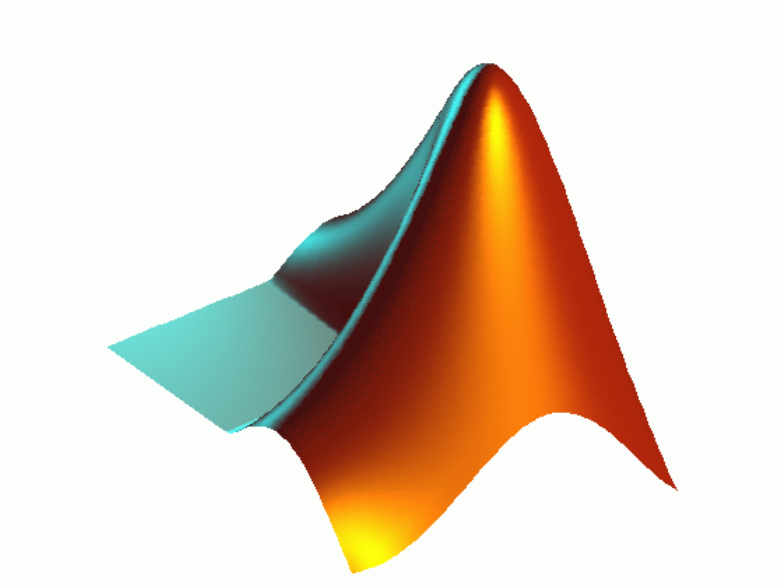
ulm university universität ulm



Wima Praktikum I

Matlab Praktikum - Tag 1

Prof. Dr. Karsten Urban,
Iris Häcker

Institut für Numerische Mathematik

Sommersemester 2013

Praktikumsübersicht

Tag 1 Erste Schritte in Matlab

- Einführung und Motivation
- Einfaches Rechnen
- Rechnen mit Vektoren und Matrizen

Tag 2 Vertiefter Umgang mit Matlab

- Anwendungen zur Matrix und Vektorrechnung
- Weitere Datentypen
- Programmieren

Tag 3 Funktionen in Matlab

- Funktionen
- m-Files
- Debuggen

Tag 4 Graphische Ausgaben mit Matlab

- 2D Plots
- Gestalten von Graphikausgaben
- Mehrdimensionale Plots
- Animationen

Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Warum Matlab?

Was ist Matlab?

Matlab

- ist ein Softwarepaket für numerische Berechnungen und zur Visualisierung;
- wurde in den 1970er Jahren zur Unterstützung von Kursen der Linearen Algebra und numerischen Analysis entwickelt.

Was kann Matlab?

Matlab bietet

- eine einfache Syntax basierend auf dem Matrix-Datentyp;
- ein breites Spektrum mathematischer Funktionen und Algorithmen aus verschiedenen Anwendungsbereichen;
- eine plattformübergreifende Programmiersprache;
- einfach zu bedienende Visualisierungsmöglichkeiten.

Wo finde ich Matlab?

- Pools und Server des kiz ([andromeda, pegasus, perseus, cassiopeia].rz.uni-ulm.de)

```
pegasus$ module avail math/matlab

--------------------- /soft/common/modulefiles/SunOS-sun4u ----------------
math/matlab/R2008b math/matlab/R2009b
pegasus$ module load math/matlab/R2009b
pegasus$ matlab
```

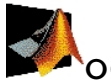
oder

```
pegasus$ cat .profile | grep matlab
module load math/matlab/R2009b;
pegasus$ matlab
```

- Nutzung auf dem eigenen Rechner
  - im Netz der Uni Ulm
  - mit einer Studentenlizenz
    (siehe www.uni-ulm.de → Hochschulportal→Software für Studierende)
    (http://portal.uni-ulm.de/PortalNG/content.title.software.html, erhältlich am Schalter des kiz für 20 Euro)

Matlab starten

Matlab wird gestartet

- über das Symbol auf dem Desktop oder in der Menüleiste  oder
- durch Eingabe von `matlab` im Terminal/in der Shell.

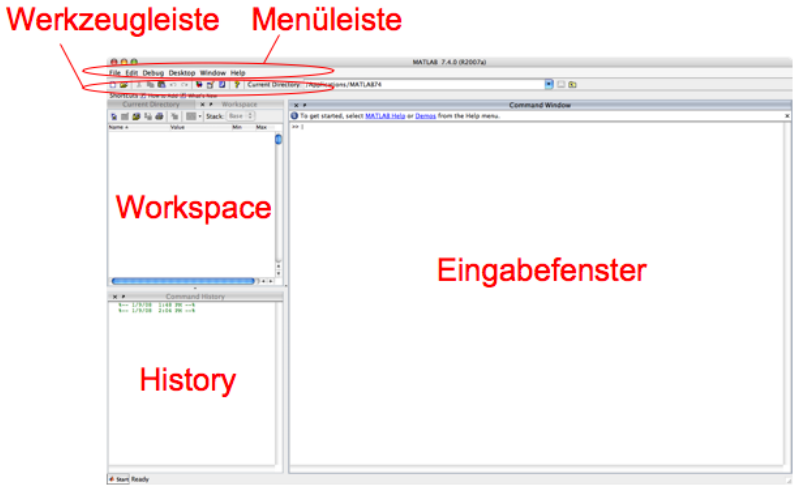
Dadurch wird ein Matlab Fenster geöffnet.
Matlab kann auch durch `matlab -nodesktop` ohne graphische Oberfläche gestartet werden:

```
pegasus$ matlab -nodesktop

                            < M A T L A B (R) >
                  Copyright 1984-2009 The MathWorks, Inc.
                 Version 7.9.0.529 (R2009b) 64-bit (sol64)
                              August 12, 2009


  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

>>
```

Matlab starten

Nach erfolgreichem Start erscheint ein dreigeteiltes Fenster bestehend aus

- Eingabefenster (Command Window): Hier werden die Matlab Befehle eingegeben;
- Workspace Fenster: Zeigt die definierten Variablen an;
- History Fenster: Zeigt die zuletzt eingegebenen Befehle an;



Weitere Fenster wie z.B. der Matlab Editor oder Grafikfenster können beliebig in das Matlabfenster integriert werden.

Matlab wird durch Eingabe von `exit` oder `quit` im Eingabefenster beendet.

Elementares Rechnen in Matlab: erstes Beispiel

Beispiel: Berechne zu einem Kreisradius r die Fläche und den Umfang des Kreises und den Umfang eines flächengleichen Quadrates

```
>> r=3
r =
     3
>> A_Kreis=r^2*pi
A_Kreis =
   28.2743
>> U_Kreis=2*r*pi
U_Kreis =
   18.8496
>> U_Quadrat=4*sqrt(A_Kreis)
U_Quadrat =
   21.2694
```

- Variablen werden durch Zuweisungen eines Wertes mit „=" definiert.
- Namen müssen mit einem Buchstaben anfangen und dürfen Buchstaben, Zahlen und den Unterstrich enthalten. Dabei wird Groß- und Kleinschreibung berücksichtigt.
- Die Grundrechenarten sind durch die Zeichen $+, -, *, /, \hat{}$(potenzieren) definiert.
- Bei den Operatoren gilt die übliche Auswertungsreihenfolge:
  Potenzieren vor Punktrechnung vor Strichrechnung.
  Auswertungsreihenfolgen können durch Klammerung geändert werden.

Elementare Funktionen

Es gibt eine Vielzahl elementarer Funktionen in Matlab:

`exp`, `pow2`	Exponentialfunktion zur Basis e bzw. 2
`log`, `log10`, `log2`	Logarithmus Funktionen
`sqrt`, `realsqrt`	Wurzelfunktionen
`sin`, `cos`, `tan`	Trigonometrische Funktionen
`asin`, `acos`, `atan`	Inverse der trigonometrischen Funktionen
`sinh`, `cosh`, `tanh`	Hyperbelfunktionen
`asinh`, `acosh`, `atanh`	Area Hyperbolicus Funktionen
`abs`, `sign`	Betragsfunktion bei skalarem Argument bzw. Signum
`round`, `floor`, `ceil`	runden, abrunden, aufrunden
`mod`, `rem`, `sign`	Modul, Divisionsrest, Vorzeichen

```
>> sin(pi)
ans =
   1.2246e-16
>> cos(pi)
ans =
    -1
...
```

```
...
>> exp(1)
ans =
    2.7183
>> sqrt(-1)
ans =
         0 + 1.0000i
```

Hilfeseite: `>> help elfun`.

Konstanten in Matlab

In Matlab sind einige spezielle Zahlen definiert:

`realmin, realmax`	kleinste bzw. größte darstellbare Gleitpunktzahl
`eps`	relative Genauigkeit von Gleitpunktzahlen
`inf, -inf`	$\pm\infty$
`NaN`	Not a number, nicht definierter Ausdruck, z.B. `0/0`
`pi`	Kreiszahl π
`i, j`	imaginäre Einheit

```
>> pi*sqrt(-1)
ans =
        0 + 3.1416i
>> 0/0
ans =
   NaN
>> 1/0
ans =
   Inf
>> realmax
ans =
  1.7977e+308
>> realmin
ans =
  2.2251e-308
  >> 1+eps
ans =
    1.0000
```

Elementare Operationen und Funktionen - Beispiele

```
>> 3^3
ans =
    27
>> 16^2/(2*8^2)
ans =
     2
>> pi^2-3^2
ans =
    0.8696
>> (3+4j)^2
ans =
  -7.0000 +24.0000i
```

```
>> sqrt(-1)
ans =
         0 + 1.0000i
>> realsqrt(2)
ans =
    1.4142
>> log(0)
ans =
  -Inf
>> exp(1)
ans =
    2.7183
```

Variablen

- In Matlab werden Variablen durch Zuweisungen ohne vorherige Deklaration angelegt.
- Variablennamen können aus Buchstaben, Ziffern und dem Zeichen _ bestehen, das erste muss ein Buchstabe sein.
- Matlab unterscheidet zwischen Groß- und Kleinschreibung bei Variablennamen (case-sensitive).
- In einem Workspace definierte Variablen können mit den Funktionen `who` und `whos` angezeigt werden.
- Durch Variablendefinition können vorhandene Matlab Funktionen und Variablen überschrieben werden.
- Mit `clear <Variablenname>` bzw. `clear` kann eine Variable bzw. alle Variablen im Workspace gelöscht werden.
- Vorsicht mit den Variablen `i` und `j`:

```
>> i=2
i =
     2
>> pi*i
ans =
    6.2832
>> clear i
>> pi*i
ans =
        0 + 3.1416i
```

Komplexe Zahlen

- Komplexe und reellwertige Zahlen können in Matlab gleichzeitg ohne besondere Deklaration verwendet werden.
- Real- und Imaginärteil einer Zahl können mit den Funktionen `real` bzw. `imag` bestimmt werden.
- Der Betrag einer komplexen Zahl kann mit `abs` bestimmt werden.
- Die Funktion `angle` bestimmt das Argument einer komplexen Zahl.

```
>> z=2*exp(i*pi/3)
z =
   1.0000 + 1.7321i
>> x=real(z)
x =
    1.0000
>> y=imag(z)
y =
    1.7321
...
```

```
...
>> r=abs(z)
r =
     2
>> phi=angle(z)
phi =
    1.0472
>> phi/(2*pi)*360
ans =
    60
```

- Mit `compass` können komplexe Zahlen dargestellt werden.

Speichern von Variablen und IO

- Variablen eines Workspace können mit `save <Dateiname> <Variablenname>` gespeichert und mit `load <Dateiname> <Variablenname>` wieder geladen werden.
- Ebenso kann mit `load` eine Textdatei mit einer Liste von Werten als Matrix eingelesen werden.
- Mit `save <Dateiname>` und `load <Dateiname>` werden alle Variablen des Workspace gespeichert bzw. alle Variablen der Datei geladen.
- Die Ein- und Ausgaben des Workspace in einer Matlab-Sitzung können mit dem Befehl `diary` aufgezeichnet werden:

```
>> diary sitzung.txt
>> r=3
r =
     3
>> h=5
h =
     5
>> V=r^2*h
V =
    45
>> diary off
```

sitzung.txt

```
r=3
r =
     3
h=5
h =
     5
V=r^2*h
V =
    45
diary off
```

Ausgabeformatierung

Ausgaben von Befehlen können mit abschließendem ; unterdrückt werden.
Die Ausgabeformatierung kann mit `format` angepasst werden. Dazu stehen unter anderem folgende Formatierungen zur Verfügung:

- `loose`: Darstellung mit großen Abständen
- `compact`: kompakte Darstellung
- `short`: Festpunktdarstellung mit 5 Stellen
- `long`: Festpunktdarstellung mit 15 Stellen
- `short e`: Gleitpunktdarstellung mit 5 Stellen
- `long e`: Gleitpunktdarstellung mit 15 Stellen
- `rat`: Rationale Näherung

```
>> format loose
>> pi

ans =

    3.1416

>> format compact
>> pi
ans =
    3.1416
>> format long
>> pi
ans =
   3.141592653589793
>> format short e
>> pi
ans =
   3.1416e+00
>> format rat
>> pi
ans =
     355/113
```

Einfache Skripte

- Matlab Befehle können in Textdateien mit Endung `.m` gespeichert und im Workspace durch Eingabe des Dateinamens (ohne Endung) ausgeführt werden. Dazu kann der Matlab Editor `edit` oder jeder andere Texteditor benutzt werden.

Kegel.m

```
% Berechnung des Volumens
% eines Kegels
r=3
h=5
V=1/3*r^2*h
```

```
>> Kegel
r =
     3
h =
     5
V =
    15
>>
```

- Zeilen, die mit einem % beginnen, werden als Kommentarzeilen behandelt.
- Lange Eingaben können durch `...` auf mehrere Zeilen verteilt werden.
- Beim Aufruf im Workspace werden alle Skripte im aktuellen Verzeichnis und im Suchpfad berücksichtigt.
- Mit den Befehlen `pwd`, `cd`, `mkdir` können das aktuelle Arbeitsverzeichnis angezeigt, geändert bzw. neue Verzeichnisse angelegt werden.
- Mit `edit <Dateiname>` wird der Matlab-Texteditor aufgerufen, `type <Skriptname>` zeigt den Inhalt eines m-Files an.
- Die Funktion `what` listet alle m-Files im aktuellen Verzeichnis auf.

Matlab Hilfe im Command Window

In Matlab gibt es ein umfassendes Hilfe-System um Informationen zu allen Funktionen zu bekommen. Es gibt verschiedene Möglichkeiten die Hilfe in Matlab zu nutzen:

- `help` oder `help <Thema>`
  Zeigt eine Übersicht über Hilfethemen oder über ein Thema bzw. einer Funktion im Command Window an;
- `lookfor <Text>`
  Sucht in den Kurzbeschreibungen der Funktionen nach `<Text>` ;

```
>> help sin
 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.


    Reference page in Help browser
       doc sin

>> lookfor lookfor

LOOKFOR Search all M-files for keyword.
```

Matlab Hilfefenster

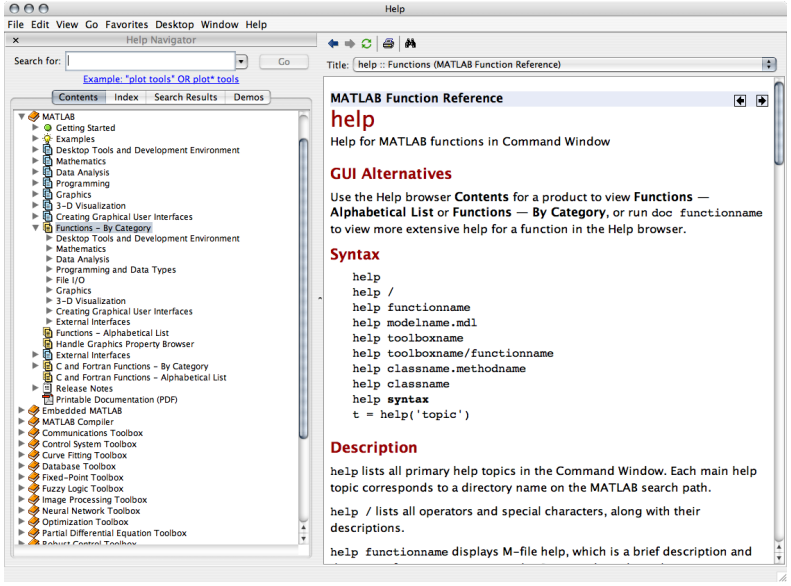
- `helpbrowser`

  Öffnet das graphische Hilfesystem;

- `doc <Thema>`

  Öffnet die Hilfe zum Thema oder zum Funktionsnamen im graphischen Hilfefenster zu einem Thema.

```
>> doc help
```



Einführung

Einfaches Rechnen in Matlab

Rechnen mit Vektoren und Matrizen

Rechnen mit Matrizen und Vektoren - Beispiel

```
>> alpha=pi/4;
>> A=[cos(alpha), -sin(alpha); sin(alpha), cos(alpha)]
A =
    0.7071   -0.7071
    0.7071    0.7071
>> x=1/sqrt(2)*[1; 1]
x =
    0.7071
    0.7071
>> A\x
ans =
    1.0000
    0.0000
>>
```

- Matrizen und Vektoren können in Matlab duch Angabe der Elemente in eckigen Klammern definiert werden.
- Dabei werden die Werte zeilenweise angegeben, Elemente einer Zeile werden durch Kommata oder Leerzeichen voneinander getrennt, verschiedene Zeilen werden durch Semikolon oder Zeilenumbruch getrennt.
- Vektoren werden als Matrizen definiert, wobei die Zeilen- oder Spaltendimension 1 ist.
- In Matlab sind Operatoren zum Rechnen mit Matrizen, Vektoren und Skalaren definiert.

Operatoren für Matrizen

- Operationen zwischen zwei Matrizen / Vektoren: +, −, * zum Addieren, Subtrahieren, Multiplizieren.

```
>> alpha=pi/5;
>> A=[cos(alpha), -sin(alpha); sin(alpha) cos(alpha)];
>> B=[cos(-alpha), -sin(-alpha); sin(-alpha) cos(-alpha)];
>> C=A+B
C =
    1.6180         0
         0    1.6180
>> A*B
ans =
     1     0
     0     1
```

- Operatoren zum Lösen linearer Gleichungssysteme: /, \.

```
>> B
B =
    0.8090    0.5878
   -0.5878    0.8090
>> A\[1 0; 0 1]
ans =
    0.8090    0.5878
   -0.5878    0.8090
```

Operatoren für Matrizen und Skalare

- Operationen zwischen Matrizen/Vektoren und Skalar: skalare Multiplikation mit den Operatoren * und /:

```
>> 3*[1 1; 0 1]
ans =
     3     3
     0     3
>> [1 1; 0 1]/3
ans =
    0.3333    0.3333
         0    0.3333
```

- Potenzieren mit ^:

```
>> [1 1; 0 1]^2
ans =
     1     2
     0     1
```

- Addition und Subtraktion mit + und −:

```
>> [1 1; 0 1]+3
ans =
     4     4
     3     4
```

Komponentenweise Operationen für Matrizen/Vektoren

- Komponentenweise Multiplikation und Division: skalare Multiplikation mit den Operatoren `.*` und `./`:

```
>> A=[2 4; 6 9];
>> B=[2 2; 3 6];
>> A.*B
ans =
     4     8
    18    54
>> A./B
ans =
    1.0000    2.0000
    2.0000    1.5000
```

- Komponentenweises Potenzieren mit `.^`:

```
>> A.^B
ans =
           4          16
         216      531441
```

Spezielle Matrizen

Für häufig verwendete Matrizen gibt es Funktionen mit denen diese Matrizen erzeugt werden können:

- Einsmatrix- bzw. -vektor:
  `ones(n)`, `ones(n,m)`
- Nullmatrix- bzw. -vektor:
  `zeros(n)`, `zeros(n,m)`
- Einheitsmatrix bzw. -vektor:
  `eye(n)`, `eye(n,m)`
- Zufallsmatrix bzw. -vektor:
  `rand(n,m)`, `randn(n,m)`
- Diagonalmatrix zu einem Vektor mit Diagonalelementen oder Vektor der Diagonalelemente einer Matrix:
  `diag(x)`, `diag(A)`
- Magisches Quadrat:
  `magic(n)`

Mit `gallery` können noch weitere spezielle Matrixformen erzeugt werden.
Hilfeseite: `>> help elmat`.

```
>> eye(2,3)
ans =
     1     0     0
     0     1     0
>> ones(3)
ans =
     1     1     1
     1     1     1
     1     1     1
>> rand(3)
ans =
    0.4898    0.7094    0.6797
    0.4456    0.7547    0.6551
    0.6463    0.2760    0.1626
>> magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2
>> gallery('jordbloc',3,2)
ans =
     2     1     0
     0     2     1
     0     0     2
```

Spezielle Vektoren

Ebenso gibt es Funktionen für häufig verwendete Vektortypen:

- Sequenz von Zahlen mit festem Inkrement: `a:b:c` bzw. `a:c` für das Inkrement 1.
- lineare Unterteilung eines Intervalls in Teilintervalle: `linspace(a,b,n)`
- logarithmische Unterteilung eines Intervalls in Teilintervalle: `logspace(a,b,n)`

```
>> 2.3:.7:5.4
ans =
    2.3000    3.0000
3.7000    4.4000    5.1000
>> 1:-.7:-2.4
ans =
    1.0000    0.3000
-0.4000   -1.1000   -1.8000
>> 2.4:5.6
ans =
    2.4000    3.4000
4.4000    5.4000
>> linspace(exp(1), pi, 5)
ans =
    2.7183    2.8241
2.9299    3.0358    3.1416
>> logspace(pi, exp(1), 5)
ans =
   1.0e+03 *
    1.3855    1.0858
0.8510    0.6670    0.5227
```

Matrixindizierung

Auf Komponenten von Matrizen/Vektoren kann mit dem ( ) Operator zugegriffen werden. Dazu können die Elemente auf zwei verschiedene Arten indiziert werden:

über Zeilen- und Spaltenindizes

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & & a_{2,n} \\ \vdots & & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

über Indizes der Elemente

$$A = \begin{pmatrix} a_{1} & a_{m+1} & \cdots & a_{(n-1)m+1} \\ a_{2} & a_{m+2} & & a_{(n-1)m+2} \\ \vdots & & \ddots & \vdots \\ a_{m} & a_{2m} & \cdots & a_{mn} \end{pmatrix}$$

Hierbei werden die Elemente, im Gegensatz zur Eingabe, spaltenweise nummeriert.

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

Matrixindizierung

Mit dem Klammeroperator kann . . .

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

. . . auf einzelne Elemente . . .

```
>> A(2, 2)
ans =
     5
>> A(2)
ans =
     4
```

. . . oder auf Teilmatrizen zugegriffen werden.

```
>> A(1:4)
ans =
     1     4     2     5
>> A(1, 2:3)
ans =
     2     3
```

Verändern und Zusammensetzen von Matrizen - cont'd

Über den Zugriff auf Komponenten einer Matrix kann diese ausgelesen und verändert werden:

```
>> A(1, 2:3)=zeros(1,2)
A =
     1     0     0
     4     5     6
```

Matrizen können aus Teilmatrizen passender Größe zusammengesetzt werden. Nützlich ist auch die Funktion `blckdiag`, die Matrizen entlang der Diagonalen anordnet.

```
>> A=[1:4; eye(1,2), zeros(1,2)]
A =
     1     2     3     4
     1     0     0     0
```

Mit `reshape` können die Dimensionen einer Matrix verändert werden:

```
>> B=reshape(1:4, 2, 2)
B =
     1     3
     2     4
```

Verändern und Zusammensetzen von Matrizen

Mit dem Operator [] können Zeilen oder Spalten von Matrizen und Vektoren gelöscht werden.

```
>> A(:,2:3)=[]
A =
     1     4
     1     0
```

Um eine Matrix zu transponieren bzw. die komplex konjugierte zu bestimmen gibt es die Operatoren .' und ':

```
>> B'
ans =
     1     2
     3     4
```

Mit den Funktionen `fliplr` bzw. `flipud` kann die Reihenfolge der Spalten bzw. Zeilen der Matrix vertauscht werden:

```
>> fliplr(B)
ans =
     3     1
     4     2
>> flipud(B)
ans =
     2     4
     1     3
```

Matrix-Abmessungen

Die Abmessungen einer Matrix kann mit der Funktion `size A` bzw. `[z,s]=size(A)` ermittelt werden,

```
>> A=[1 2 3; 4 5 6];
>> [z, s]=size(A)
z =
     3
s =
     4
```

mit `length(A)` kann man die Länge eines Vektors ermitteln.

```
>> length(2.3:0.4:8.9)
ans =
    17
```

Die Funktion `numel(A)` gibt die Anzahl der Elemente von `A` zurück, mit `isempty(A)` kann man ermitteln, ob die Matrix leer ist.

```
>> numel(A)
ans =
    12
>> isempty(A)
ans =
     0
>> isempty([])
ans =
     1
```

Teilmatrizen

Weitere Möglichkeiten auf Teile von Matrizen zuzugreifen sind z. B.:

Zugriff auf die Diagonalelemente mit `diag`. Die Funktion `diag` angewand auf einen Vektor erzeugt eine Diagonalmatrix:

```
>> A=magic(3);
>> diag(A)
ans =
     8
     5
     2
>> diag(diag(A))
ans =
     8     0     0
     0     5     0
     0     0     2
```

Die Funktionen `triu` und `tril` liefern die obere bzw. untere Dreiecksmatrix:

```
>> triu(A)
ans =
     8     1     6
     0     5     7
     0     0     2
>> tril(A)
ans =
     8     0     0
     3     5     0
     4     9     2
```

Funktionen für skalare Kenngrößen einer Matrix

Funktionen, mit denen Matrizen charakterisiert werden, sind

die Spur `trace`

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii},$$

```
>> trace(A)
ans =
    15
```

den Rang `rank`, also die Dimension des Bildraums einer Matrix

$$\dim(\mathrm{Bild}(A)),$$

```
>> rank(A)
ans =
     3
```

die Determinante `det`

$$\det(A) = \sum_{\sigma \in S_n} \mathrm{sign}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)},$$

```
>> det(A)
ans =
  -360
```

(wird nicht so berechnet!!)

und die Kondition `cond`

$$\kappa(A) = \frac{\max_{\|x\|=1} \|Ax\|}{\min_{\|x\|=1} \|Ax\|}$$

```
>> cond(A)
ans =
    4.3301
```

Weitere Kenngrößen von Vektoren und Matrizen

Die Norm eines Vektors oder einer Matrix:

$$\|x\| = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}},$$

$$\|A\| = \max_{\|x\|=1} \|Ax\|;$$

```
>> norm(A)
ans =
    15
>> a=A(1,:)
a =
     8     1     6
>> norm(a)
ans =
    9.4340
```

die Summer der Elemente eines Vektors, bzw. die Summe der Elemente in jeder Spalte einer Matrix

```
>> sum(a)
ans =
    15
```

maximales bzw. minimales Element eines Vektors:

```
>> min(a)
ans =
     3
>> max(a)
ans =
     8
```

Bild und Kern einer Matrix

Zum Berechnen von Bild und Kern einer Matrix sind Matlab Funktionen implementiert. Diese Funktionen sind

`null` zur Berechnung einer Orthonormalbasis des Kerns, also der Vektoren x, für die $Ax = 0$ gilt:

```
>> A=[1 2 3; 4 5 6; 7 8 9];
>> null(A)
ans =
   -0.4082
    0.8165
   -0.4082
```

`orth`, die eine orthonormale Basis des Bildraums berechnet:

```
>> orth(A)
ans =
   -0.2148    0.8872
   -0.5206    0.2496
   -0.8263   -0.3879
```

Eigenwertanalyse

Um eine Eigenwertanalyse durchzuführen, können folgende Funktionen benutzt werden:

`poly` berechnet die Koeffizienten des charakteristischen Polynoms einer Matrix

$$c(A) = \det(A - \lambda I);$$

```
>> poly(A)
ans =
    1.0000  -15.0000
-18.0000   -0.0000
```

`eig` bestimmt die Eigenwerte s_i und Eigenvekoren v_i einer Matrix, also Skalare und Vektoren für die

$$Av_i = s_i v_i$$

gilt. Die Ergebnisse werden als Diagonalmatrix und eine Orthonormalbasis aus Eigenwerten als Spaltenvektoren einer Matrix gespeichert.

```
>> [V,S]=eig(A)
V =
   -0.2320   -0.7858    0.4082
   -0.5253   -0.0868   -0.8165
   -0.8187    0.6123    0.4082
S =
   16.1168         0         0
         0   -1.1168         0
         0         0   -0.0000
```

Singulärwertzerlegung

Mit der Funktion `svd` kann eine Singulärwertzerlegung einer Matrix A berechnet werden, also eine Zerlegung

$$A = USV$$

mit einer Diagonalmatrix S und Transformationsmatrizen U und V.

```
>> [U, S, V] = svd(A)
U =
   -0.2148    0.8872    0.4082
   -0.5206    0.2496   -0.8165
   -0.8263   -0.3879    0.4082
S =
   16.8481         0         0
         0    1.0684         0
         0         0    0.0000
V =
   -0.4797   -0.7767   -0.4082
   -0.5724   -0.0757    0.8165
   -0.6651    0.6253   -0.4082
```

Hilfeseite: `help matfun`

Matrixzerlegung

Weitere Funktionen zur Zerlegung von Matrizen sind `chol`, `lu` und `qr` berechnen die Cholesky-Zerlegung, eine LR-Zerlegung bzw. eine QR-Zerlegung einer Matrix. Bei der LR- bzw. QR-Zerlegung werden Matrizen bestimmt, so dass

$$A = LU \quad \text{bzw.} \quad A = QR$$

mit einer rechten oberen Dreiecksmatrix U bzw. R und einer linken unteren Dreiecksmatrix L bzw. einer orthogonalen Matrix Q gilt. Die Cholesky-Zerlegung kann auf symmetrische, positiv definite Matrizen angewendet werden und ergibt ähnlich der LR-Zerlegung eine Faktorisierung

$$A = L^T L$$

```
>> A=magic(2);
>> [L, U]=lu(A)
L =
    0.2500    1.0000
    1.0000         0
U =
    4.0000    2.0000
         0    2.5000
>> [Q, R]=qr(A)
Q =
   -0.2425   -0.9701
   -0.9701    0.2425
R =
   -4.1231   -2.6679
         0   -2.4254
>> A=[2 1; 1 2];
>> L = chol(A)
L =
    1.4142    0.7071
         0    1.2247
>> L'*L
ans =
    2.0000    1.0000
    1.0000    2.0000
```

Lösen von Gleichungssystemen

Zum Lösen von linearen Gleichungssystemen ist in Matlab der \-Operator definiert. Hierbei wird das Gleichungssystem mit einer QR-Zerlegung gelöst falls die Matrix invertierbar ist, andernfalls wird ein least-squares Problem gelöst. Genaueres dazu in den Übungen.

Um die Inverse einer invertierbaren Matrix zu bestimmen gibt es darüber hinaus noch die Funktion `inv`:

```
>> A=magic(2)
A =
     1     3
     4     2
>> inv(A)
ans =
   -0.2000    0.3000
    0.4000   -0.1000
```

Für nichtinvertierbare bzw. nichtquadratische Matrizen kann eine Pseudoinverse, also eine Matrix X berechnet werden, für die gilt

$$AXA = X \quad \text{und} \quad XAX = A$$

```
>> A=[magic(2),[1;2]]
A =
     1     3     1
     4     2     2
>> pinv(A)
ans =
   -0.2000    0.2667
    0.4000   -0.1167
    0.0000    0.0833
```

Matrixoperationen

```
>> A=[-1 1 1; 0 -2 -3; 4 0 3]
A =
    -1     1     1
     0    -2    -3
     4     0     3
>> rank(A)
ans =
     3
>> det(A)
ans =
     2
>> trace(A)
ans =
     0
>> null(A)
ans =
   Empty matrix: 3-by-0
>> orth(A)
ans =
    0.0572    0.5109    0.8578
   -0.4824   -0.7381    0.4717
    0.8741   -0.4407    0.2042
...
```

```
...
>> [V,D]=eig(A)
V =
   -0.0880   -0.3811    0.5060
    0.4835   -0.7907   -0.7988
   -0.8709    0.4791   -0.3253
D =
    3.4040         0         0
         0   -0.1824         0
         0         0   -3.2217
>> p=poly(A)
p =
    1.0000    0.0000
-11.0000   -2.0000
>> roots(p)
ans =
    3.4040
   -3.2217
   -0.1824
>> inv(V)*A*V
ans =
    3.4040   -0.0000   -0.0000
   -0.0000   -0.1824   -0.0000
   -0.0000    0.0000   -3.2217
```

Funktionsauswertungen mit Matrizen

- Skalare Funktionen angewendet auf Matrizen oder Vektoren werden komponentenweise ausgewertet.

```
>> A=magic(2)
A =
     1     3
     4     2
>> factorial(A)
ans =
     1     6
    24     2
```

- Dadurch können Funktionen einfach auf diskreten Gittern ausgewertet werden

```
>> x=0:pi/4:pi;
>> sin(x)
ans =
         0    0.7071    1.0000    0.7071    0.0000
```

- Für Funktionen, die über eine Reihendarstellung definiert werden können und somit auch mit Matrizen als Argument sinnvoll definiert sind, gibt es spezielle Funktionen, z. B. `expm`, `polyvalm`, `sqrtm`, `logm`.

```
>> expm(magic(2))
ans =
   63.6830   63.5476
   84.7302   84.8655
```

```
>> exp(magic(2))
ans =
    2.7183   20.0855
   54.5982    7.3891
```

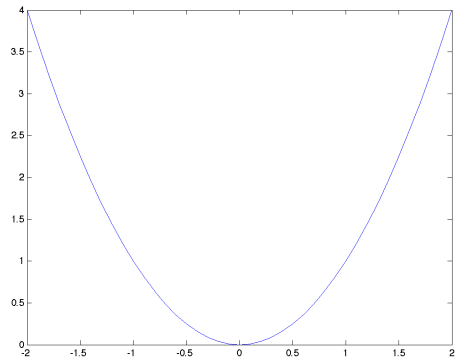
Elementares Plotten

Funktionen können in Matlab auf zwei Arten geplottet werden:

- Durch Plotten von endlich vielen Wertepaaren $(x, f(x))$, die ausgewertet und anschließend an die Funktion `plot` übergeben werden:

```
>> x=[-2:0.1:2];
>> y=x.^2;
>> plot(x,y);
```

Mit `plot` können auch mehrere Graphen mit unterschiedlichen Linientypen geplottet werden. Näheres dazu in der Hilfe...



- Durch Angabe einer Inline-Funktion, die zusammen mit einem Intervall an die Funktion `ezplot` übergeben wird:

```
>> ezplot('sin(x)', [0,2*pi]);
```