Name `quad` für das Integrier-Programm kommt von *Quadratur*. Darunter versteht man den Versuch, irgendein Gebiet durch Operationen, die den Flächeninhalt bewahren, so umzuformen, dass am Ende ein Quadrat entsteht. An der Quadratur des Kreises sind die alten Griechen bekanntlich gescheitert. Sie haben es nicht vermocht, das Integral

$$4 \int_0^1 dx \sqrt{1-x^2} = \pi \quad (8)$$

zu berechnen (weil sie Grenzwerte und die irrationalen Zahlen noch nicht kannten).

Heute<sup>5</sup> schreiben wir in MATLAB:

```
>> f=@(x) sqrt(1-x.*x);
```

Das bedeutet: `f` ist eine Formel mit der Variablen `x`, nämlich `sqrt(1-x.*x)`. Diesen Ausdruck kann man wie eine Funktion verwenden, etwa in

```
>> 4*quad(f,0,1)
```

Das Ergebnis ist 3.1416. Übrigens kann man auch die eingebauten Funktionen wie den Sinus als `@sin` ansprechen.

Man kann natürlich auch eine Datei

```
1 % this file is circle.m
2 function y=circle(x);
3 y=sqrt(1-x.*x);
```

erzeugen und dann

```
>> 4*quad('circle',0,1)
```

aufrufen. Auch

```
>> 4*quad(@(x) sqrt(1-x.*x),0,1)
```

funktioniert.

---

<sup>5</sup>erst ab MATLAB version 7

## 7 Vermischtes

Wir stellen abschließend einige wichtige Konstrukte vor, die sich bisher nicht zwanglos einfügen ließen.

### 7.1 Zugriff auf eine Matrix

Wir haben bisher immer nur eine Matrix als Ganzes verarbeitet oder mit  $A(j,k)$  oder  $A(1)$  auf die Matrixelemente einzeln zugegriffen. In der ersten Form über (Zeilenindex,Spaltenindex), in der zweiten über (Laufindex). Das ist aber nur die halbe Wahrheit.  $j$ ,  $k$  oder  $1$  können nämlich selber wieder Vektoren von Indizes sein! Um solche Indexvektoren zu bilden, ist der Doppelpunktoperator nützlich. Dabei steht  $:$  allein für 'alle erlaubten Indizes'.  $m:n$  sind alle Indizes im Intervall von  $m$  bis  $n$ . Bei zwei Doppelpunkten  $m:d:n$  ist die mittlere Zahl  $d$  die Schrittweite. □

```
>> B=A(:, [1,2,4])
```

etwa stellt die Spalten 1,2 und 4 von  $A$  zu einer neuen Matrix  $B$  zusammen.

### 7.2 Wahr und Falsch

Ob eine Matrix einer logischen Bedingung genügt, wird elementenweise überprüft. Aus der mithilfe von

```
>> A=rand(5,4)
```

erzeugten  $5 \times 4$ -Matrix zufälliger Zahlen wird durch

```
>> B=(A>0.5)
```

eine gleichgroße Matrix mit 1 für 'true' (wahr) und 0 für 'false' (falsch). □

Mit

```
>> sum(sum(B))
```

kann man abzählen, wieviele Einträge größer als 0.5 sind. Zuerst wird über die Spalten summiert, danach über den Zeilenvektor der Spaltensummen. true  
false

Die (laufenden) Indizes der Matrixelemente, die den Wert 0.5 übersteigen, lassen sich mithilfe von sum

```
>> k=find(A>0.5);
```

finden. Will man beispielsweise für einem Datensatz  $x$  punktweise  $y = \sin(x)/x$  ausrechnen, so ist zu berücksichtigen, dass nicht durch Null dividiert werden darf. Vielmehr ist gemäß find

```
1 y=ones(size(x));
```

```

2  k=find(x~=0);
3  y(k)=sin(x(k))./x(k);

```



zu programmieren<sup>6</sup>. Die Tilde steht für das logische 'nicht', ~= mithin für 'ungleich'. Auf Gleichheit wird mit zwei Gleichheitszeichen überprüft.

### 7.3 Winzig, Unendlich und Unsinn

**eps** Das sogenannte Maschinen-Epsilon, die Zahl `eps`, ist die kleinste Zahl, so dass sich `1+0.5*eps` und `1` nicht mehr unterscheiden. Auf keinem Rechner, mit seiner endlichen Speicherfähigkeit, kann man alle reelle Zahlen genau darstellen. `eps` ist ein Maß dafür, wie fein die Zahlengerade unterteilt ist. Probieren Sie

```
>> (1+0.51*eps)-1
```

und danach

```
>> (1+0.49*eps)-1
```

aus.

**Inf** Die wirkliche Zahlengerade ist nach beiden Seiten unbeschränkt. Auf einem Rechner ist das nicht möglich: es gibt eine größte darstellbare reelle Zahl. Wenn das Ergebnis einer Rechnung diese Zahl übersteigt, wird einfach nur noch 'Unendlich' vermerkt, der Wert `Inf`. Unendlich hat ein Vorzeichen. Probieren Sie

```
>> -10^400
```

aus.

**NaN** Wenn das Ergebnis einer Rechnung undefiniert ist, vermerkt MATLAB den Unsinn und gibt `NaN` zurück, 'not a number'. In manchen anderen Programmiersprachen wird entweder Null eingesetzt, oder Unendlich, oder irgendetwas, oder das Programm wird angehalten. Falls der Unsinnswert weiterverarbeitet wird, erhält man wiederum Unsinn. Prüfen Sie das nach durch

```
>> x=0*Inf
```

```
>> x=0*x
```

`NaN` ist fast immer das Anzeichen für einen Programmierfehler. In unserem  $\sin(x)/x$ -Beispiel haben wir vorgeführt, wie man eine `NaN`-Operation, nämlich  $0/0$ , vermeiden kann. MATLAB selber verwendet `Inf` in manchen Funktionen für 'beliebig oft', z. B. beim Durchsuchen einer Datei.

---

<sup>6</sup>Mit  $x \rightarrow 0$  strebt  $\sin(x)/x$  gegen 1.

## 7.4 Einfache Graphik

Viele Sachverhalte in der Physik drückt man durch funktionale Abhängigkeiten vom Typ  $y = f(x)$  aus. Man hat einen Vektor von  $x$ -Werten und zugehörige  $y$ -Werte. Die Datenpunkte  $(x_k, y_k)$  kann man einzeln darstellen oder durch Linienstücke miteinander verbinden. Man kann die Datenpunkte durch Kreise (o), Kreuze (x), Pluszeichen (+), Sterne (\*) usw. kennzeichnen. Für Farben stehen die Buchstaben b, g, r, c, m, y, k zur Verfügung (blau, grün, rot, cyan, magenta, gelb und schwarz). Linien zwischen den Punkten können ausgezogen (-), gepunktet (:) oder gestrichelt (–) sein. Diese Merkmale fasst man in einer Zeichenkette zusammen. Hier ein Beispiel:

```
>> x=linspace(-4,4,64);
>> y=exp(-x.*x);
>> plot(x,y,'rx-');
```

Sie können in das Bild mehr als einen Graphen einzeichnen, indem Sie die entsprechenden  $y$ -Werte zu einer Matrix zusammenfassen, etwa wie in

```
>> plot(x,[y1;y2], 'rx-');
```

Beide Graphen werden gleichartig dargestellt. Es geht aber auch

```
>> plot(x,y1,'rx-',x,y2,'bx-');
```

Die Graphen sind dann rot und blau.

An dem Bild lässt sich alles verändern: die Achsenbeschriftung, die Liniestärken, die Maßstäbe der Achsen usw.

Das Bild können Sie auch abspeichern. Wir empfehlen das Format *encapsulated PostScript*. Mit dem `print`-Befehl wird das Bild farbig auf die Festplatte `print` geschrieben<sup>7</sup>:

```
>> print -depsc 'gaussian.eps';
```

.eps-Dateien sind beliebig skalierbar. Sie können sehr einfach in  $\text{\LaTeX}$ -Dokumente montiert werden.

## 7.5 Schreiben und Lesen von Dateien

Mit

```
>> fid=fopen('test.dat', 'w'); fopen
```

öffnet man eine Datei `'test.dat'`, in die man anschließend schreiben kann (*writing*). Wenn die Datei vorhanden ist, wird sie auf die Länge 0 zurückgesetzt, wenn sie nicht vorhanden war, wird sie erzeugt. Diese Datei kann man nun

---

<sup>7</sup>-`depsc` ist als 'Option Device eps Color' zu lesen

unter dem *file identifier* `fid` ansprechen. (`fid==-1`) zeigt einen Fehler an.

```
>> x=0:0.1:1;
```

```
>> y=exp(x);
```

erzeugt im Arbeitsspeicher eine Tabelle der Exponentialfunktion. Diese kann man nun mit

`fprintf`

```
>> fprintf(fid,'%6.2f %12.8f\n', [x;y]);
```

in die vorbereitete Datei schreiben.

Die Formatierungs-Zeichenkette ist so zu lesen: Zuerst kommt eine Gleitkommazahl (*floating point number*), für die 6 Plätze gebraucht werden, davon zwei Stellen nach dem Dezimalpunkt. Das würde gerade bis -99.99 ausreichen. Dann folgen zwei Leerzeichen, dann kommt ein Gleitkommazahl mit 12 Plätzen, davon 8 nach dem Komma. Anschließend wird das Sonderzeichen `\n` geschrieben, um eine neue Zeile (*newline*) anzufangen.

Diese Formatierungsvorschrift wird immer wieder angewendet, bis `[x;y]` abgearbeitet ist. Man beachte, dass Matrizen spaltenweise ausgelesen werden: zuerst die erste Spalte von oben nach unten, dann die zweite Spalte von oben nach unten, usw. Unsere Matrix hat zwei Zeilen, oben `x`, unten `y`. Das ergibt dann zwei Zahlen auf einer Zeile in der Datei.

Mit

`fclose`

```
>> fclose(fid);
```

schließt man die Datei.

In einem ganz anderen MATLAB-Programm kann man später

```
1  td=fopen('test.dat','r');
2  z=fscanf(td,'%f',[2,inf]);
3  fclose(td);
```

schreiben.

`fscanf`

Zeile 1 öffnet die Datei für den Lese-Zugriff (*reading*). In Zeile 2 wird angeordnet, dass diese Datei immer wieder auf reelle Zahlen durchsucht werden soll. Damit sind Spalten der Länge 2 zu füllen, und zwar so oft es geht. Das Ergebnis `z` wird nicht genau mit dem ehemaligen `[x;y]` übereinstimmen, weil wir nur mit achtstelliger Genauigkeit geschrieben haben.

Es gibt noch sehr viel mehr Möglichkeiten, aus dem Arbeitsspeicher in Geräte zu schreiben und Daten aller Art aus Geräten in den Arbeitsspeicher zu lesen. Mit den hier vorgeführten Möglichkeiten kommt man jedoch schon recht weit.