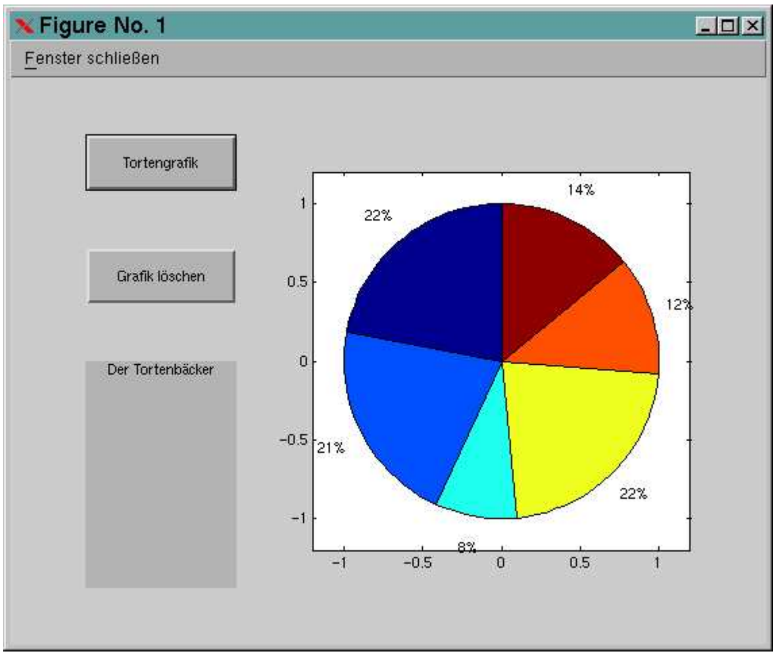
Mathematik–Online–Kurs

MATLAB



http://www.mathematik-online.org/

Mathematik–Online–Kurs

MATLAB

Stand: 13. November 2003

Konzipiert von J. Hörner und J. Wipper

`http://www.mathematik-online.org/`

Inhaltsverzeichnis

Kapitel 1

Grundlagen

1.1 Einführung

1.1.1 MATLAB

MATLAB ist eine Abkürzung von „MATrix LABoratory“. Hersteller ist die Firma MathWorks (`http://www.mathworks.de`).

Anwendungsgebiete:

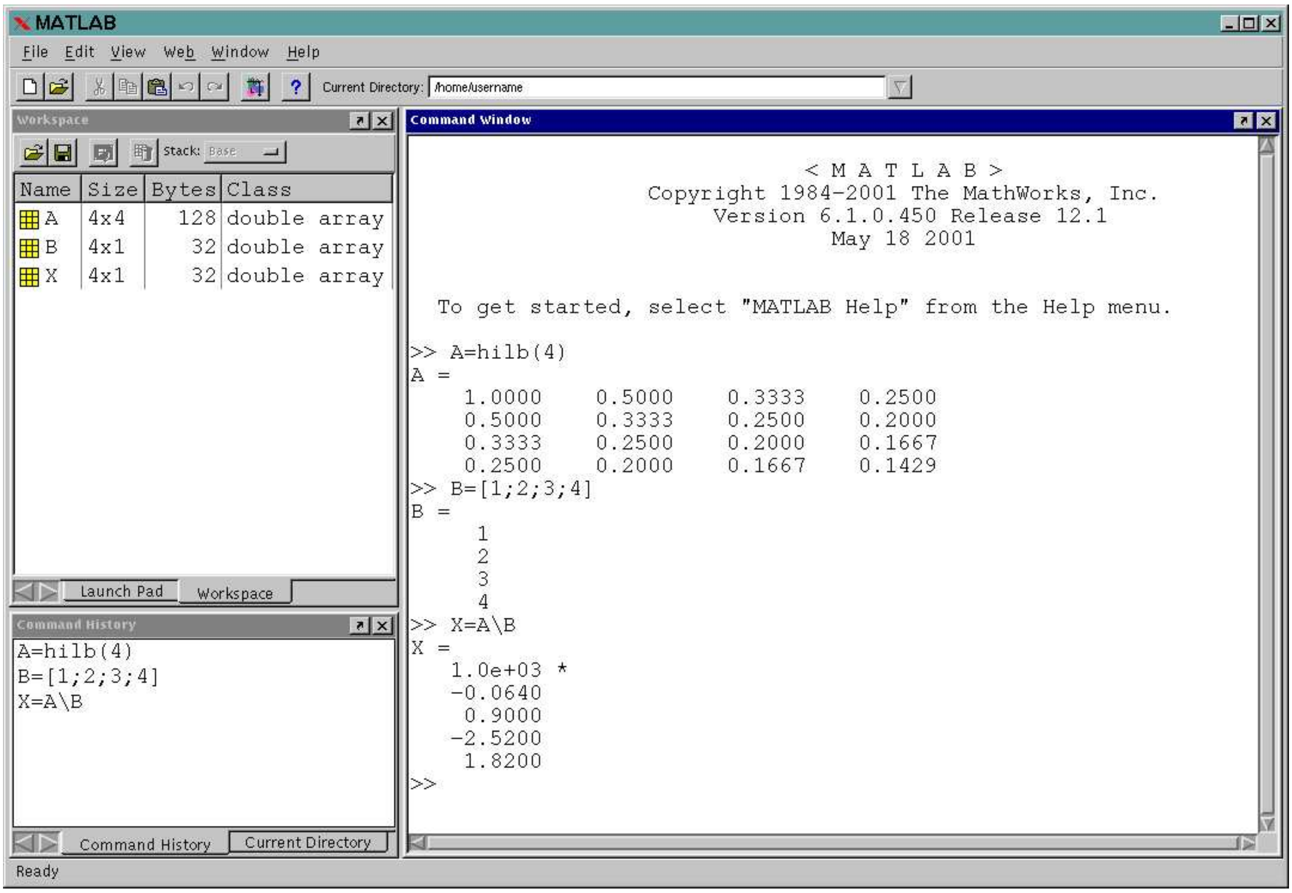
- Numerische Berechnungen
- Visualisierung von Daten
- Entwicklung von Algorithmen und graphischen Benutzeroberflächen

Vorteile:

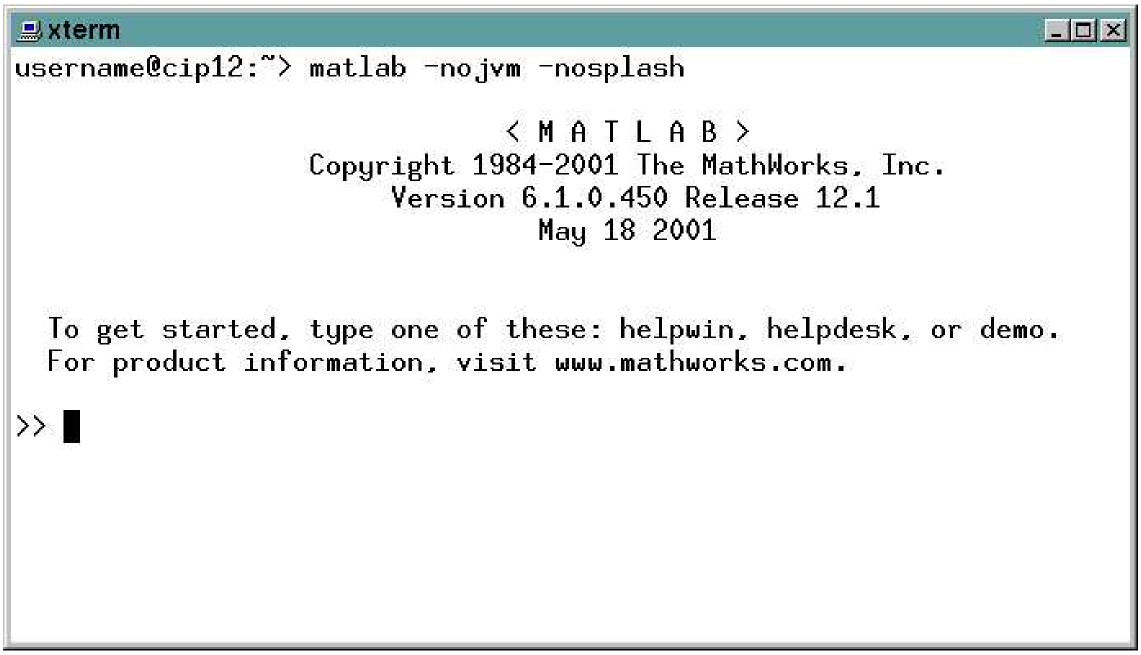
- Großer Funktionsumfang
- Einfach zu erlernende und mächtige Programmiersprache
- Auf den wichtigsten Rechnersystemen verfügbar
- Ausführliche Dokumentation

1.1.2 Benutzeroberfläche

Desktop



Kommando-Fenster



- Eingabeaufforderung: >>
- Befehlswiederholung mit Cursor-Tasten
- Einschränkung durch Befehlsanfang
- Namensergänzung mit Tabulatortaste (ab MATLAB 6)

- Beenden mit `quit` oder `exit`

1.1.3 Hilfe

- `help`: Überblick über alle Funktionsgruppen
- `help` <Gruppe>: Überblick über die Funktionen der Gruppe
  z.B. `help elfun` für die Elementar-Funktionen
- `help` <Funktion>: Beschreibung der Funktion
- `helpwin`: Hilfetexte in eigenem Fenster
- `helpdesk`: HTML-Version der Hilfe
- `lookfor` <Stichwort>: Stichwortsuche in den Hilfetexten
- `demo`: Demonstrationsprogramme
- `type` <Funktion>: Quellcode einer Funktion anzeigen
- In den Hilfetexten sind MATLAB-Funktionen durch Großbuchstaben hervorgehoben

Beispiel:

```
>> help cross

 CROSS  Vector cross product.
    C = CROSS(A,B) returns the cross product of the vectors
    A and B.  That is, C = A x B.  A and B must be 3 element
    vectors.

    C = CROSS(A,B) returns the cross product of A and B along the
    first dimension of length 3.

    C = CROSS(A,B,DIM), where A and B are N-D arrays, returns the cross
    product of vectors in the dimension DIM of A and B. A and B must
    have the same size, and both SIZE(A,DIM) and SIZE(B,DIM) must be 3.

    See also DOT.
```

1.2 Zahlen

1.2.1 (Rationale) Zahlen

- Eingabe im Dezimalsystem
- Form: [Vorzeichen] Zahl [Exponent]
  - Vorzeichen: + oder −
  - Zahl: mindestens eine Ziffer, ggf. Dezimalpunkt

  - Exponent: e oder E gefolgt von ganzer Zahl

- Speicherung im Binärsystem:
  - 1 Bit Vorzeichen
  - 11 Bit Exponent
  - 52 Bit Mantisse

- Bereich:
  - kleinste positive Zahl: $1 \cdot 2^{\left(2-2^{10}\right)} \approx 2.225074 \cdot 10^{-308}$
  - größte Zahl: $(2-2^{-52}) \cdot 2^{\left(2^{10}-1\right)} \approx 1.797693 \cdot 10^{+308}$
  - (relative) Genauigkeit: $2^{-52} \approx 2.220446 \cdot 10^{-16}$

Beispiele:

```
>> 1.2345e-78
ans =
   1.2345e-78

>> format rat
>> 11111111111111111111
ans =
11111111111111114752

>> 1e-400
ans =
     0
```

1.2.2 Komplexe Zahlen

- 2 rationale Zahlen durch + verknüpft
- Imaginärteil durch `i` oder `j` gekennzeichnet
- Bei Imaginärteil 0 wird nur Realteil angezeigt

Beispiele:

```
>> 1.2+3.4e5i
ans =
  1.2000e+00 + 3.4000e+05i

>> 1.2j+3.4e5
ans =
  3.4000e+05 + 1.2000e+00i

>> 0+4i
ans =
  0 + 4.0000i
```

```
>> 4+0i
ans =
  4
```

1.2.3 Spezielle Zahlen

- `Inf`:
  - steht für ∞ (Infinity)
  - entsteht bei Überlauf
  - ist vorzeichenbehaftet
- `NaN`:
  - steht für Not a Number
  - entsteht bei Berechnungen mit undefiniertem Ergebnis
  - Operationen mit `NaN` ergeben `NaN`

Beispiele:

```
>> 1e309             >> 1e308-Inf           >> Inf-Inf
ans =                ans =                  ans =
   Inf                 -Inf                    NaN

>> 1/Inf             >> Inf+Inf             >> 1/NaN
ans =                ans =                  ans =
     0                  Inf                    NaN
```

1.3 Terme, Funktionen und Variablen

1.3.1 Terme

- Operatoren: `+`, `.+`, `-`, `.-`, `*`, `.*`, `/`, `./`, `\`, `.\`, `^`, `.^`
- Auswertung
  - Potenz, Punkt, Strich
  - von links nach rechts
- Klammerung mit `( )`

Beispiele:

```
>> 1+2*3^4
ans =
  163

>> 3^3^3
ans =
```

```
  19683

>> 3^(3^3)
ans =
  7625597484987

>> 128\768/3
ans =
  2

>> 2*3*128
ans =
  768
```

1.3.2 Element(ar)-Funktionen

- Ohne Argument: `pi`, `i`, `j`, `eps`, `realmin`, `realmax`
- Mit Argument: Argument in `( )`, z.B. `exp(1)`
- Trennung mehrerer Argumente: durch Kommata, z.B. `mod(5,2)`
- Trigonometrisch: `sin, sinh, asin, asinh` (Winkel in Bogenmaß) analog für `cos, tan, cot, sec, csc`
- Exponentiell: `exp, log, log10, log2, pow2, sqrt, nextpow2`
- Komplex: `abs, angle, complex, conj, imag, real`
- Runden: `fix, floor, ceil, round`
- Division mit Rest: `mod, rem`
- Vorzeichen: `sign`

Beispiele:

```
>> exp(2*pi*j)
ans =
   1.0000 - 0.0000i

>> complex(1,2)
ans =
  1.0000 + 2.0000i

>> abs(1+2i)
ans =
    2.2361

>> angle(1+2i)
ans =
    1.1071
```

```
>> floor(-1.5)
ans =
    -2

>> ceil(-1.5)
ans =
    -1

>> round(-1.5)
ans =
    -2

>> fix(-1.5)
ans =
    -1
```

1.3.3 Variablen

- Variablennamen beginnen mit einem Buchstaben
- Bestehen aus bis zu 31 Zeichen. Erlaubt sind: Buchstaben (keine Umlaute), Zahlen und `_`
- Sind case sensitive, d.h. zwischen Groß- und Kleinbuchstaben wird unterschieden
- Typdeklaration erfolgt indirekt durch Zuweisung
- Zuweisung von Werten mit `=`
- Rückmeldung unterdrücken mit abschließendem `;`
- Nicht zugewiesene Rückgabewerte werden in der speziellen Variable `ans` (answer) gespeichert
- Anzeige der definierten Variablen mit `who` bzw. `whos`
- Löschen mit `clear` <Variable>. `clear` alleine löscht alle Variablen
- Variablen überdecken Funktionen. Daher sollten Funktionsnamen vermieden werden (besonders gefährdet sind die Funktionen `i` und `j`, welche die imaginäre Einheit zurückliefern)

Beispiele:

```
>> V_3456789012345678901234567890l234567890=31
V_3456789012345678901234567890l =
    31

>> A=5*i
A =
        0 + 5.0000i

>> i=4
i =
     4

>> A=5*i
A =
    20

>> A=5i
A =
        0 + 5.0000i

>> who
Your variables are:
A         ans       i

>> clear i
```

```
>> i
ans =
        0 + 1.0000i
```

1.4 Matrizen und Vektoren

1.4.1 Matrizen

- Wichtigstes Datenformat in MATLAB
- Eingabe in `[ ]`
- Einträge einer Zeile durch , oder Leerzeichen getrennt
- Zeilen durch ; oder Zeilenschaltung getrennt
- Zeilenfortsetzung durch `...`
- Vektoren sind $(1 \times n)$- bzw. $(n \times 1)$-Matrizen
- Skalare sind (1×1)-Matrizen (Klammern können entfallen)

Beispiele:

```
>> [1,2,3;4,5,6;7,8,9]
ans =
     1     2     3
     4     5     6
     7     8     9

>> [1 2 3
    4 5 6
    7 8 9]
ans =
     1     2     3
     4     5     6
     7     8     9

>> [ 1, 2, 3, 4,...
     5, 6, 7, 8,
    11,12,13,14,...
    15 16 17 18]
ans =
     1     2     3     4     5     6     7     8
    11    12    13    14    15    16    17    18
```

1.4.2 Spezielle Matrizen

- `[ ]`: leere Matrix (0×0)
- `ones(n,m)`: ($n \times m$)-Matrix bei der alle Einträge 1 sind
- `zeros(n,m)`: ($n \times m$)-Matrix bei der alle Einträge 0 sind
- `eye(n,m)`: ($n \times m$)-Matrix bei der nur die Einträge auf der Hauptdiagonalen 1 sind
- `rand(n,m)`: ($n \times m$)-Matrix mit Zufallswerten zwischen 0 und 1
- `gallery`: weitere spezielle Matrizen (siehe `help gallery`)
- `magic(n)`: magisches Quadrat mit Einträgen 1 bis n^2

Beispiele:

```
>> eye(4,5)
ans =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0

>> zeros(3)
ans =
     0     0     0
     0     0     0
     0     0     0

>> rand(4)
ans =
    0.5271    0.5376    0.8886    0.3942
    0.4848    0.3863    0.7693    0.9034
    0.2623    0.7750    0.8788    0.3710
    0.0994    0.7070    0.7210    0.1896

>> magic(4)
ans =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

1.4.3 Spezielle Vektoren

- `a:s:b`: Vektor `[a,a+s,a+2*s,...,a+m*s]` mit `m=floor((b-a)/s)`. `a:b` entspricht `a:1:b`
  als Funktion: `colon(a,s,b)` bzw. `colon(a,b)`
- `linspace(a,b,n)`: lineare Unterteilung des Intervalls `[a,b]` in `n` Punkte
  `linspace(a,b)` entspricht `linspace(a,b,100)`

- `logspace(a,b,n)`: `n` Punkte, die das Intervall `[10^a,10^b]` logarithmisch unterteilen `logspace(a,b)` entspricht `logspace(a,b,50)`

Beispiele:

```
>> 2.3:6.7
ans =
    2.3000    3.3000    4.3000    5.3000    6.3000

>> 11:-2:1
ans =
    11     9     7     5     3     1

>> 10:-10
ans =
    []

>> linspace(1,10,5)
ans =
    1.0000    3.2500    5.5000    7.7500   10.0000

>> logspace(0,1,5)
ans =
    1.0000    1.7783    3.1623    5.6234   10.0000
```

1.4.4 Blockmatrizen

- Matrizen können wiederum aus Matrizen aufgebaut werden
- Zeilen- und Spaltenzahlen müssen zueinander passen

Beispiel:

```
>> A11=[1 1 1
        1 1 1];

>> A12=[2 2
        2 2];

>> A21=[3 3
        3 3
        3 3];

>> A22=[4 4 4
        4 4 4
        4 4 4];

>> [[A11 A12;A21 A22],[5;5;5;5;5]]
ans =
     1     1     1     2     2     5
```

```
    1     1     1     2     2     5
    3     3     4     4     4     5
    3     3     4     4     4     5
    3     3     4     4     4     5
```

1.4.5 Indizierung

- Durch Angabe von Indexvektoren als Argumente der Variablen
  Erster Index 1, letzter Index `end`. Beispiel: `A(2,end)`
- Negativer Index erzeugt Fehler
- Positiver rationaler Index erzeugt Warnung und wird gerundet
- `A(z,s)` erzeugt Teilmatrix der durch `z` und `s` indizierten Zeilen und Spalten. Reihenfolge entsprechend der Indexvektor-Einträge
- `:` ist Kurzform für `1:end`
- Eindimensionale Indizierung über Spalten der Matrix möglich
- Teilmatrizen können Werte zugewiesen werden
- Zuweisung zu einem Element mit Index größer als Matrixgröße bewirkt Auffüllung mit 0
- Löschen von ganzen Zeilen und Spalten durch Zuweisung von `[ ]`

Beispiele:

```
>> A=[11 12 13 14 15 16 17 18 19
      21 22 23 24 25 26 27 28 29];

>> A(2,4)
ans =
    24

>> A(1,[7 5 3])
ans =
    17    15    13

>> A([7 1 end])
ans =
    14    11    29

>> A(:,2:2:end)
ans =
    12    14    16    18
    22    24    26    28

>> A(:,5:end)=[]
A =
    11    12    13    14
```

```
    21    22    23    24

>> A(1:3,2:3)=ones(3,2)
A =
    11     1     1    14
    21     1     1    24
     0     1     1     0
```

1.4.6 Matrixumwandlung

- `fliplr(A), flipud(A), rot90(A)`: Spiegelung, Drehung
- `A.', A'`: transponiert, komplex konjugiert transponiert
  als Funktion: `transpose(A), ctranspose(A)`
- `tril(A,k), triu(A,k)`: untere bzw. obere Dreiecksmatrix ab `k`-ter Nebendiagonalen
- `diag(A,k)`: `k`-te Nebendiagonale von `A`, bzw. Matrix mit `A` als `k`-ter Nebendiagonale
- `A(:)`: Vektor in dem die Spalten von `A` nacheinander stehen
- `reshape(A,zn,sn)`: wandelt `A` in eine `zn` $\times$ `sn` Matrix um, indem von `A(:)` der Reihe nach neue Spalten der Länge `zn` abgegriffen werden
- `repmat(A,zf,sf)`: `zf` $\times$ `sf` Blockmatrix aus `A`

Beispiele:

```
>> A=rand(2,4)
A =
    0.7037    0.9329    0.2280    0.1722
    0.5221    0.7134    0.4496    0.9688

>> fliplr(A)
ans =
    0.1722    0.2280    0.9329    0.7037
    0.9688    0.4496    0.7134    0.5221

>> rot90(A)
ans =
    0.1722    0.9688
    0.2280    0.4496
    0.9329    0.7134
    0.7037    0.5221

>> reshape(A,4,2)
ans =
    0.7037    0.2280
    0.5221    0.4496
    0.9329    0.1722
    0.7134    0.9688
```

```
>> A=rand(3)
A =
    0.7699    0.0466    0.2888
    0.3751    0.5979    0.8888
    0.8234    0.9492    0.1016

>> D=diag(A,1)
D =
    0.0466
    0.8888

>> diag(D,-1)
ans =
         0         0         0
    0.0466         0         0
         0    0.8888         0

>> triu(A,1)
ans =
         0    0.0466    0.2888
         0         0    0.8888
         0         0         0

>> tril(A,-2)
ans =
         0         0         0
         0         0         0
    0.8234         0         0
```

1.4.7 Größe von Matrizen

- `size(A)`: Vektor mit der Zeilen- und Spaltenzahl der Matrix `A`
- `size(A,d)`: Ausdehnung in Richtung `d`; `d`=1: Zeilen, `d`=2: Spalten
- `[z,s]=size(A)`: Zeilenzahl in `z` und Spaltenzahl in `s`
  Anmerkung: Mehrere Rückgabewerte werden den in `[ ]` aufgeführten Variablen zugewiesen
- `length(A)`: gibt `max(size(A))` bzw. 0 bei leeren Matrizen zurück

Beispiele:

```
>> A=ones(2,8);

>> size(A)                >> size(A,1)        >> length(A)
ans =                     ans =               ans =
     2     8                   2                   8
```

```
>> [z,s]=size(A)
z =
     2
s =
     8
```

1.4.8 Matrix-Operatoren

- Element-Operatoren: `.+`, `.-`, `.*`, `./`, `.\`, `.^`
  Element des ersten Operanden wird mit entsprechendem Element des zweiten Operanden verknüpft. Ist ein Operand ein Skalar, so wird dieser mit allen Elementen des anderen Operanden verknüpft
- Element(ar)-Funktionen wirken auf jedes Element einer Matrix
- `A+B`: Matrixaddition, entspricht `A.+B`
- `A-B`: Matrixsubtraktion, entspricht `A.-B`
- `A*B`: Matrixmultiplikation
- `A^k`: `k`-fache Matrixmultiplikation (falls `k` natürlich).
- `X=A\B`: Lösung von `AX=B`
- `X=B/A`: Lösung von `XA=B`

Ist ein Operand skalar, werden die Element-Operationen verwendet
Bei den Lösungen werden ggf. Näherungen berechnet
Beispiele:

```
>> A=20.*rand(3)-10
A =
   -5.9471   -9.6072    6.6359
    3.4427    3.6255    0.0563
    6.7624   -2.4104    4.1894

>> D=reshape(1:9,3,3)'; D.^2
ans =
     1     4     9
    16    25    36
    49    64    81

>> A=rand(3); B=eye(3); C=A\B
C =
    3.3394    0.2250   -2.6405
   -1.4273    1.5252    0.7781
   -0.2883   -0.7228    1.8129

>> C-A^-1
ans =
     0    2.7760e-16          0
```

```
     0              0         0
     0              0         0

>> A*C-B
ans =
            0   5.3939e-18  -1.5607e-16
   7.4569e-17  -1.1102e-16  -1.6220e-16
   2.6902e-17  -9.7849e-17  -1.1102e-16
```

1.4.9 Vektor-Funktionen

- `norm(V,p)`: p-Norm des Vektors `V`
- Skalar: `min`, `max`, `sum`, `prod`, `mean`, `median`
- Vektor gleicher Länge: `sort`, `cumsum`, `cumprod`
- Vektor anderer Länge: `diff`, `find`, `unique`
- Weitere Funktionen siehe `help datafun`

Beispiele:

```
>> V=fix(100*rand(1,10))
V =
    71    89    27    25    86    23    80    90    23    23

>> median(V)
ans =
    49

>> [S,I]=sort(V)
S =
    23    23    23    25    27    71    80    86    89    90
I =
     6     9    10     4     3     1     7     5     2     8

>> F=cumprod(1:8)
F =
  1       2       6      24     120     720    5040   40320
```

1.4.10 Matrix-Funktionen

- `norm(M)`: betragsmäßig größter Singulärwert
- Skalar: `rank`, `det`, `trace`, `cond`
- Vektor: `poly`, `eig`
- Matrix: `inv`, `pinv`, `null`
- Mehrere Matrizen: `eig`, `lu`, `qr`, `svd`

- Weitere Funktionen: siehe `help matfun`

Beispiele:

```
>> A=[2 1 1;1 1 0;1 0 1];

>> rank(A)
ans =
     2

>> [EV,EW]=eig(A)
EV =
   -0.5774   -0.0000    0.8165
    0.5774   -0.7071    0.4082
    0.5774    0.7071    0.4082
EW =
    0.0000         0         0
         0    1.0000         0
         0         0    3.0000

>> inv(A);
Warning: Matrix is singular to working precision.

>> B=pinv(A);

>> norm(B*A*B-B)
ans =
   1.4583e-16

>> norm(A*B*A-A)
ans =
   2.7628e-16
```

1.4.11 Ausgabeformate

`format short`:	Festpunktdarstellung mit 5 Stellen
`format short e`:	Fließpunktdarstellung mit 5 Stellen
`format long`:	Festpunktdarstellung mit 15 Stellen
`format long e`:	Fließpunktdarstellung mit 15 Stellen
`format rat`:	Darstellung mit Brüchen
`format +`:	Nur Vorzeichen des Realteils
`format compact`:	Keine zusätzlichen Leerzeilen
`format loose`:	Zusätzliche Leerzeilen
`format`:	Entspricht `format loose` und `format short`

Weitere Formate siehe `help format`.
Beispiele:

```
>> format compact, format long
```

```
>> A=[20*rand(2,3)-7 ; 0,Inf,NaN]
A =
  -3.58414372516640   1.79581713892760  -0.71565378911943
  12.88590981027840  -0.19904102288619   0.30156773857339
                  0                Inf                NaN

>> format short e; A
A =
  -3.5841e+00   1.7958e+00  -7.1565e-01
   1.2886e+01  -1.9904e-01   3.0157e-01
            0          Inf          NaN

>> format rat; A
A =
 -2577/719     1803/1004   -1943/2715
  1920/149    -1868/9385     981/3253
      0           1/0          0/0

>> format +; A
A =
-+-
+-+
 +
```

Kapitel 2

Datenstrukturen

2.1 Datentypen

2.1.1 Dünn besetzte Matrizen (Sparse)

- Gespeichert werden lediglich von Null verschiedene Elemente im Listenformat (Werte, Zeilen- und Spaltenindizes)
- Platzersparnis falls weniger als 2/3 der Elemente nicht Null sind
- Verknüpfungen mit vollbesetzten Matrizen möglich
- Erzeugen: `sparse`, `spalloc`, `spones`, `speye`, `sprand`, `spconvert`
- Konvertieren: `sparse`, `full`
- Struktur: `spy`, `nnz`, `nonzeros`
- Lineare Algebra: `eigs`, `svds`, `normest`, `condest`
- Iterative Löser: `minres`, `pcg`
- Sonstige Funktionen: `help sparfun`

Beispiele:

```
>> A=200*sprand(100,100,1/1000);

>> A=A-100*spones(A)
A =
  (73,24)   -71.32470045062594
  (89,35)   -32.73209687203639
  (40,52)   -37.57709809498553
  (28,53)    80.03796441968240
  (46,62)    40.77664420681884
  (88,69)   -90.19628365353900
  (26,81)    -1.73698501631982
  (81,90)    65.00643237474648
  (74,91)    -2.92984630688086
  (78,93)   -98.62310404027863
```

```
>> normest(A)
ans =
  98.62289050554142

>> norm(full(A))
ans =
  98.62310404027863

>> clear all
>> S=sprand(1000,1000,1/100);
>> F=full(S);
>> P=S*F;
>> X=S\sprand(1000,1,1/10);
>> Z=exp(S);
>> whos
  Name      Size         Bytes  Class

  F      1000x1000     8000000  double array
  P      1000x1000     8000000  double array
  S      1000x1000      123464  sparse array
  X      1000x1          12008  sparse array
  Z      1000x1000    12004004  sparse array

Grand total is 3010955 elements using 28139476 bytes
```

2.1.2 Mehrdimensionale Felder (ND-Arrays)

- Erzeugen: `zeros`, `ones`, `reshape`, `rand`
- Ausgabe: Matrizen in den ersten 2 Dimensionen, hintereinander für die restlichen Dimensionen
- Indizieren: Mit entsprechend vielen Indexvektoren
- Letzte Dimension hat Ausdehnung > 1 (falls > 2 Dimensionen)
- Operatoren: Nur Element-Operationen möglich
- Funktionen: Elementar-Funktionen,
  Vektor-Funktionen mit Angabe der Arbeits-Dimension (z.B. `sum`)
  Sonstige Funktionen sofern sinnvoll (z.B. `size`)
- Spezielle Funktionen:
  `cat`, `ndims`, `permute`, `ipermute`, `shiftdim`, `squeeze`

Beispiele:

```
>> A=rand(2,4,3,1,1,1,1)
A(:,:,1) =
    0.9994    0.0589    0.5485    0.5973
```

```
    0.9616    0.3603    0.2618    0.0493
A(:,:,2) =
    0.5711    0.9623    0.7400    0.6343
    0.7009    0.7505    0.4319    0.8030
A(:,:,3) =
    0.0839    0.9159    0.2536    0.5134
    0.9455    0.6020    0.8735    0.7327

>> size(A)
ans =
     2     4     3

>> A=ones(1,2,3,4,5);
>> size(shiftdim(A,2))
ans =
     3     4     5     1     2

>> size(shiftdim(A,-2))
ans =
     1     1     1     2     3     4     5

>> size(A(1,1,1,:,:))
ans =
     1     1     1     4     5

>> size(A(:,:,1,1,1))
ans =
     1     2

>> size(squeeze(A(1,1,1,:,:)))
ans =
     4     5

>> size(A(1,:))
ans =
     1   120
```

2.1.3 Logische Werte

- Falsch: 0, wahr: 1 (bzw. nicht 0)
- Erzeugen: `<`, `<=`, `==`, `>=`, `>`, `~=`, `logical`
- Operatoren: `&`, `|`, `~`, `xor(A,B)`
- Vektor-Funktionen: `any`, `all`
- Sonstige Funktionen:
  `find`, `isempty`, `isnan`, `isinf`, `isreal`, `exist`
- Zum Indizieren verwendbar

Beispiele:

```
>> A=round(200*rand(3,5))-100
A =
    21    47   -77    18    37
    81   -18    14    -1    59
    21   -13    94   -59    32

>> L1=A>60
L1 =
     0     0     0     0     0
     1     0     0     0     0
     0     0     1     0     0

>> L2=A<-20
L2 =
     0     0     1     0     0
     0     0     0     0     0
     0     0     0     1     0

>> L3=L1|L2
L3 =
     0     0     1     0     0
     1     0     0     0     0
     0     0     1     1     0

>> A(L3)=0
A =
    21    47     0    18    37
     0   -18    14    -1    59
    21   -13     0     0    32

>> any(A<0)
ans =
     0     1     0     1     0

>> find(~A)'
ans =
     2     7     9    12
```

2.1.4 Zeichenketten

- Matrizen aus Zeichencodes (ganze Zahlen im Intervall [0,65535])
- Eingabe: in `' '`; Das Zeichen `'` wird durch `''` erzeugt
- Umwandlungen: `char`, `double`
- Funktionen: `strcat`, `strvcat`, `str2mat`, `strcmp`, `upper`, `lower`, `eval`
- Logische Abfragen: `ischar`, `isletter`, `isspace`

- Weitere Funktionen: `help strfun`

Beispiele:

```
>> A='Hallo'; B='Welt'; C=[A,' ',B]
C =
Hallo Welt

>> ascii = char(reshape(32:127,48,2)')
ascii =
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

>> M=str2mat('Hallo','','Welt')
M =
Hallo

Welt

>> double(M(:,1:2:end))
ans =
    72   108   111
    32    32    32
    87   108    32

>> upper(M)
ans =
HALLO

WELT

>> lower(M)
ans =
hallo

welt

>> sum(isletter(M(:)))
ans =
     9

>> befehl='J=gallery(''jordbloc'',3)'
befehl =
J=gallery('jordbloc',3)

>> eval(befehl)
J =
     1     1     0
     0     1     1
     0     0     1
```

2.1.5 Struct-Variablen

- Zusammenfassung mehrerer Variablen unter einem Sammelnamen
- Form: Variablenname.Feldname
- Arrays von Strukturen mit gleichen Feldnamen möglich
- Feldvariablen können jedem Datentyp angehören (auch struct)
- Erzeugung: Feldzuweisungen, `struct`
- Zugriff:
  - Gesamter Inhalt: Variablenname
  - Ein Feld: Variablenname.Feldname
  - Bei Arrays steht Indizierung direkt hinter dem Variablennamen
- Existierende Felder: `fieldnames`
- Bearbeiten: `isfield`, `setfield`, `getfield`, `rmfield`

Beispiele:

```
>> Studierende=struct('name','Berta Beispiel','noten',[1.3 2 1.7])
Studierende =
     name: 'Berta Beispiel'
    noten: [1.3000 2 1.7000]

>> Studierende(3).name='Emil Exempel';
>> Studierende(3).noten='drei';

>> Studierende
Studierende =
1x3 struct array with fields:
    name
    noten

>> Studierende.noten
ans =
      1.3000   2.0000   1.7000
ans =
      []
ans =
drei

>> Studierende(1).noten(3)
ans =
    1.7000
```

2.1.6 Cell-Arrays

- Arrays bei denen jeder Eintrag ein beliebiger Datentyp sein kann (auch Cell-Arrays oder Struct-Variablen)
- Erzeugen: { }, `cell`, `num2cell`, `struct2cell`
- Indizieren: `( )`
- Inhalt auslesen: { }
- Teilfelder füllen, auslesen: `deal`
- Informationen über Inhalte: `cellfun`
- Häufige Verwendung:
  Speicherung von Zeichenketten unterschiedlicher Länge

Beispiele:

```
>> C=cell(2,3);
>> C{3}=rand(2); C{1,4}='Hallo'; C{2,4}=struct('a',1,'b',2)
C =
     []    [2x2 double]     []    'Hallo'
     []              []     []    [1x1 struct]

>> C(1:2,2:4)
ans =
    [2x2 double]     []    'Hallo'
              []     []    [1x1 struct]

>> C{1,2}
ans =
    0.9806    0.6831
    0.6668    0.8754

>> C(2,1)
ans =
    {[]}

>> L=cellfun('isempty',C)
L =
     1     0     1     0
     1     1     1     0

>> [M1,M2,M3,M4,M5,M6]=deal(C{:,2:4})
M1 =
    0.9806    0.6831
    0.6668    0.8754
M2 =
     []
M3 =
```

```
     []
M4 =
     []
M5 =
Hallo
M6 =
    a: 1
    b: 2
```

2.1.7 Polynome

- Koeffizientenvektor: $[a_n, a_{n-1}, \ldots, a_0]$
- Erzeugen: `poly`, `polyfit`
- Bearbeiten: `conv`, `deconv`, `polyint`, `polyder`
- Faktorisierung: `roots`, `residue`
- Auswerten: `polyval`, `polyvalm`
- Stückweise Polynome:
  - Vektor mit Schnittstellen und
    Matrix mit Koeffizienten der Polynomstücke
  - Erzeugen: `mkpp`, `pchip`, `spline`
  - Auswerten: `ppval`
  - Vektorwertig möglich
- Sonstige Funktionen: `help polyfun`

Beispiel zu Polynomen:

```
>> P=poly([3,4,5])
P =
     1   -12    47   -60

>> [Q,R]=deconv(P,[1,-4])
Q =
     1    -8    15
R =
     0     0     0     0

>> P2=conv(Q,[1,-7])
P2 =
     1   -15    71  -105

>> roots(P2)'
ans =
    7.0000    5.0000    3.0000
```

```
>> polyval(P2,3:7)
ans =
     0    3    0   -3    0
```

Beispiel zu stückweisen Polynomen:

```
>> PP=spline([0,1,2,3],[0,1,4,9])
PP =
      form: 'pp'
    breaks: [0 1 2 3]
     coefs: [3x4 double]
    pieces: 3
     order: 4
       dim: 1

>> x=linspace(0,3);
>> y=ppval(PP,x);
>> max(abs(y-x.^2))
ans =
   1.7764e-15

>> PP=spline([0,1,2,3],[0,1,16,81]);
>> y=ppval(PP,x);
>> max(abs(y-x.^4))
ans =
    0.9993
```

2.2 Ein- und Ausgabe von Daten

2.2.1 Übersicht

- Protokolldatei: `diary`
- Variablen speichern/lesen: `save, load`
- Dateiverwaltung: `fopen`, `fclose`
- Dateizugriff: `fwrite`, `fprintf`, `fread`, `fscanf`
- Weitere Funktionen: `help iofun`

2.2.2 Protokolldatei

- Alle Ein- und Ausgaben, die im Kommandofenster erscheinen, können in einer Text-Datei im Arbeitsverzeichnis mitprotokolliert werden
- Name der Protokolldatei setzen mit `diary` <Dateiname>. Standardname ist `diary`
- Protokollierung starten: `diary on`
- Protokollierung beenden: `diary off`
- Protokollstatus wechseln: `diary`

2.2.3 Speichern und Laden von Variableninhalten

- Variablen im MATLAB-Format speichern bzw. laden
- Speichern: `save` <Dateiname> <Variablenliste>
  - Dateiname wird ggf. um Endung `.mat` erweitert
  - `save` ohne Parameter speichert alle im Kommandofenster definierten Variablen in der Datei `matlab.mat`
  - Mit zusätzlicher Option `-ascii` wird eine Textdatei erzeugt
    (Wird in dieser Datei mehr als eine Variable gespeichert, kann sie nicht mehr geladen werden. Die Speicherung von Struct- und Cell-Variablen im ASCII-Format ist nicht möglich)
- Laden: `load` <Dateiname> <Variablenliste>
  - `load` ohne Parameter lädt alle Variablen aus `matlab.mat`
  - Wird eine ASCII-Datei geladen, entspricht der Variablenname dem Dateinamen. Ladeoption `-ascii` erforderlich
- Weitere Optionen: `help save`

Beispiele:

```
>> X=rand(4); Y=ones(3); whos
  Name      Size         Bytes  Class
  X         4x4            128  double array
  Y         3x3             72  double array
Grand total is 25 elements using 200 bytes

>> save
Saving to: matlab.mat
>> clear
>> load matlab Y
>> who
Your variables are:
Y

>> save matrix Y -ascii
>> clear
>> load matrix -ascii
>> who
Your variables are:
matrix
```

2.2.4 Dateiverwaltung

- Öffnen einer Datei: `FID=fopen(`<Dateiname>,<Modus>`)`
- Zugriff: über den File-Identifier `FID`

- Modus: Zeichenkette, welche die Art des Dateizugriffs angibt
  - `'r'`: lesen (read)
  - `'w'`: schreiben, ggf. erzeugen (write)
  - `'a'`: anhängen, ggf. erzeugen (append)
  - weitere Modi: `help fopen`
- Schließen einer Datei: `fclose(FID)`

2.2.5 Schreiben von Dateien

- Binäres schreiben: `fwrite(FID,`<Variable>,<Format>`)`
  - Format: Zeichenkette, die das Binärformat angibt
    z.B.: `int8`, `int16`, `int32`, `float32`, `float64`
  - Binär-Formate sind maschinenabhängig und daher nicht zum Datentransport zwischen unterschiedlichen Systemen geeignet
- Formatiertes schreiben: `fprintf(FID,`<Format>,<Variablen>`)`
  - Format: Zeichenkette mit Platzhaltern für die Variablen und Sonderzeichen
  - Platzhalter: %-Zeichen gefolgt von Steuerzeichen und Ausgabetyp
  - Steuerzeichen: `-,+,#,0, <Zahl>.<Zahl>`
  - Ausgabetypen: `c,d,e,f,g,i,o,s,x,E,G,X`
  - Sonderzeichen: `\n,\r,\t,\b,\\,%%`
  - Bei `FID=1` bzw. fehlendem `FID`: Ausgabe im Kommandofenster
  - Format wird wiederholt, bis alle Elemente/Variablen abgearbeitet sind

`fprintf`-Steuerzeichen

`+`:	Mit Vorzeichen
`-`:	Linksbündig
`n.m`:	Ausgabe mit Mindestbreite von n Zeichen mit m Nachkommastellen bei `e,f,g` auf m Stellen mit führenden Nullen aufgefüllt bei `d,i`
`0`:	Mit führenden Nullen bis zur Feldbreite aufgefüllt
`#`:	Alternative Darstellung: mit 0x bzw. 0X bei `x` mit führender 0 bei `o` immer mit Dezimalpunkt bei `e,f,g` mit abschließenden Nullen bei `g`

`fprintf`-Ausgabetypen

`f`:	Fließkommadarstellung
`e,E`:	Exponentialdarstellung
`g,G`:	Mischung aus f und e je nach Größe der Zahl
`c`:	Einzelnes Zeichen
`s`:	Zeichenkette
`d,i`:	Dezimaldarstellung
`o`:	Octaldarstellung
`x,X`:	Hexadezimaldarstellung

`fprintf`-Sonderzeichen

`\n`:	Zeilenschaltung. Notwendig, falls die Eingabeaufforderung in einer neuen Zeile erscheinen soll
`\r`:	Wagenrücklauf. Z.B. um während eines Programmablaufs Fortschrittsmeldungen übereinander zu schreiben
`\t`:	Tabulatorschaltung
`\b`:	Vorhergehendes Zeichen löschen
`\\`:	Erzeugt \
`%%`:	Erzeugt %

Beispiele:

```
>> x=3.5;
>> fprintf('Inhalt der Variable x: %f\n', x)
Inhalt der Variable x: 3.500000

>> fprintf('Inhalt der Variable x: %+015.2E\n', x)
Inhalt der Variable x: +0000003.50E+00

>> X=[1,2;3,4];
>> fprintf('Die Elemente von X sind: %d %d %d %d\n', X)
Die Elemente von X sind: 1 3 2 4

>> fprintf('Die Elemente von X sind: %d %d\n', X', x)
Die Elemente von X sind: 1 2
Die Elemente von X sind: 3 4
Die Elemente von X sind: 3.500000e+00 >>

>> X=1:10;
>> fprintf('Elemente von X:'); fprintf(' %d ',X); fprintf('\n');
Elemente von X: 1  2  3  4  5  6  7  8  9  10
```

2.2.6 Dateien lesen

- Binäres lesen: `fread(FID,`<Variable>,<Format>`)`

- Formatiertes lesen: `fscanf(FID,`<Format>,<Num>`)`
  - Format: Zeichenkette wie bei `fprintf`

  - Num: Anzahl der zu lesenden Werte. `Inf` für alle

- Abfrage auf Dateiende:
  `feof(FID)`. Rückgabewert 1 bei Dateiende, sonst 0

Kapitel 3

Grafik

3.1 2D-Grafiken

3.1.1 Übersicht

- Polygonzüge: `plot`
  Mit anderen Achsen: `loglog`, `semilogx`, `semilogy`
- Polarkoordinaten: `polar`
- Flächen: `area`, `fill`
- Funktionen: `fplot`, `ezplot`
- Treppenfunktionen: `stairs`
- Diagramme: `bar`, `hist`, `pie`
- Höhenlinien: `contour`, `contourf`
- Vektorfelder: `quiver`, `streamline`
  Hilfsfunktionen: `gradient`, `stream2`
- Weitere Funktionen: `help graph2d, help specgraph`

3.1.2 plot

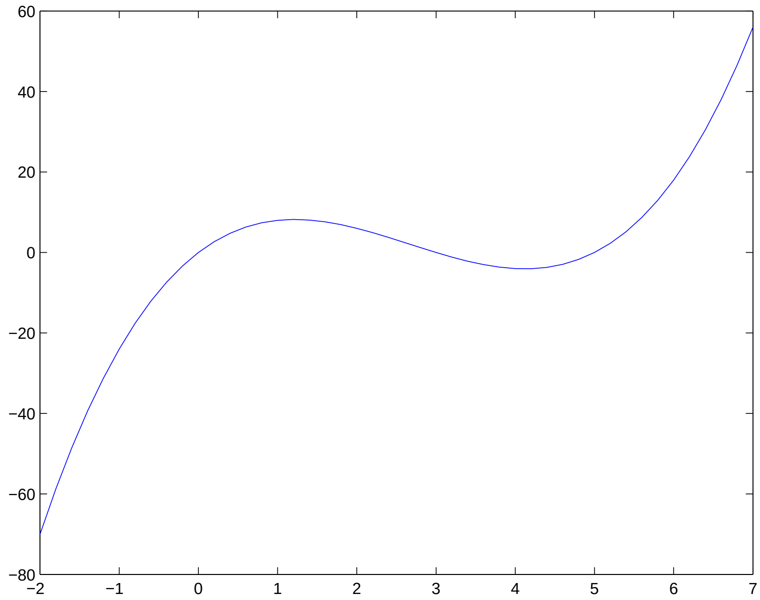
- `plot(X,Y)`: Polygonzug mit den Knoten (x_k, y_k). Imaginärteile werden ignoriert
- `plot(Y)` entspricht
  - `plot(1:length(Y),Y)` falls `Y` reell
  - `plot(real(Y),imag(Y))` falls `Y` komplex
- Bei Matrizen wird pro Spalte ein Polygonzug gezeichnet
- Zeichenkette aus Stilparametern als weiteres Argument möglich. Erlaubte Typkennzeichner (Bedeutung siehe `help plot`):
  - Punkttypen: `.,o,x,+,*,s,d,v,^,<,>,p,h`
  - Linientypen: `-,:,-.,--`

  - Farben: `y,m,c,r,g,b,w,k`

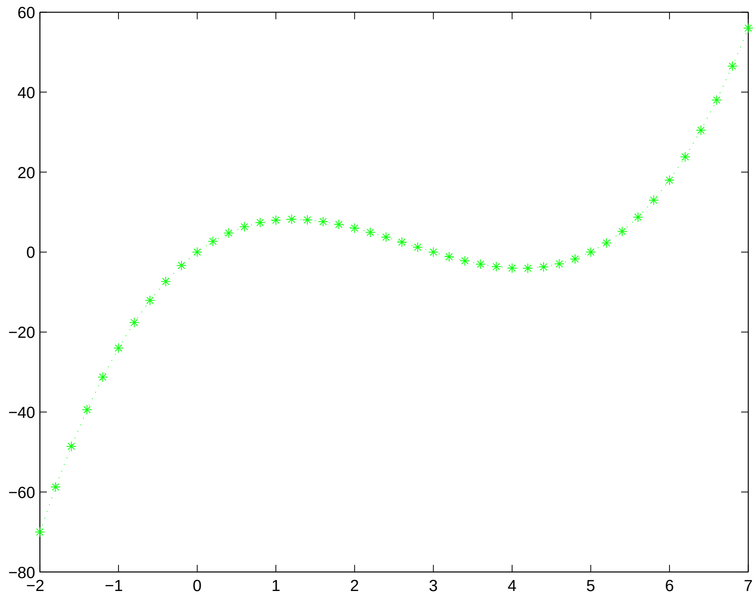
- Mehrere Linien mit unterschiedlichen Stilparametern:
  `plot(x1,y1,s1,x2,y2,s2,...)`

Beispiele:

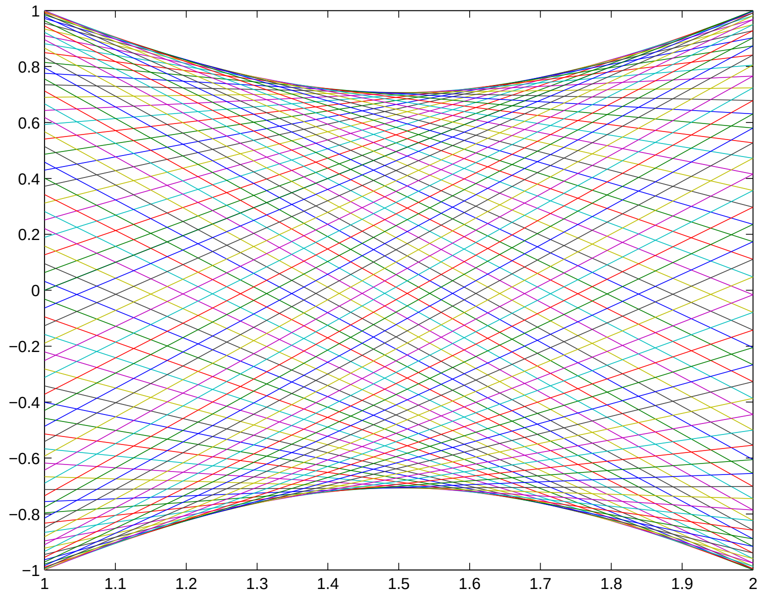
```
>> X=-2:.2:7;
>> P=[1 -8 15 0];
>> Y=polyval(P,X);
>> plot(X,Y)
```



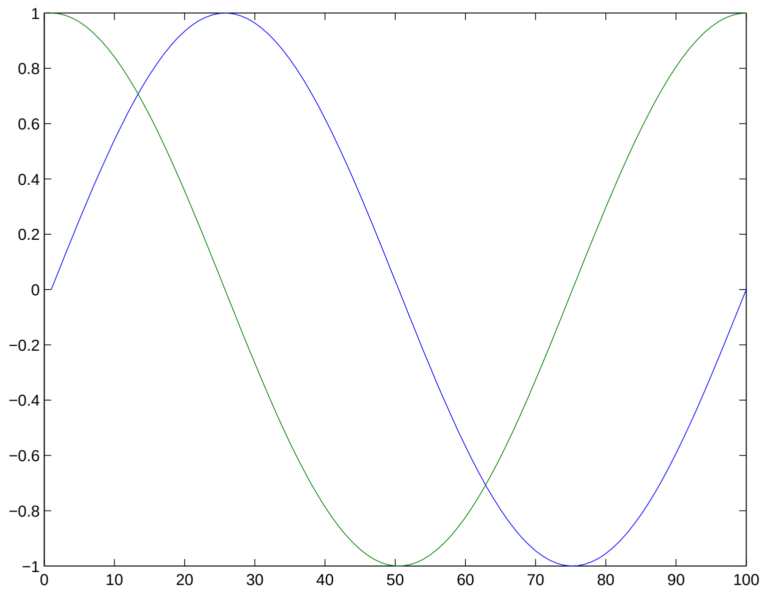

```
>> Z=X+i*Y;
>> plot(Z,'*g:')
```



```
>> X=linspace(0,2*pi);
>> Y=[sin(X);cos(X)];
>> plot(Y)
```

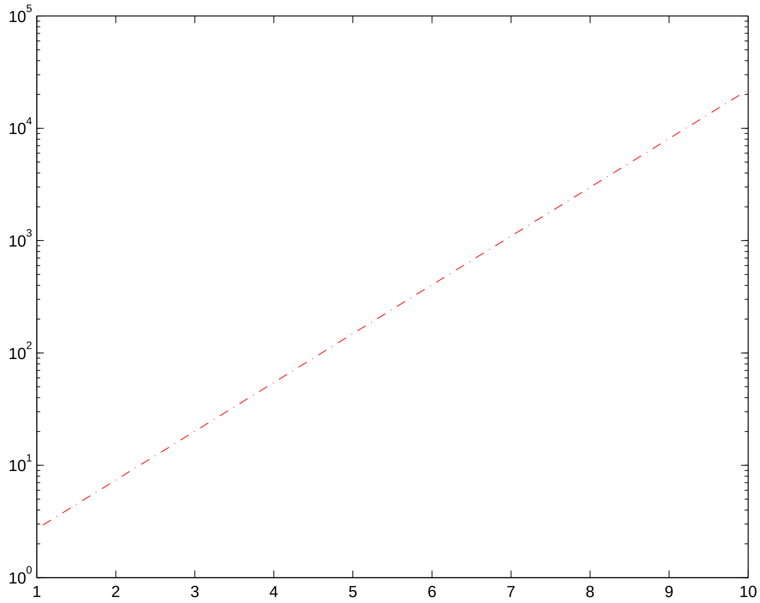



```
>> plot(Y')
```



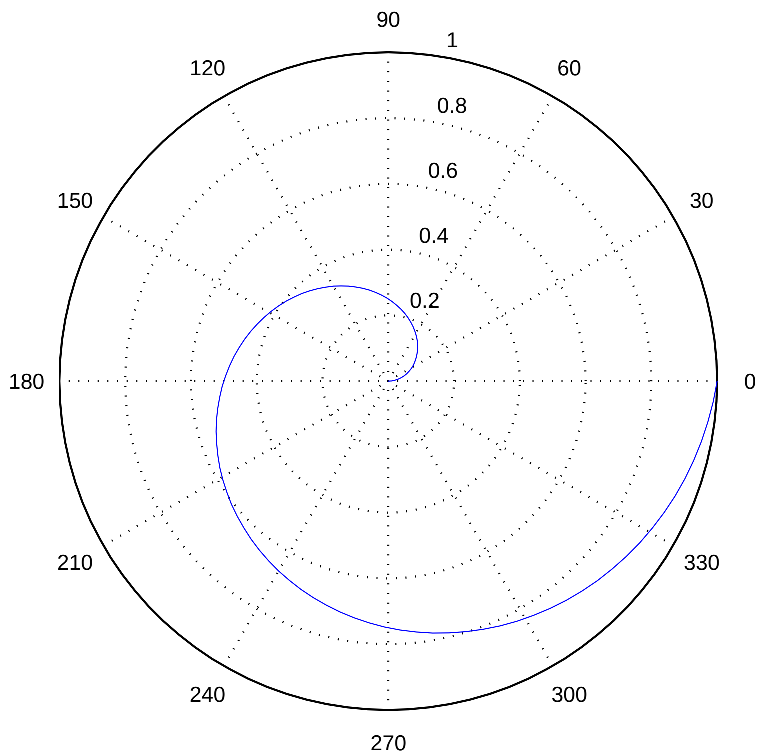
3.1.3 semilogy und polar

```
>> x=linspace(1,10);
>> y=exp(x);
>> semilogy(x,y,'r-.')
```



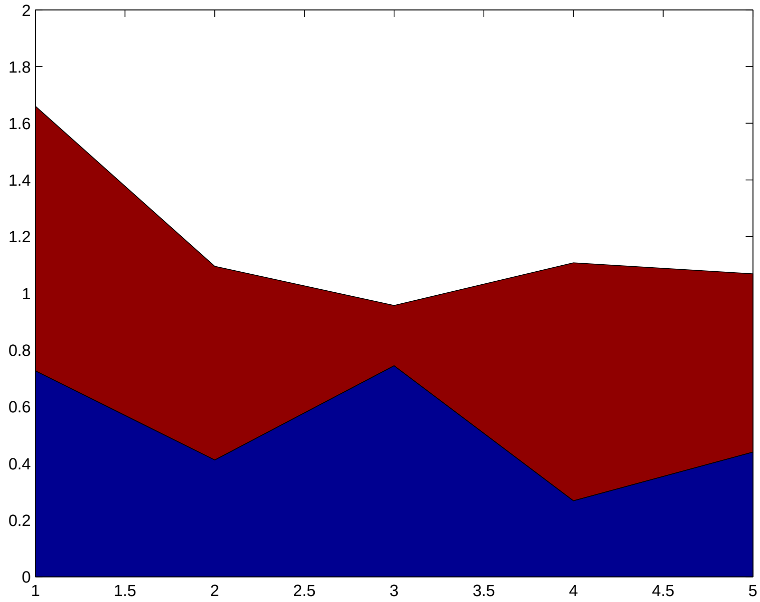
Beispiel:

```
>> phi=linspace(0,2*pi);
>> r=linspace(0,1);
>> polar(phi,r)
```



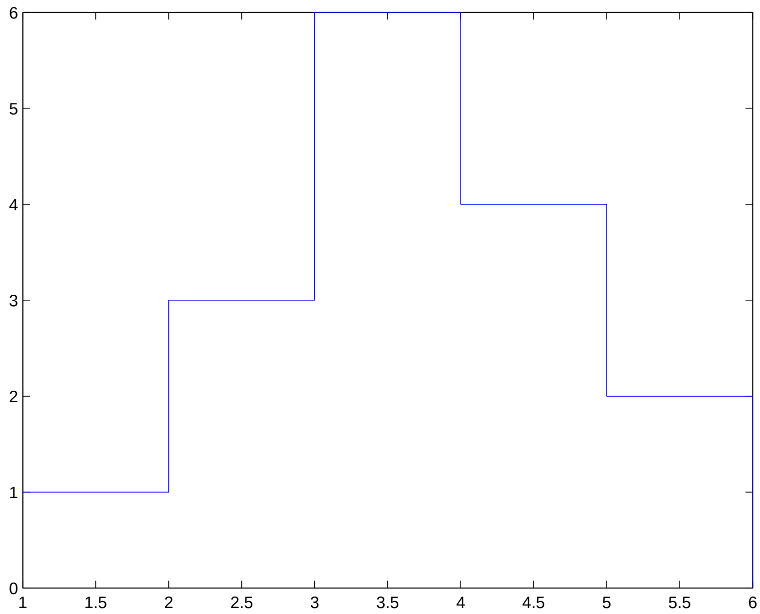
3.1.4 Diagramme

```
>> area(rand(5,2));
```



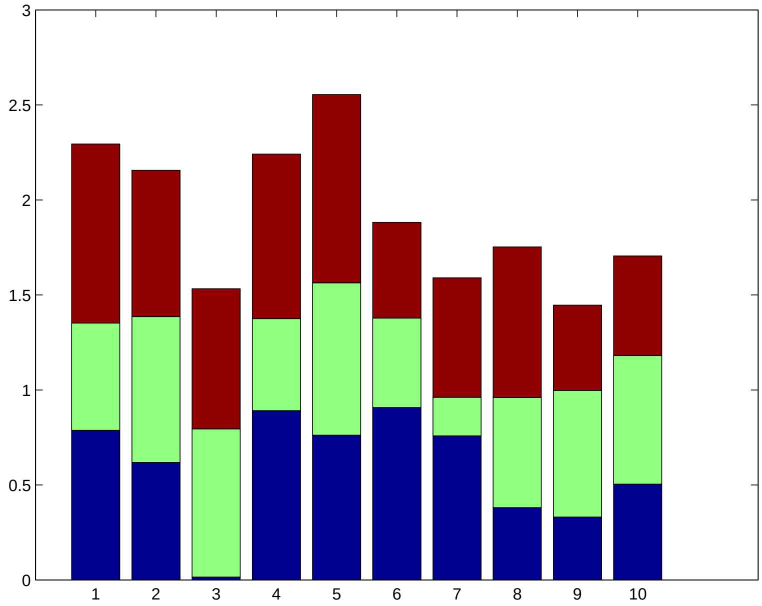
Beispiel:

```
>> stairs([1 3 6 4 2 0]);
```



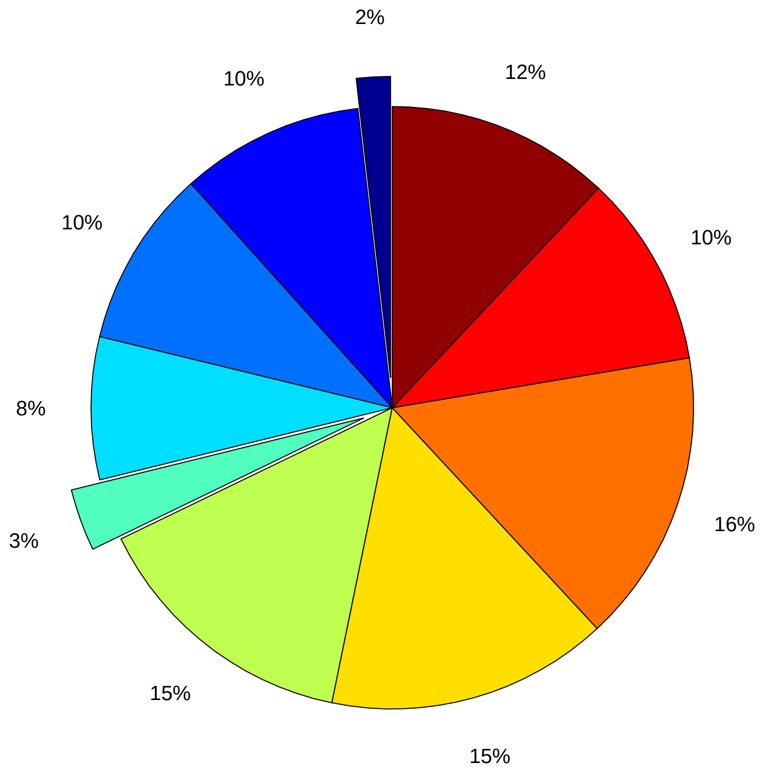
Beispiel:

```
>> bar(rand(10,3),'stacked')
```



Beispiel:

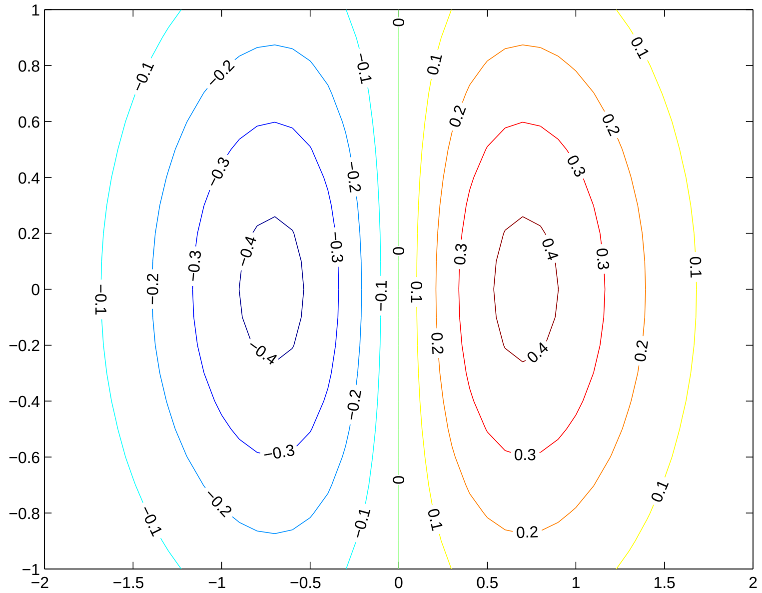
```
>> x=rand(1,15);
>> explode=x./sum(x)<.05;
>> pie(x,explode)
```



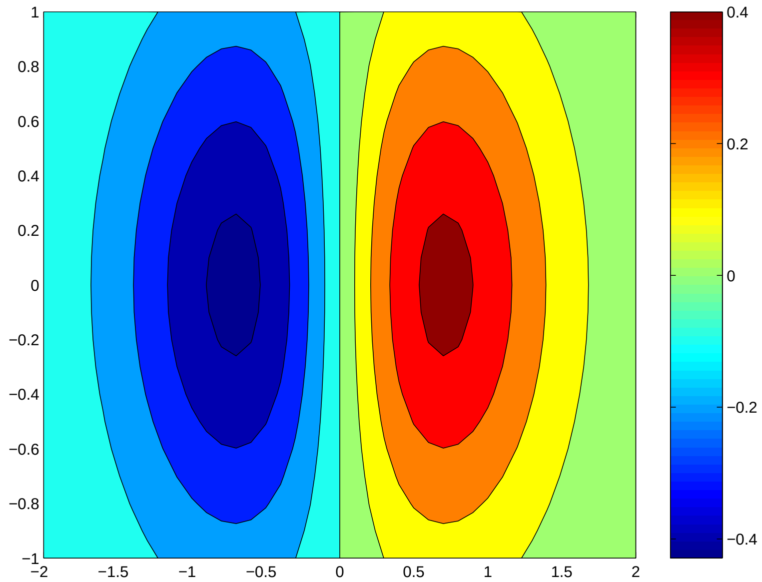
3.1.5 Höhenlinien

```
>> [X,Y]=meshgrid(-2:.1:2, -1:.1:1);
>> Z=X.*exp(-X.^2-Y.^2);
```

```
>> [C,H]=contour(X,Y,Z);
>> clabel(C,H);
```

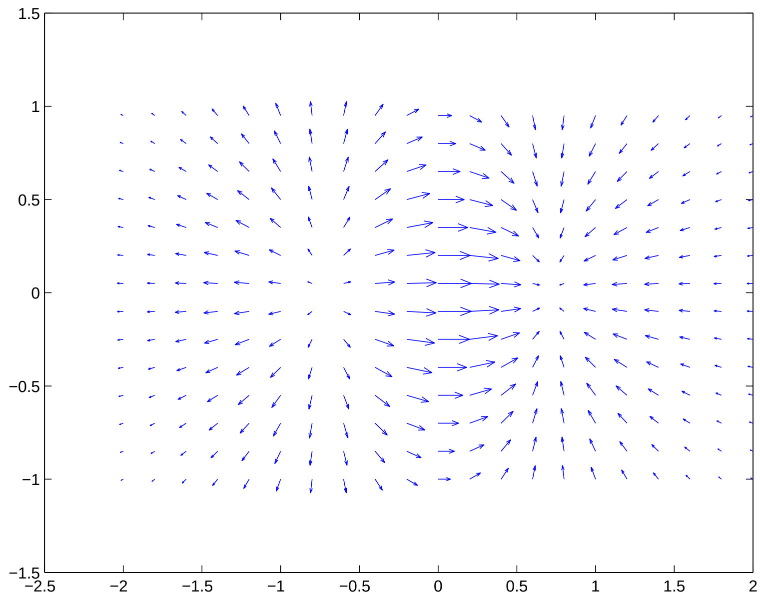



```
>> contourf(X,Y,Z);
>> colorbar;
```

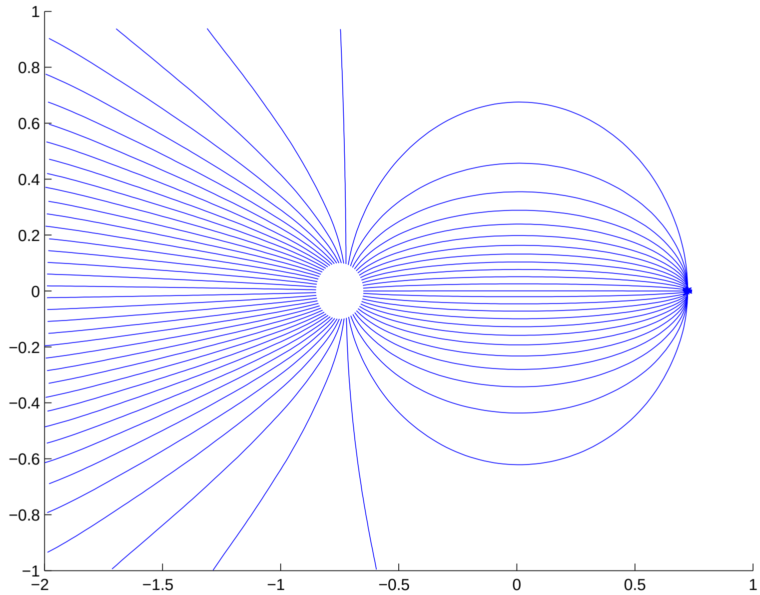


3.1.6 Vektorfelder

```
>> [X,Y]=meshgrid(-2:.2:2, -1:.15:1);
>> Z=X.*exp(-X.^2-Y.^2);
>> [DX,DY]=gradient(Z,.2,.15);
>> quiver(X,Y,DX,DY);
```




```
>> sx=-.75+.1*cos(0:.1:2*pi);
>> sy=     .1*sin(0:.1:2*pi);
>> streamline(X,Y,DX,DY,sx,sy);
```



3.2 3D-Grafiken

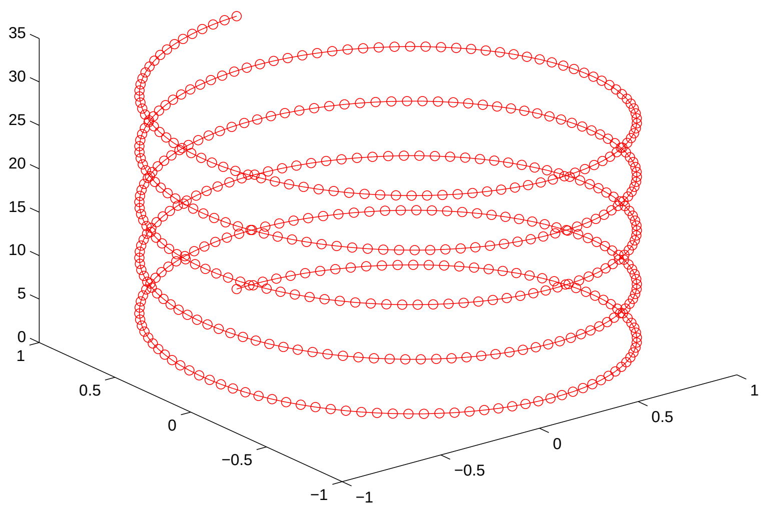
3.2.1 Übersicht

- Polygonzüge: `plot3`
- Polygonale Flächen: `fill3`
- Oberflächen: `mesh`, `surf`, `surfl`
  Parametergebiete generieren: `meshgrid`, `ndgrid`
  Färbung einstellen: `shading`, `colormap`

- Diagramme: `pie3`, `bar3`
- Höhenlinien: `contour`, `contourf`, `contour3`
- Vektorfelder: `quiver3`, `streamline`
- Schnittbilder: `slice`, `countourslice`
- Weitere Funktionen: `help graph3d`, `help specgraph`

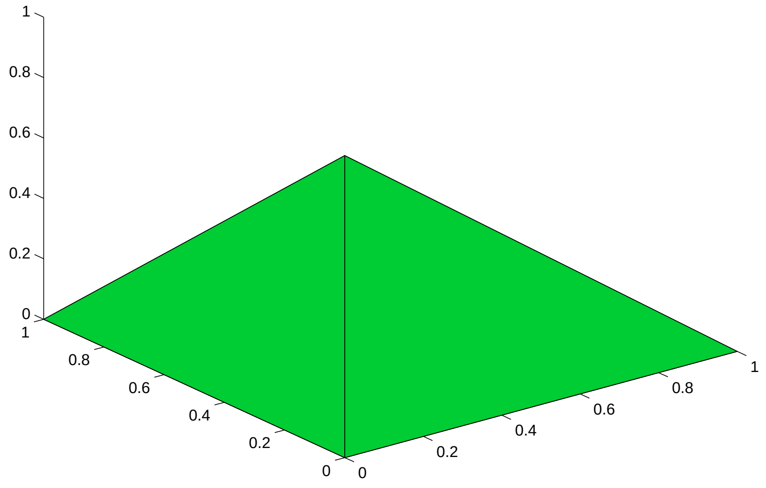
3.2.2 Polygonzüge

```
>> t=linspace(0,10*pi,500);
>> plot3(sin(t),cos(t),t,'o-r')
```



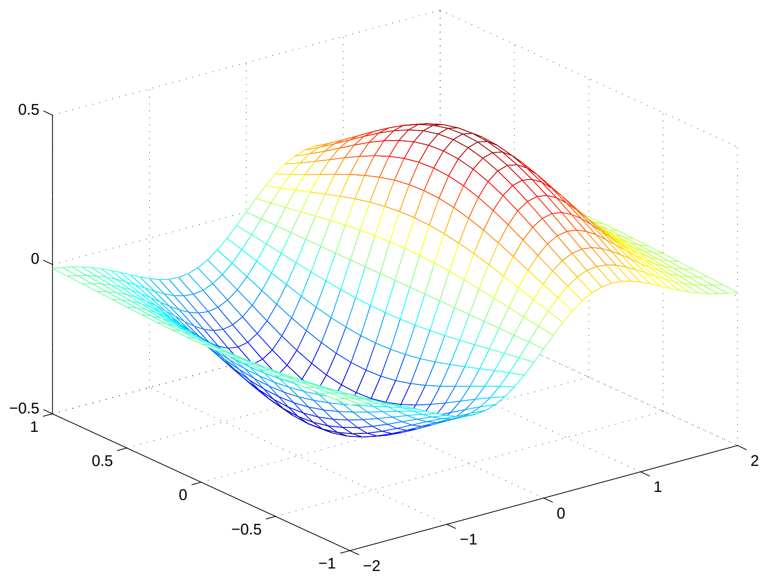
3.2.3 Polygonale Flächen

```
>> X=[0 1 0;0 1 0;0 0 0;0 1 0]';
>> Y=[0 0 1;0 0 0;0 1 0;0 0 1]';
>> Z=[0 0 0;0 0 1;0 0 1;1 0 0]';
>> rgb=[0 .8 .2];
>> fill3(X,Y,Z,rgb)
```

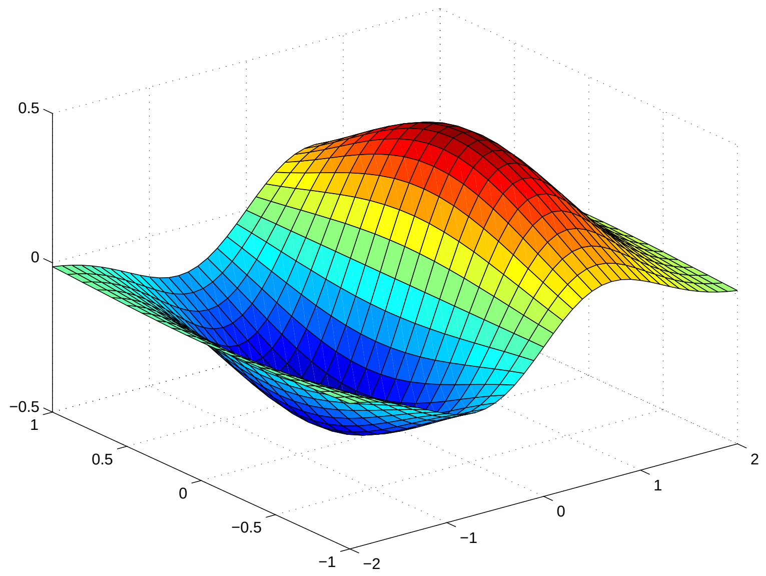


3.2.4 Flächen

```
>> [X,Y]=meshgrid(-2:.1:2,-1:.1:1);
>> Z=X.*exp(-X.^2-Y.^2);

>> mesh(X,Y,Z);
```

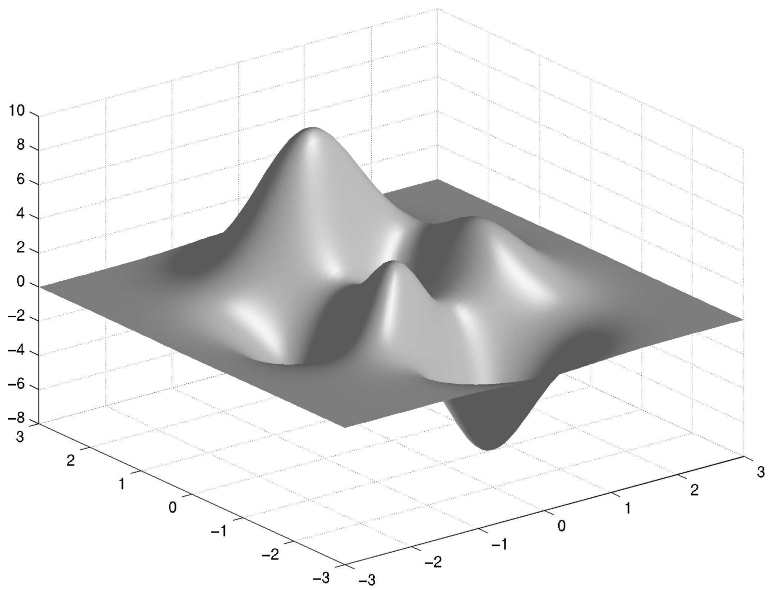



```
>> surf(X,Y,Z);
```



Beispiel:

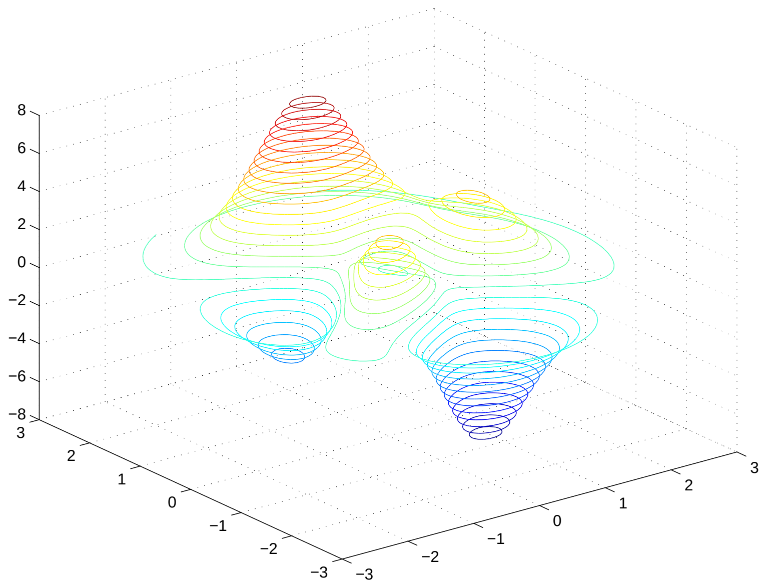
```
>> [X,Y,Z]=peaks(100);
>> surfl(X,Y,Z);
>> shading interp;
>> colormap(gray(1000));
```



Weitere color maps siehe: `help graph3d`

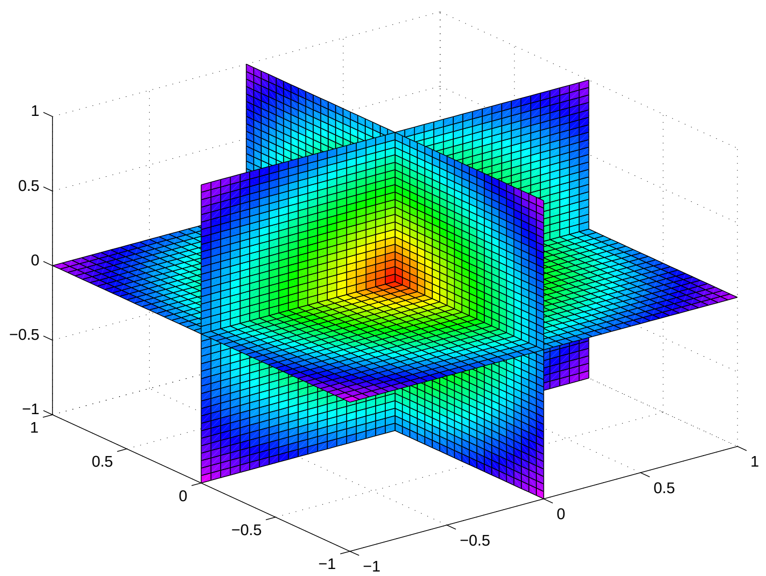
3.2.5 Höhenlinien

```
>> [X,Y,Z]=peaks(100);
>> contour3(X,Y,Z,30);
```

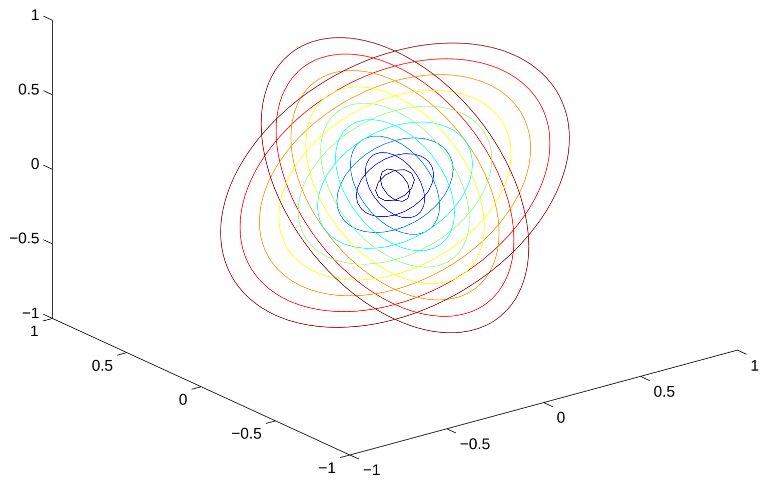


3.2.6 Schnittbilder

```
>> [X,Y,Z]=meshgrid(-1:.05:1);
>> V=sqrt(X.^2+Y.^2+Z.^2);
>> slice(X,Y,Z,V,0,0,0)
>> colormap(hsv)
```




```
>> V(V>1)=NaN;
>> colormap(jet);
>> contourslice(X,Y,Z,V,0,0,[])
>> view(3)
```



3.3 Gestaltung und Beschriftung

3.3.1 Achsensystem

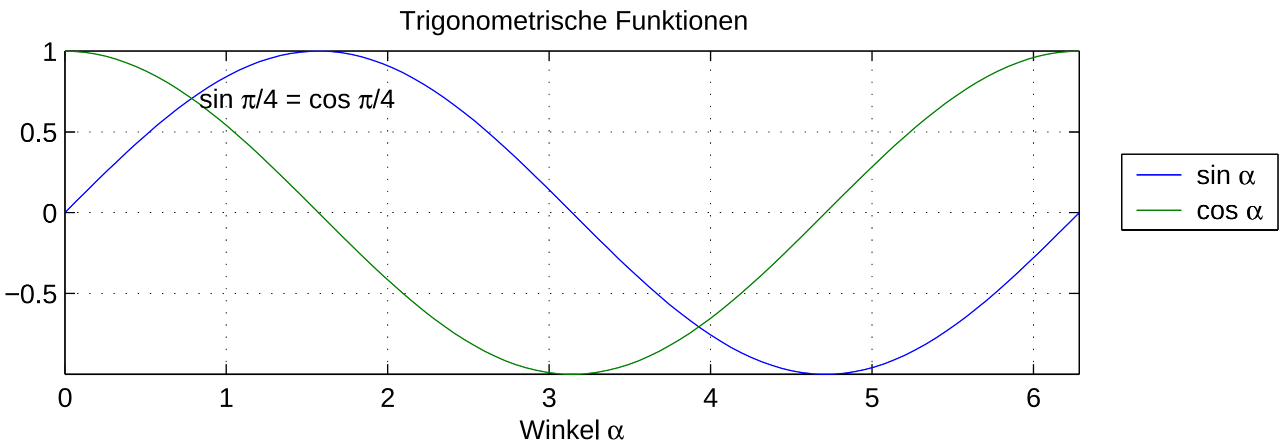
- `axis([xmin xmax ymin ymax])`: Grenzen des Achsensystems setzen
- `axis on, axis off`: Achsensystem ein- und ausschalten
- `axis auto, axis manual`: automatische Anpassung ein-/ausschalten
- `axis equal`: gleiche Längeneinheiten
- `axis tight`: an Daten angepasster Ausschnitt
- `axis image`: entspricht `axis equal`, `axis tight`
- `axis square`: quadratischer Ausschnitt
- `axis fill`: Ausfüllen des Bildfensters
- `grid on, grid off`: Gitterlinien ein- und ausschalten
- `box on, box off`: Achsensystem im Box-Format
- `pbaspect([x,y,z])`: Seitenverhältnisse des Achsensystems
- `view(az,el)`: Blickwinkel einstellen (in Grad)
- `view(2)`: 2D-Ansicht (`az=0`, `el=90`)
- `view(3)`: 3D-Ansicht (`az=-37.5`, `el=30`)

3.3.2 Beschriftung

- `title`: Überschrift
- $\left.\begin{array}{l}\texttt{xlabel}\\ \texttt{ylabel}\\ \texttt{zlabel}\end{array}\right\}$: Beschriftung der Achsen
- `legend`: Legende
- `text`: Text in Grafik
- `colorbar`: Farblegende

Beispiel:

```
>> X=linspace(0,2*pi);
>> Y=[sin(X);cos(X)];
>> plot(X,Y)
>> title('Trigonometrische Funktionen')
>> xlabel('Winkel \alpha');
>> text(pi/4,sin(pi/4),' sin \pi/4 = cos \pi/4')
>> axis equal
>> axis tight
>> grid on
>> legend(['sin \alpha';'cos \alpha'],-1)
```

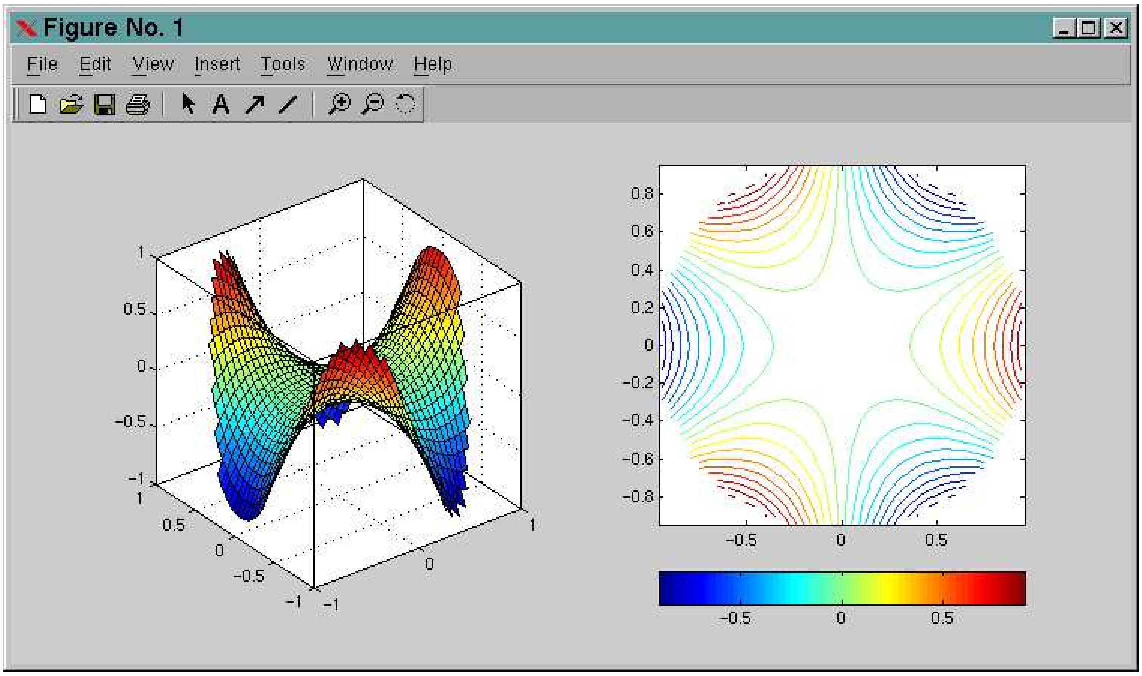


3.3.3 Mehrere Grafiken

- In einem Achsensystem: `hold on`
- In einem Fenster: `subplot`
- In unterschiedlichen Fenstern: `figure`
- Umschalten zwischen Fenstern: `figure(figureno)`
- Grafik löschen: `clf`, `clf reset`, `cla`, `cla reset`
- Fenster schließen: `close(figureno)`, `close all`

Beispiel:

```
>> [X,Y]=meshgrid(-1:.05:1);
>> Z=X.^3-3*X.*Y.^2;
>> ind=X.^2+Y.^2>1;
>> Z(ind)=NaN;

>> subplot(1,2,1)
>> surf(X,Y,Z)
>> axis equal
>> axis tight
>> box on

>> subplot(1,2,2)
>> contour(X,Y,Z,20);
>> axis equal
>> axis tight
>> colorbar('horiz');
```



3.4 Ein- und Ausgabe

3.4.1 Bild-Formate

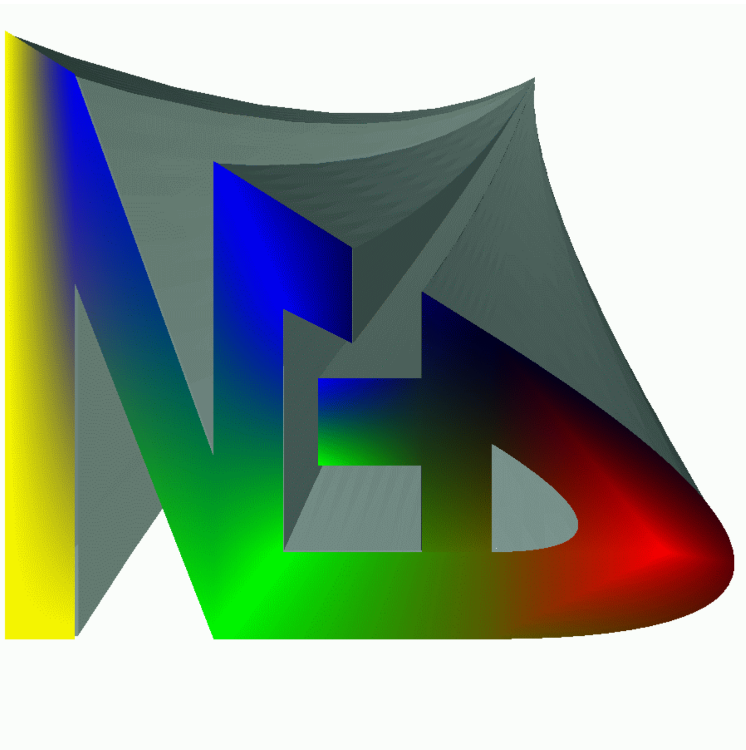
- Bilder
  - lesen und schreiben: JPEG, TIFF, BMP, PNG, HDF, PCX, XWD
  - nur lesen (MATLAB 6): GIF, ICO, CUR
  - lesen: `imread`
  - schreiben: `imwrite`
  - Informationen: `imfinfo`

  - darstellen: `image`
  - erzeugen: `getframe`, `frame2im`, `capture`

- Filme
  - MATLAB-Movie: `getframe`, `movie`
  - AVI-Datei (MATLAB 6): `movie2avi`, `avifile`, `addframe`

Beispiel:

```
>> [L,M]=imread('logo.tif');
>> image(L);
>> colormap(M);
>> axis equal; axis off
>> whos
  Name      Size         Bytes  Class

  L       900x900       810000  uint8 array
  M       256x3           6144  double array

Grand total is 810768 elements using 816144 bytes
```



3.4.2 Grafiken drucken und speichern

- `print [devicetype] [options] [filename]`
- Devices: diverse Drucker und Dateiformate, Zwischenablage
- Optionen: Papierformat, Druckmodus, Auflösung

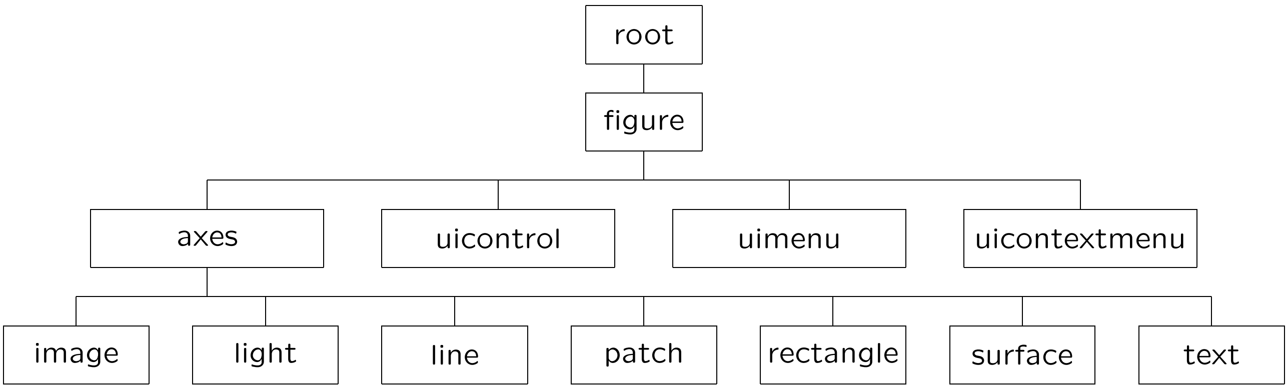
- Vektorformate: `ps`, `psc`, `eps`, `epsc`
- Bild-Formate: `bmp`, `tiff`, `jpeg`, `png`
- Auflösung bei Bild-Formaten: `-rn` mit `n` dpi (Standard: 150)
- Bei hidden-line-Fehlern: `-zbuffer`

Beispiele:

```
print -depsc meinbild
print -dtiff -r600 -zbuffer meinbild
```

3.5 Grafik-System

3.5.1 Verwaltungsstruktur des Grafik-Systems



- Baumstruktur von Objekten
- Jeder Objekttyp besitzt individuelle Eigenschaften
- Jedem Objekt wird eine Kennzahl (Handle) zugewiesen
- Wurzel des Objektbaums ist das Kontrollfenster (Handle 0)

3.5.2 Grafik-Handles

- Mit Hilfe des Handles können die Eigenschaften eines Grafikobjekts verändert werden
- `gcf`: Handle des aktiven Grafikfensters
- `gca`: Handle des aktiven Achsensystems
- `allchild(H)`: Handles aller Grafikobjekte, die dem Objekt mit Handle `H` untergeordnet sind
- `get`: Eigenschaften von Objekten anzeigen
- `set`: Eigenschaften setzen, bzw. möglichen Werte einer Eigenschaft anzeigen
- `propedit(H)`: Grafische Oberfläche zur Bearbeitung der Eigenschaften des Objekts mit Handle `H`
- Fast alle Grafikfunktionen geben die Handles der erzeugten Grafikobjekte zurück

3.5.3 Eigenschaften ausgewählter Objekte

figure-Objekt

`color`:	Farbe des Hintergrunds als rgb-Vektor
`colormap`:	Farbpalette der Grafik-Objekte
`renderer`:	Grafiktreiber. Bei falscher Darstellung von verdeckten Teilen sollte `zbuffer` oder `opengl` verwendet werden
`visible`:	Sichtbarkeit. Das Fenster kann unsichtbar gemacht werden, ohne es zu löschen
`name`:	Name des Fensters, der in der Titelzeile angezeigt wird
`units`:	Maßstab für Positionierungen (z.B. `pixel` oder `cm`)
`position`:	Position und Größe des Fensters. Mehrere Fenster können so positioniert werden, dass alle sichtbar sind. Die Werte sind [x y Breite Höhe] bezüglich der linken unteren Bildschirmecke

Weitere Eigenschaften: siehe `set(gcf)`

axes-Objekt

`color`:	Farbe des Hintergrunds als rgb-Vektor
`fontsize`:	Größe der Beschriftung
`units`:	Bemaßungssystem für Positionsangaben
`position`:	Position und Größe. Werte: [x y Breite Höhe] bezüglich der linken unteren Fensterecke
`projection`:	Projektionsart. Umschaltung zwischen orthogonal und perspektivisch möglich
`xticklabel`:	Beschriftungen der Markierungen auf der x-Achse (analog für y- und z-Achse)
`children`:	Liste aller im Koordinatensystem dargestellten Grafikobjekte

Weitere Eigenschaften: siehe `set(gca)`

3.5.4 Beispiele zu get und set

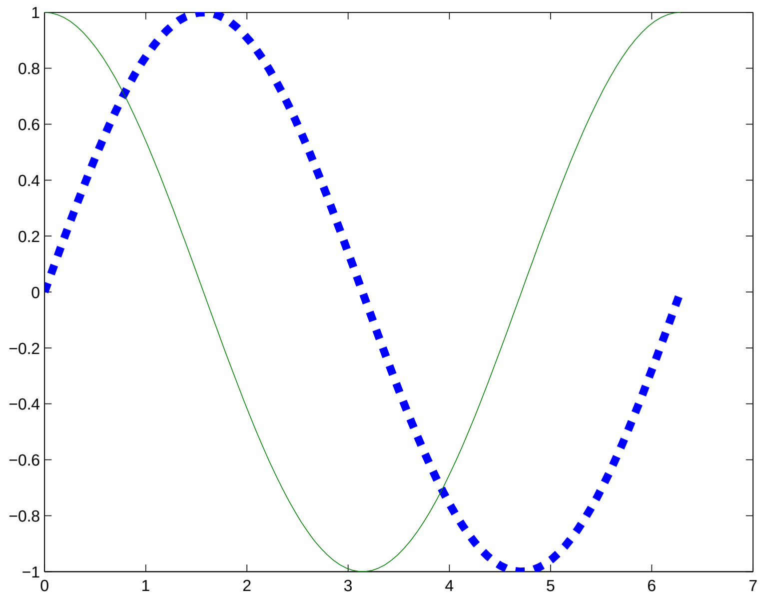
get

```
>> h=plot(1:5,6:10)
h =
  99.00634765625000
>> get(h)
        Color = [0 0 1]
        EraseMode = normal
        LineStyle = -
        LineWidth = [0.5]
        Marker = none
        MarkerSize = [6]
        MarkerEdgeColor = auto
        MarkerFaceColor = none
        XData = [1 2 3 4 5]
        YData = [6 7 8 9 10]
        ZData = []
        BeingDeleted = off
        ButtonDownFcn =
```

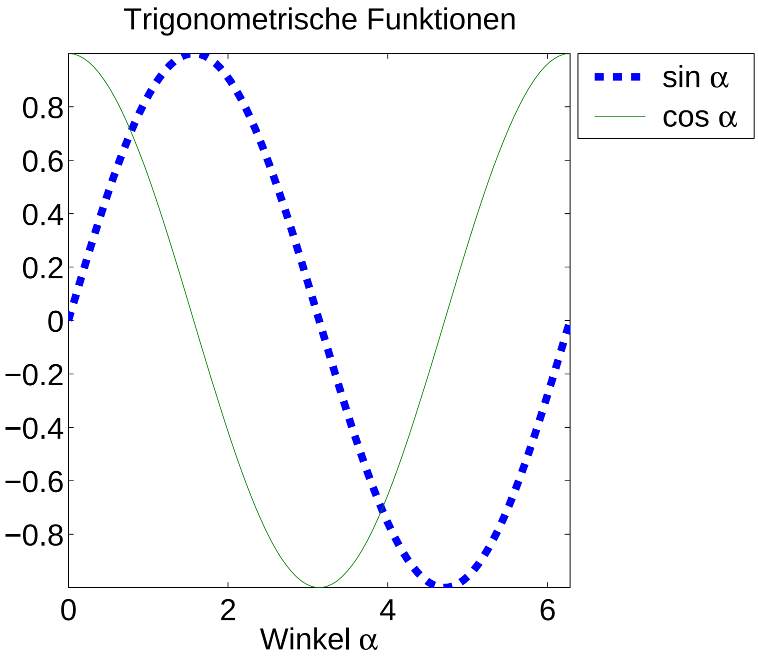
```
        Children = []
        Clipping = on
        CreateFcn =
        DeleteFcn =
        BusyAction = queue
        HandleVisibility = on
        HitTest = on
        Interruptible = on
        Parent = [100.102]
        Selected = off
        SelectionHighlight = on
        Tag =
        Type = line
        UIContextMenu = []
        UserData = []
        Visible = on
```

set

```
>> X=linspace(0,2*pi);
>> Y=[sin(X);cos(X)];
>> h1=plot(X,Y)'
h1 =
    3.0308  101.0558
>> set(h1(1),'linewidth',5,'linestyle','--')
```




```
>> h2=title('Trigonometrische Funktionen');
>> xlabel('Winkel \alpha','fontsize',20);
>> h3=legend(['sin \alpha';'cos \alpha'],-1);
>> set([h2,h3,gca],'fontsize',20);
>> axis tight
```



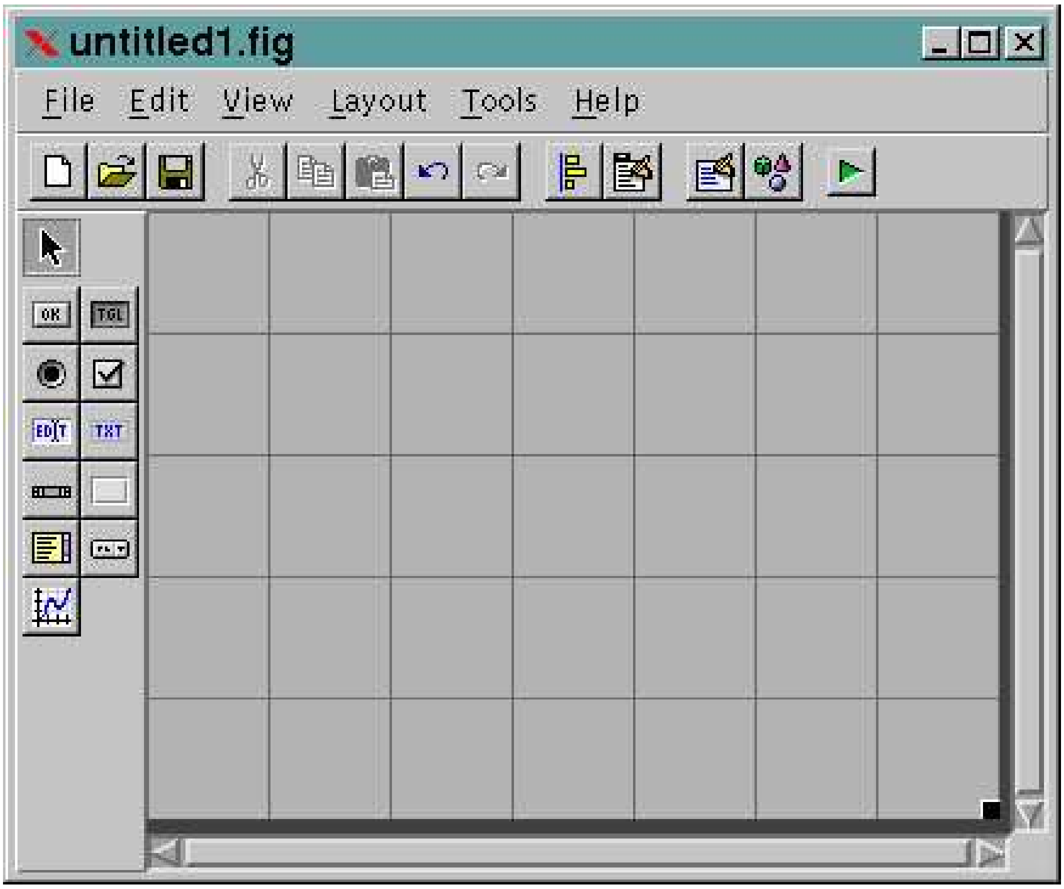
3.6 Grafische Benutzeroberflächen

3.6.1 Überblick

- Interaktiv erstellen: `guide` (GUI Design Environment)
- Kontroll-Elemente: `uicontrol`. Verfügbare Typen:
  - pushbutton (Druckknopf)
  - togglebutton (Umschalter)
  - radiobutton (Auswahlfeld)
  - checkbox (Anwahlfeld)
  - edit (Textfeld editierbar)
  - text (Textfeld nicht editierbar)
  - slider (Rollbalken)
  - frame (Rahmen)
  - listbox (Textauswahl)
  - popupmenu (Aufklappmenü)
- Menü-Einträge: `uimenu`

3.6.2 guide

`guide` startet ein interaktives Programm zum Entwurf von Grafikfenstern mit Kontrollobjekten.



3.6.3 Beispiel zu uicontrol

```
>> figure
>> set(gcf,'menubar','none','units','normalized')
>> uimenu('label','&Fenster schließen','callback','close')
>> set(gca,'position',[.4 .1 .5 .8])

>> uicontrol('style','pushbutton','string','Tortengrafik',...
             'units','normalized','position',[.1 .8 .2 .1],...
             'callback','pie(rand(1,6)); axis on; box on;')

>> uicontrol('style','pushbutton','string','Grafik löschen',...
             'units','normalized','position',[.1 .6 .2 .1],...
             'callback','cla')

>> uicontrol('style','text','string','Der Tortenbäcker',...
             'units','normalized','position',[.1 .1 .2 .4])
```

Figure nach Betätigung der Tortengrafik-Schaltfläche:

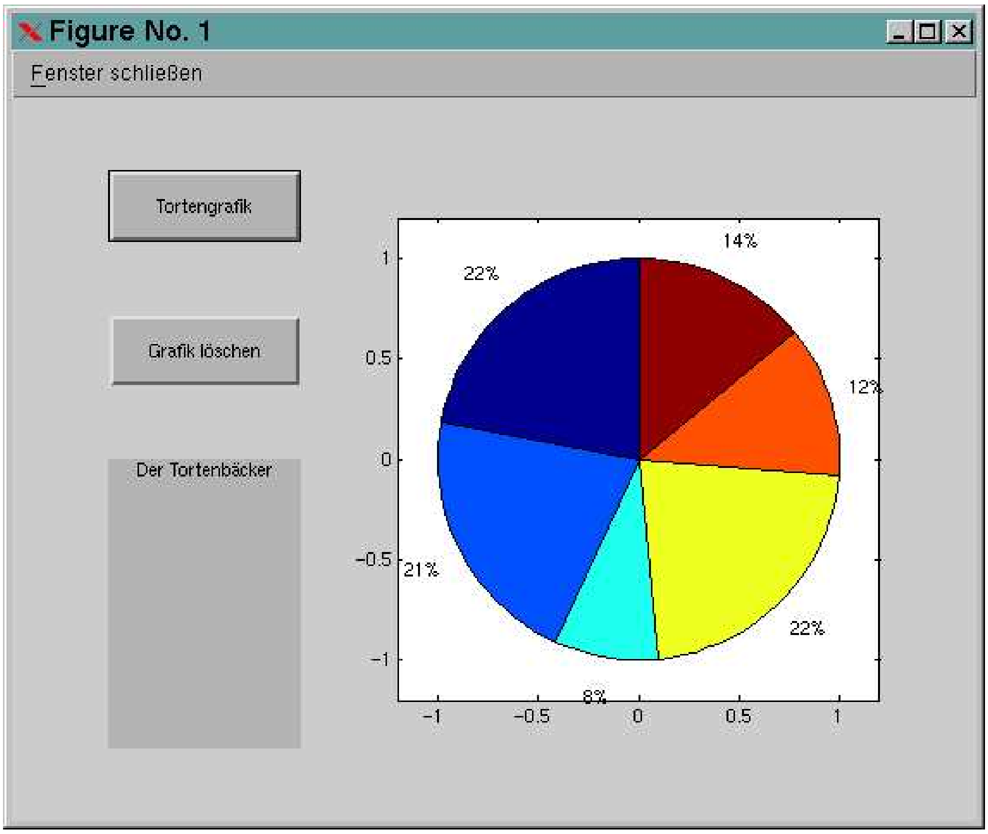


Figure No. 1
Fenster schließen
Tortengrafik
Grafik löschen
Der Tortenbäcker
14%
22%
12%
21%
22%
8%
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1

Kapitel 4

Programmierung

4.1 Skripten

4.1.1 Skripten

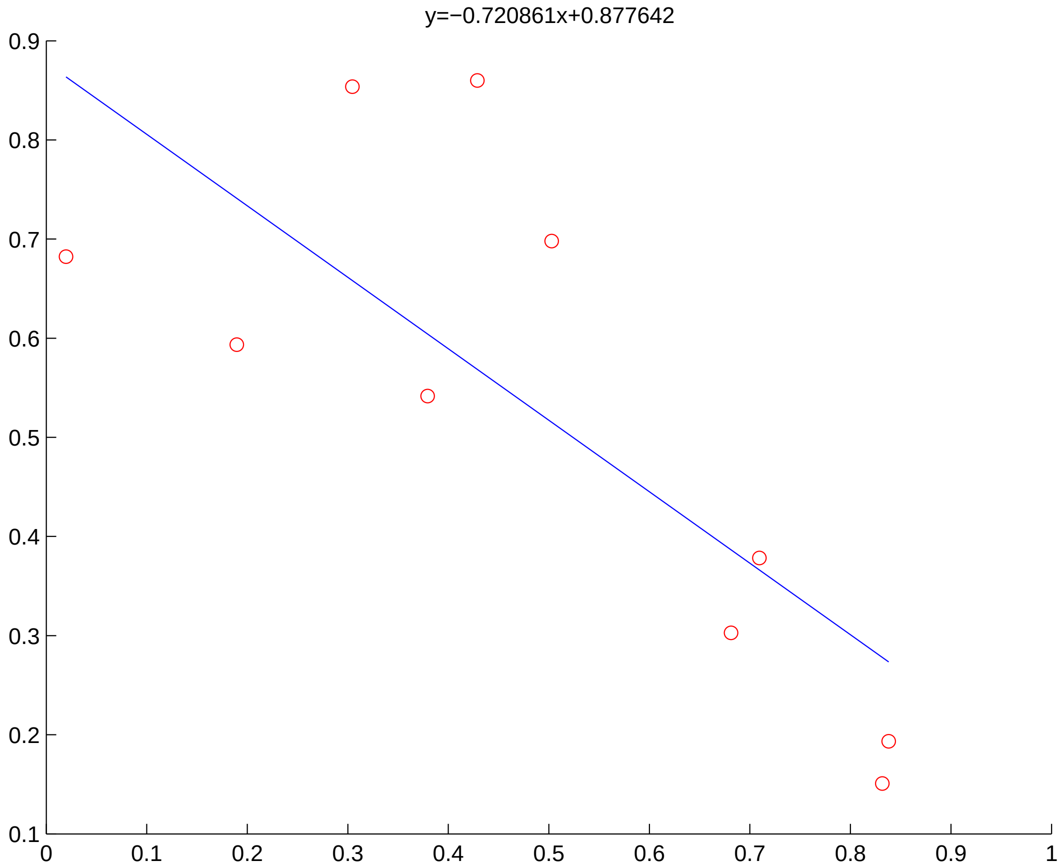
- Sammlung von Befehlen in einer Textdatei mit der Endung `.m`
- Aufruf mit Dateinamen ohne Endung `.m`
- Wirkung wie bei direkter Eingabe der Befehle im Kommandofenster
- Befehlstrennung durch Zeilenschaltung
- Befehlsreihung mit , oder ;
- Kommentarzeichen: `%` (Rest der Zeile wird ignoriert)
- `pause`: auf beliebigen Tastendruck warten
  `pause(n)`: Abarbeitung um `n` Sekunden verzögern
  `pause off`: weitere Pause-Befehle abschalten
- `edit`: Editor zum Bearbeiten von `.m`-Dateien starten

Beispiel:
Inhalt der Datei `ausgleichsgerade.m`:

```
% Ausgleichsgerade berechnen
K=[X(:),ones(length(X),1)]\Y(:);
m=K(1)
c=K(2)

% Ausgleichsgerade visualisieren
figure
plot(X,Y,'ro')
hold on
plotx=[min(X),max(X)];
plot(plotx,m*plotx+c);
title(sprintf('y=%gx+%g',m,c))
```

Aufruf:

```
>> X=rand(1,10);
>> Y=rand(1,10);
>> ausgleichsgerade
m =
   -0.7209
c =
    0.8776
```



4.1.2 Verzeichnisse, MATLAB-Pfad

- Skripten werden zunächst im Arbeitsverzeichnis und dann im MATLAB-Pfad gesucht
- `pwd`: Arbeitsverzeichnis ausgeben
- `cd`: wechseln des Arbeitsverzeichnisses
- `matlabpath`: MATLAB-Pfad ausgeben und setzen
- `addpath`, `rmpath`: MATLAB-Pfad bearbeiten
- `pathtool`: interaktive Benutzeroberfläche zur Pfadbearbeitung
- `which`: Pfad zu einem Skript oder einer Funktion ausgeben
- `what`: MATLAB-Dateien in einem Verzeichnis ausgeben

4.1.3 Datenabfrage aus Skripten

- `input`: Aufforderung an den Benutzer Werte einzugeben
  - Zeichenkette mit Aufforderungstext als erstes Argument
  - zweiter Parameter 's' falls Eingabe Zeichenkette sein soll
  - die Eingabe wird ausgewertet, kann also Funktionen und Variablen enthalten
- `ginput`: Auslesen von Punkten aus dem aktiven Grafikfenster
  - Positionierung mit Maus
  - Auswahl mit Mausklick oder Tastendruck
  - Beenden mit Eingabetaste
  - Rückgabe: x- und y-Vektoren der Punktkoordinaten. Bei Bedarf auch Vektor mit Nummer der jeweils betätigten Maustaste
- `uigetfile`, `uiputfile`: Grafische Dialogfenster zur Auswahl von Dateinamen

Beispiel zu input:

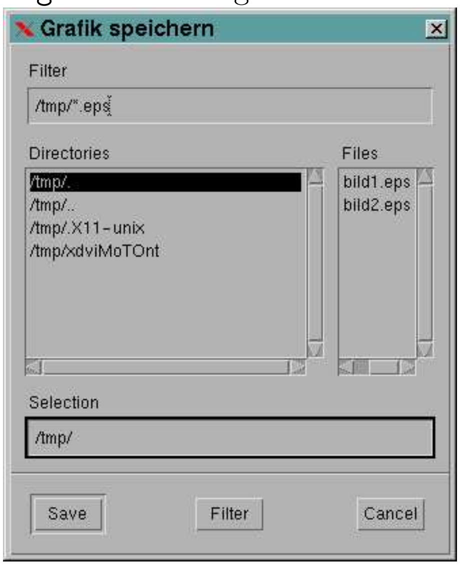
```
>> M=input('Bitte (nx2)-Matrix der Messpunkte eingeben: ')
Bitte (nx2)-Matrix der Messpunkte eingeben: [1 2;4 1;5 1]
M =
     1     2
     4     1
     5     1

>> M=input('Bitte (nx2)-Matrix der Messpunkte eingeben: ')
Bitte (nx2)-Matrix der Messpunkte eingeben: rand(2)
M =
    0.4966    0.8216
    0.8998    0.6449

>> Fct=input('Bitte geben Sie einen Funktionsnamen ein: ');
Bitte geben Sie einen Funktionsnamen ein: sin
??? Error using ==> sin
Incorrect number of inputs.

>> Fct=input('Bitte geben Sie einen Funktionsnamen ein: ','s')
Bitte geben Sie einen Funktionsnamen ein: sin
Fct =
sin
```

Beispiel zur Datenabfrage:
Inhalt der Skript-Datei `ausgleichsgerade.m`:

```
% Ausgleichspunkte grafisch einlesen
disp('Punkte anklicken. Eingabe mit Enter-Taste beenden.')
[X,Y]=ginput;
```

```
% Ausgleichsgerade berechnen
K=[X(:),ones(length(X),1)]\Y(:);
m=K(1), c=K(2)

% Ausgleichsgerade visualisieren
figure
plot(X,Y,'ro')
hold on
plotx=[min(X),max(X)];
plot(plotx,m*plotx+c);
title(sprintf('y=%gx+%g',m,c))

% Grafik abspeichern und Fenster schließen
[file,path]=uiputfile('/tmp/*.eps','Grafik speichern');
print('-depsc',[path file])
disp('Beliebige Taste schließt das Grafikfenster.')
pause
close
```

`uigetfile`-Dialogfenster:



4.2 Kontrollstrukturen

4.2.1 Übersicht

- if-Abfrage: Bedingte Verzweigung.
  Unterschiedlicher Befehlssequenzen in Abhängigkeit des Wahrheitswertes logischer Ausdrücke ausführen.

- switch-Anweisung: Fallunterscheidung.
  Ausführung von Befehlssequenzen in Abhängigkeit des Wertes einer Variablen. Diskreten Wertemengen werden entsprechende Befehlssequenzen zugeordnet.

- for-Schleife: n-fache Ausführung einer Befehlssequenz.
  Bei der for-Schleife wird einer im Schleifenkopf gekennzeichneten Variablen bei jedem

Schleifendurchlauf ein neuer Wert zugewiesen.

- while-Schleife: Ausführung einer Befehlssequenz, bis ein im Schleifenkopf stehender logischer Ausdruck den Wert falsch annimmt oder im Schleifenrumpf eine Abbruchanweisung ausgeführt wird.

4.2.2 if-Abfrage

Syntax:
if *logischer Ausdruck*
Befehle
elseif *logischer Ausdruck*
Befehle
else
Befehle
end

Erläuterung:
Ist der logische Ausdruck wahr, werden die unmittelbar folgenden Befehle ausgeführt. Andernfalls wird der logische Ausdruck der nachfolgenden `elseif`-Anweisung geprüft usw. Sind alle logischen Ausdrücke falsch, werden die Befehle des `else`-Zweigs ausgeführt.
Hinweise:

- `elseif`- und `else`-Zweige können entfallen
- Beliebig viele `elseif`-Zweige erlaubt
- `if x~=0` entspricht `if x`
- Indikatorfunktionen `isempty`, `isstr`, `ischar`, `isinf`, `isnan`, `isfinite`, usw. verwenden
- Soll anhand einer überschaubaren Menge diskreter Werte entschieden werden, ist `switch` der Verwendung von `if` vorzuziehen

Beispiel:
Signum `s` einer rationalen Zahl `x` bestimmen.
1. Variante:

```
if x>0
  s=1;
elseif x<0
  s=-1;
else
  s=0;
end
```

2. Variante:

```
if x>0
  s=1;
elseif ~x
  s=0;
```

```
else
  s=-1;
end
```

3. Variante (ohne if-Abfrage):

```
s=(x>0)-(x<0);
```

4.2.3 switch-Anweisung

Syntax:
switch *Ausdruck*
case *Vergleichsausdruck*
Befehle
case {*VA1,VA2,...,VAn*}
Befehle
otherwise
Befehle
end

Erläuterung:
`switch`-Ausdruck mit `case`-Ausdrücken vergleichen und bei Übereinstimmung zugehörige Befehle ausführen, sonst `otherwise`-Befehle ausführen.
Hinweise:

- `switch`-Ausdruck kann Skalar oder Zeichenkette sein
- `otherwise` optional, beliebig viele `case`-Zweige

Beispiel:

```
switch n
  case {1,4,9}
    fprintf('%d ist eine Quadratzahl\n',n);
  case {2,3,5,7}
    fprintf('%d ist eine Primzahl\n',n);
  case 6
    fprintf('%d hat zwei Primfaktoren: 2 und 3\n',n);
  case 8
    fprintf('%d ist eine Kubikzahl\n',n);
  otherwise
    disp('n muss natürliche Zahl zwischen 1 und 9 sein.');
end
```

4.2.4 for-Schleife

Syntax:
for *Variable* = *Matrix*
Befehle
end

Erläuterung:
Der Variablen werden nacheinander die Spalten der Matrix zugewiesen. Für jede Spalte werden die Befehle einmal ausgeführt.
Hinweise:

- n Schleifendurchläufe: `for zaehler=1:n`
- Vorzeitiger Abbruch der Schleife durch `break` möglich
- `for`-Schleifen können oft durch geeignete Vektor-/Matrixoperationen ersetzt werden. Diese sind in der Regel wesentlich effizienter

Beispiel:
Monte-Carlo-Verfahren zur Bestimmung der durchschnittlichen Norm eines dreidimensionalen Vektors mit in (0,1) gleichverteilten Koeffizienten.
Implementierung mit for-Schleife:

```
>> tic
>> N=0;
>> for k=1:100000
   x=rand(3,1);
   N=N+norm(x);
end
>> N/100000
ans =
    0.9615
>> toc
elapsed_time =
    3.5918
```

Implementierung ohne for-Schleife:

```
>> tic
>> mean(sqrt(sum(rand(3,100000).^2)))
ans =
     0.9603
>> toc
elapsed_time =
     0.1544
```

4.2.5 while-Schleife

Syntax:
while *logischer Ausdruck*
Befehle
end
Erläuterung:
Befehle werden ausgeführt, bis der logische Ausdruck falsch ist.
Hinweise:

- Logischer Ausdruck ist wahr, wenn alle Elemente des Realteils nicht Null sind

- Vorzeitiger Abbruch der Schleife durch `break`-Anweisung möglich
- Endlos-Schleife, wenn der logische Ausdruck stets wahr ist

Beispiel:
Wie lautet die größte Zahl, deren Fakultät noch darstellbar ist?

```
>> f=1;
>> n=1;
>> while ~isinf(f)
  n=n+1;
  f=f*n;
end
>> n-1
ans =
   170
```

Variante: Schleifen-Abbruch mit `break`-Anweisung:

```
>> f=1;
>> n=1;
>> while 1
  n=n+1;
  f=f*n;
  if isinf(f)
    break
  end
end
>> n-1
ans =
   170
```

4.3 Funktionen

4.3.1 Funktionen

- Funktionen haben die Dateinamen-Erweiterung `.m`
- Sie werden mit ihrem Namen aufgerufen (ohne `.m`)
- Erste Zeile:
  - Schlüsselwort `function`
  - Definition des Aufrufs: `[`Rückgabeliste`]=`Name`(`Parameterliste`)`
  - Name beliebig, sollte aber dem Dateinamen entsprechen
  - bei leeren Listen können auch die Klammern entfallen
- Variablen sind lokal
- Parameter werden kopiert (deep copy)

- Verlassen mit `return` oder bei Dateiende
- `type` <Funktion> zeigt den Quellcode der Funktion an
- Viele MATLAB Funktionen sind als .m Dateien verfügbar

Beispiel einer Funktion:

```
function [m,c]=ausgleichsgerade(X,Y)

% Ausgleichsproblem loesen
K=[X(:),ones(length(X),1)]\Y(:);
m=K(1);
c=K(2);

% Ergebnis visualisieren
figure
plot(X,Y,'ro')
hold on
plotx=[min(X),max(X)];
plot(plotx,m*plotx+c);
title(sprintf('y=%gx+%g',m,c))
```

Beispiel zu Funktionen anzeigen:

```
>> type cot

function y = cot(z)
%COT    Cotangent.
%   COT(X) is the cotangent of the elements of X.

%   Copyright 1984-2000 The MathWorks, Inc.
%   $Revision: 5.5 $  $Date: 2000/06/01 00:40:16 $

y = 1./tan(z);
```

4.3.2 Länge von Argumentlisten

- `nargin`: Anzahl der Eingabeargumente
- `nargout`: Anzahl der Ausgabeargumente
- `exist`: prüfen, ob eine Variable existiert
- `varargin`: Eingabeparameterliste unbestimmter Länge (cell-array)
- `varargout`: Ausgabeparameterliste unbestimmter Länge (cell-array)

Beispiel:
Inhalt der Datei `ausgleichsgerade.m`:

```
function [m,c]=ausgleichsgerade(X,Y)

if nargin==0
  error('Keine Daten vorhanden')
end
if ~exist('Y','var')
  Y=X;
  X=1:length(Y);
end

K=[X(:),ones(length(X),1)]\Y(:);

if nargout>0
  m=K(1);
end
if nargout>1
  c=K(2);
end
```

Ausgaben der Funktion:

```
>> X=rand(1,10);
>> Y=rand(1,10);

>> ausgleichsgerade
??? Error using ==> ausgleichsgerade
Keine Daten vorhanden

>> [m,c]=ausgleichsgerade(Y)
m =
    0.0463
c =
    0.2706

>> m=ausgleichsgerade(X,Y)
m =
   -0.7209

>> ausgleichsgerade(X,Y)
```

4.3.3 Kommentare

- Beginnen mit `%` und gehen bis zum Ende der Zeile
- Bei `help` <Funktion> wird der erste Kommentarblock angezeigt
- Bei `help` <Verzeichnis> wird die erste Kommentarzeile angezeigt

Beispiel:
Inhalt der im Verzeichnis `/tmp` gespeicherten Datei `ausgleichsgerade.m`:

```
function [m,c]=ausgleichsgerade(X,Y)
% AUSGLEICHSGERADE: Berechnung einer Ausgleichsgeraden
%
%   Input : X,Y  Vektoren mit Messpunktdaten
%   Output: m,c  Parameter der Ausgleichsgeraden

% Version 0.1, 15.4.2002
K=[X(:),ones(length(X),1)]\Y(:); m=K(1); c=K(2);
```

Ausgaben:

```
>> help /tmp

 AUSGLEICHSGERADE: Berechnung einer Ausgleichsgeraden


>> help ausgleichsgerade

  AUSGLEICHSGERADE: Berechnung einer Ausgleichsgeraden

    Input : X,Y  Vektoren mit Messpunktdaten
    Output: m,c  Parameter der Ausgleichsgeraden
```

4.3.4 Rekursion

- Funktionen können sich selbst aufrufen
- Jede Instanz hat eigene (lokale) Variablen
- Maximale Rekursionstiefe 100
  Ändern mit `set(0,'RecursionLimit',N)`

Beispiel:
Inhalt der Datei `fak.m`:

```
function f=fak(n)
if n==1
  f=1;
else
  f=n*fak(n-1);
end;
```

Funktionsaufruf:

```
>> fak(8)
ans =
    40320

>> fak(101)
??? Maximum recursion limit of 100 reached. Use set(0,'RecursionLimit',N)
to change the limit.  Be aware that exceeding your available stack space can
crash MATLAB and/or your computer.
```

```
Error in ==> /tmp/fak.m
On line 6  ==>   f=n*fak(n-1);
```

4.3.5 Funktionen als Parameter

Übergabe als Zeichenkette:

- Funktionsnamen als Zeichenkette übergeben
- Auswertung der Funktion mit `feval`(Name,Parameterliste)
- Parameterliste wie beim direkten Aufruf der Funktion übergeben

Übergabe als Handle:

- Funktions-Handle: `@`<Funktionsname>
- Auswertung der Funktion mit `feval`

Übergabe als Inline-Objekt:

- Erzeugen von Inline-Objekten: `inline(’`*Ausdruck*`’)`
- Parameternamen werden automatisch bestimmt
- Auswertung der Funktion durch direkten Aufruf

Beispiel:
Übergabe als Zeichenkette:

```
function plot_sf(f,xmin,xmax)
x=linspace(xmin,xmax);
y=feval(f,x);
plot(x,y)
```

```
>> f='sin'
f =
sin
>> plot_sf(f,0,pi)
```

Übergabe als Funktionen-Handle:

```
function plot_hf(f,xmin,xmax)
x=linspace(xmin,xmax);
y=feval(f,x);
plot(x,y)
```

```
>> f=@sin
f =
    @sin
>> plot_hf(f,0,pi)
```

Übergabe als Inline-Objekt:

```
function plot_if(f,xmin,xmax)
x=linspace(xmin,xmax);
y=f(x);
plot(x,y)
```

```
>> f=inline('sin(x)')
f =
     Inline function:
     f(x) = sin(x)
>> plot_if(f,0,pi)
```

4.3.6 Globale Variablen

- Deklaration mit dem Befehl `global`
- Deklaration vor der ersten Zuweisung
- Müssen auch auf der Kommandozeile deklariert werden
- Anzeigen mit `who global`, `whos global`
- Löschen mit `clear global`

Beispiel:
Zeitmessungs-Funktionen `tic` und `toc` (ohne Kommentare):

```
function tic
global TICTOC
TICTOC = clock;


function t = toc
global TICTOC
if isempty(TICTOC)
  error('You must call TIC before calling TOC.');
end
if nargout < 1
   elapsed_time = etime(clock,TICTOC)
else
   t = etime(clock,TICTOC);
end
```

4.3.7 Lokale Funktionen

- Weitere Funktionsdefinitionen in einer Datei
- Direkter Aufruf nur durch Funktionen in der gleichen Datei
- Aufruf mit dem in der Funktionsdeklaration verwendeten Namen

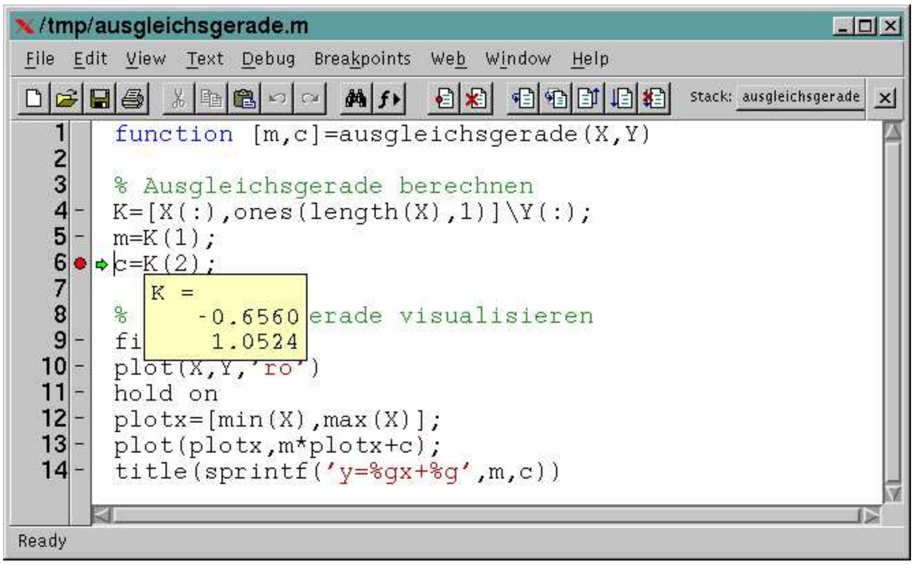
Beispiel:
Inhalt der Datei `ausgleichsgerade.m` mit den beiden lokalen Funktionen `ag_berechnen` und `ag_visualisieren`:

```
function [m,c]=ausgleichsgerade(X,Y)
[m,c]=ag_berechnen(X,Y);
ag_visualisieren(X,Y,m,c);

function [m,c]=ag_berechnen(X,Y)
K=[X(:),ones(length(X),1)]\Y(:);
m=K(1);
c=K(2);
```

```
function ag_visualisieren(X,Y,m,c)
figure
hold on
plot(X,Y,'ro')
plotx=[min(X),max(X)];
plot(plotx,m*plotx+c);
```

4.4 Sonstiges

4.4.1 Fehlerbeseitigung (Debugging)



- MATLAB-Editor ist zugleich Debugger
- Haltepunkte einfügen, Einzelschrittausführung
- Tooltip-Anzeige von Variableninhalten
- Steuerung auch im Kommandofenster möglich
  Steuerbefehle: `dbstop`, `dbstep`, `dbcont`, `dbclear`, `dbquit`

4.4.2 Programmanalyse (Profiling)

Zeitmessung:

- `cputime`: verwendete Rechenzeit seit MATLAB-Start
- `clock`: aktuelle Systemzeit, Differenzberechnung mit `etime`
- `tic`,`toc`: Stoppuhr starten und anhalten

Programmanalyse

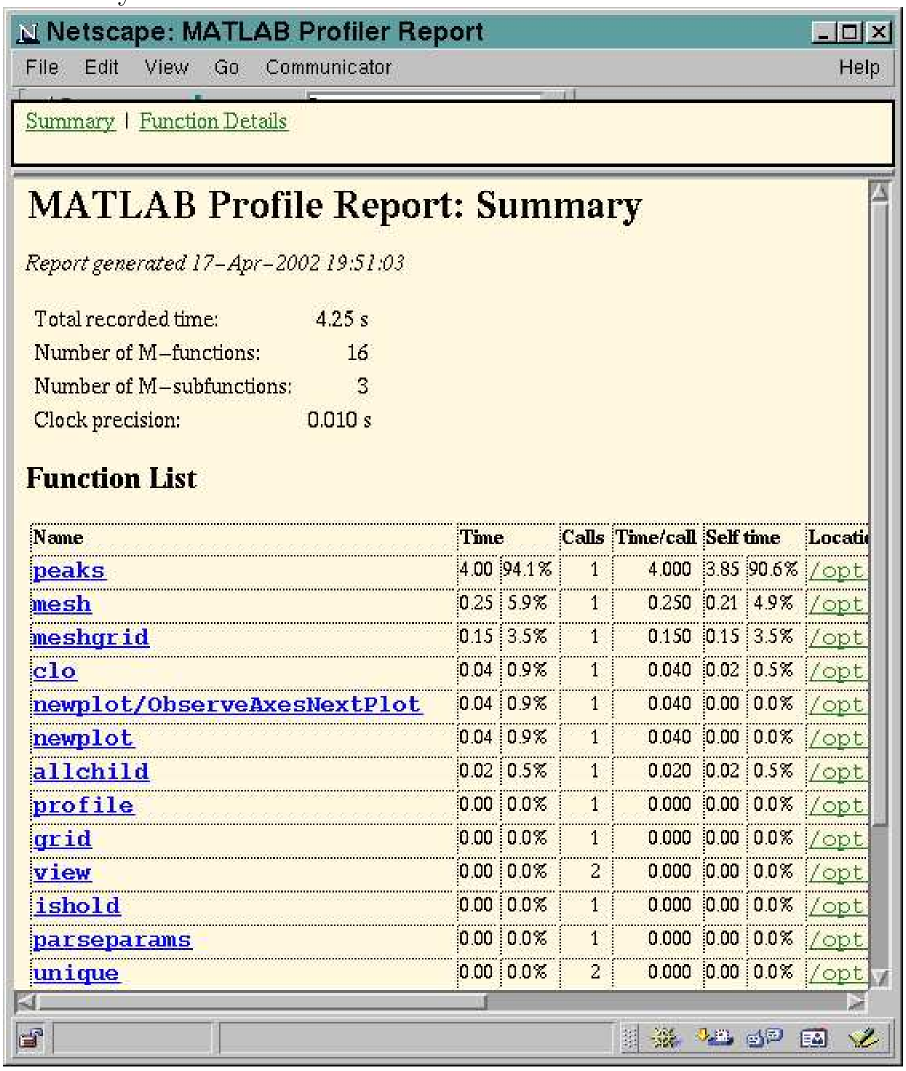
- `profile on`: Programmanalyse einschalten
- `profile report`: Analysebericht erstellen und im WEB-Browser (z.B. Netscape) anzeigen

- Auskunft über Laufzeit, aufgerufene Funktionen, Aufrufstruktur

Beispiel:

```
>> profile on; mesh(peaks(1000)); profile report
```

Summary:



Netscape: MATLAB Profiler Report

File Edit View Go Communicator Help

Summary | Function Details

MATLAB Profile Report: Summary

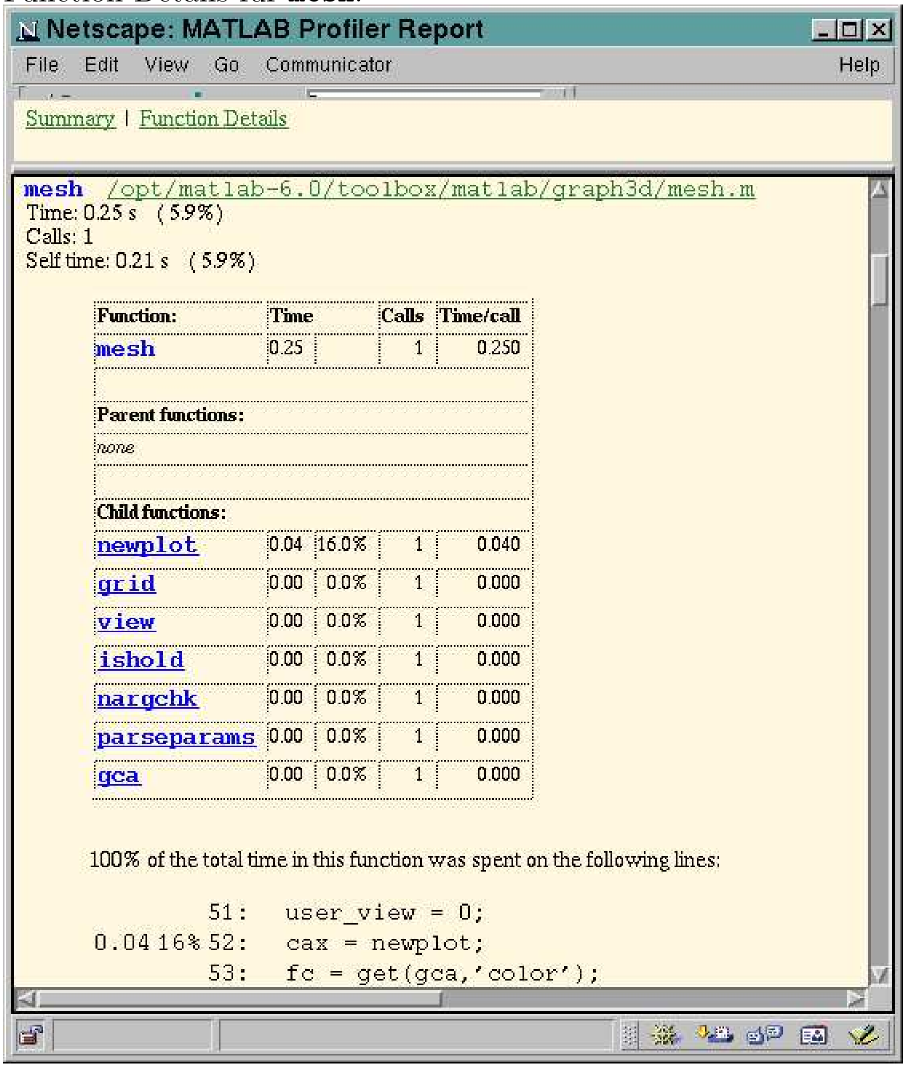
Report generated 17-Apr-2002 19:51:03

Total recorded time:	4.25 s
Number of M-functions:	16
Number of M-subfunctions:	3
Clock precision:	0.010 s

Function List

Name	Time		Calls	Time/call	Self time		Locati
peaks	4.00	94.1%	1	4.000	3.85	90.6%	/opt
mesh	0.25	5.9%	1	0.250	0.21	4.9%	/opt
meshgrid	0.15	3.5%	1	0.150	0.15	3.5%	/opt
clo	0.04	0.9%	1	0.040	0.02	0.5%	/opt
newplot/ObserveAxesNextPlot	0.04	0.9%	1	0.040	0.00	0.0%	/opt
newplot	0.04	0.9%	1	0.040	0.00	0.0%	/opt
allchild	0.02	0.5%	1	0.020	0.02	0.5%	/opt
profile	0.00	0.0%	1	0.000	0.00	0.0%	/opt
grid	0.00	0.0%	1	0.000	0.00	0.0%	/opt
view	0.00	0.0%	2	0.000	0.00	0.0%	/opt
ishold	0.00	0.0%	1	0.000	0.00	0.0%	/opt
parseparams	0.00	0.0%	1	0.000	0.00	0.0%	/opt
unique	0.00	0.0%	2	0.000	0.00	0.0%	/opt

Function Details für `mesh`:



Kapitel 5

Anwendungsbeispiele

5.1 Lineare Algebra

5.1.1 Skalar- und Kreuzprodukt

`s=dot(x,y)` bestimmt zu den Vektoren `x,y` der Länge n das Skalarprodukt $s = \langle x, y \rangle$.
`z=cross(x,y)` ermittelt zu den Vektoren `x,y` der Länge 3 das Kreuzprodukt $z = x \times y$.
Beispiel:

```
>> % Vektoren x und y definieren
>> x=[1  0 -2];
>> y=[0 -1  1];

>> % Kreuzprodukt berechnen
>> z=cross(x,y)
z =
    -2    -1    -1

>> % Orthogonalitaetsbetrachtung
>> o=[dot(x,y), dot(x,z), dot(y,z)]
o =
    -2     0     0
```

5.1.2 Rang und Determinante

`r=rank(A)` schätzt den Rang der $(n \times m)$-Matrix `A`.
`d=det(A)` berechnet die Determinante der $(n \times n)$-Matrix `A`.
Hinweis: Aufgrund von Rundungsfehlern sind die Ergebnisse in der Regel nicht exakt.

Beispiel:

Beispiel 1: Matrix vom Rang 2, d.h. Zeilen sind linear abhängig. Ergebnisse sind exakt.

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
```

```
     7     8     9
>> rank(A)
ans =
     2
>> det(A)
ans =
     0
```

Beispiel 2: (5×5)-Zufallsmatrix, deren fünfte Zeile ein Vielfaches der ersten Zeile ist. Determinante ist nicht exakt, sondern nur bis auf Rechengenauigkeit 0.

```
>> A=rand(5);
>> A(5,:)=17*A(1,:);
>> rank(A)
ans =
     4
>> det(A)
ans =
   1.3423e-17
```

5.1.3 Eigenwerte und Eigenvektoren

`[V,D]=eig(A)` berechnet die Eigenvektoren und Eigenwerte der $(n \times n)$-Matrix `A`.
Die Spalten der $(n \times n)$-Matrix `V` bestehen aus den zu 1 normierten Eigenvektoren. Die Diagonale der $(n \times n)$-Matrix `D` enthält die zugehörigen Eigenwerten von `A`. Es gilt $AV = VD$.
Ist die algebraische Vielfachheit k eines Eigenwertes größer als seine geometrische Vielfachheit l, enthält `V` neben den l Eigenvektoren noch $k - l$ weitere Linearkombinationen dieser Eigenvektoren.
Beispiel:
Matrix mit Eigenwerten 1 und $1 \pm 2i$:

```
>> A=[1 0 0 ; 2 1 -2 ; 3 2 1]
A =
     1     0     0
     2     1    -2
     3     2     1

>> [V,D]=eig(A)
V =
        0                  0            0.4851
   0.7071             0.7071           -0.7276
        0 - 0.7071i        0 + 0.7071i   0.4851

D =
   1.0000 + 2.0000i        0                 0
        0             1.0000 - 2.0000i        0
        0                  0             1.0000

>> % Diagonalisierung (bis auf Rundungsfehler)
>> inv(V)*A*V
```

```
D =
   1.0000 + 2.0000i   0.0000 - 0.0000i  -0.0000 - 0.0000i
   0.0000 + 0.0000i   1.0000 - 2.0000i  -0.0000 + 0.0000i
        0                  0             1.0000
```

Matrix mit Eigenwerten 3 und 2. Algebraische Vielfachheit des Eigenwerts 2 ist 3, geometrische Vielfachheit 1:

```
>> A=[3 0 0 0 ; 0 2 1 0 ;  0 0 2 1 ; 0 0 0 2]
A =
     3     0     0     0
     0     2     1     0
     0     0     2     1
     0     0     0     2

>> [V,D]=eig(A)
V =
    1.0000         0         0         0
         0    1.0000   -1.0000    1.0000
         0         0    0.0000   -0.0000
         0         0         0    0.0000


D =
     3     0     0     0
     0     2     0     0
     0     0     2     0
     0     0     0     2

>> % 3. und 4. Eigenvektor sind linear abhaengig
>> rank(V)
ans =
     2
```

5.1.4 Lösung linearer Gleichungssysteme und Matrixinversion

`I=inv(A)` berechnet die Inverse der $(n \times n)$-Matrix `A`.

`X=A\B` bestimmt die (Ausgleichs-)Lösung des linearen Gleichungssystems $AX = B$.

- Ist `A` eine $(n \times n)$-Matrix, wird das System mit Hilfe der Gauß-Elimination gelöst.
- Ist `A` eine $(m \times n)$-Matrix mit $m \neq n$, wird die Ausgleichslösung des über- bzw. unterbestimmten Gleichungssystems gelöst, d.h. es wird $||AX - B||_2^2$ minimiert.

Bei numerisch instabilen Matrizen oder Systemen mit Rangverlust wird eine Warnung ausgegeben (Matrix is close to singular or badly scaled).
Anmerkung: Die Berechnung der Lösung durch `X=A\B` ist unbedingt der Darstellung `X=inv(A)*B` vorzuziehen, da bei dem \-Befehl effizientere und stabilere Algorithmen verwendet werden.
Beispiel: Lösung linearer Gleichungssysteme
Inverse einer (2×2)-Matrix mit vollem Rang berechnen und lineares Gleichungssystem mit rechter Seite $(2, 0)^t$ lösen:

```
>> A=[0 -1; -1 2]
A =
     0    -1
    -1     2

>> I=inv(A)
I =
    -2    -1
    -1     0

>> X=A\[2;0]
X =
    -4
    -2
```

(3 × 3)-Matrix mit Rang 2 (nicht invertierbar):

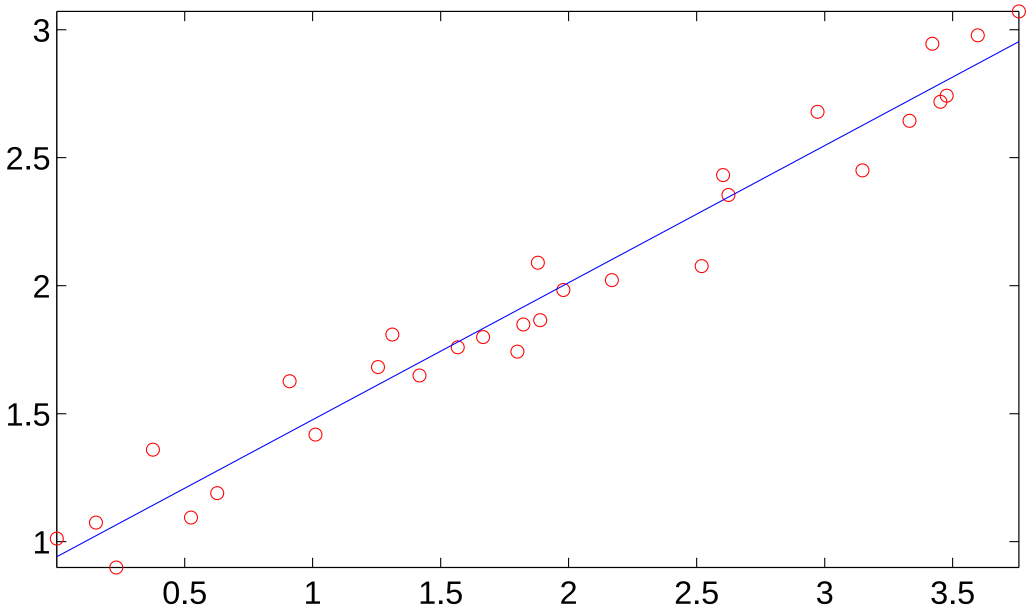
```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> rank(A)
ans =
     2

>> % A nicht invertierbar!
>> I=inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.203039e-18.
I =
   1.0e+16 *
    0.3152   -0.6304    0.3152
   -0.6304    1.2609   -0.6304
    0.3152   -0.6304    0.3152

>> % System mit mehrdeutiger Loesung
>> X=A\[1;2;3]
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.203039e-18.
X =
   -0.2334
    0.4667
    0.1000

>> % unloesbares System
>> X=A\[4;2;-3]
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.203039e-18.
```

```
X =
   1.0e+16 *
   -0.9457
    1.8913
   -0.9457
```

Beispiel: Bestimmung einer Ausgleichsgeraden
Ausgleichsgerade $y = ax+b$ der in den Vektoren `X,Y` gespeicherten Koordinaten der Messpunkte (x_i, y_i) mit $1 \le i \le n > 2$.
Überbestimmtes lineares Gleichungssystem:

$$\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} .$$

```
>> % Ausgleichsproblem aufstellen und loesen
>> par=[X,ones(size(X))]\Y;
par =
    0.5352
    0.9414

>> % Messpunkte und Ausgleichsgerade visualisieren
>> figure; hold on
>> plot(X,Y,'or')
>> intervall=[min(X) max(X)];
>> plot(intervall,par(1)*intervall+par(2))
```



5.1.5 Singulärwert-Zerlegung

`[U,S,V] = SVD(A)` bestimmt die Singulärwert-Zerlegung der $(m \times n)$-Matrix `A`. Die Diagonalelemente der $(m \times n)$-Rückgabematrix `S` bestehen aus den absteigend sortierten Wurzeln der

Eigenwerten von A^tA. Die Spalten der unitären Matrizen `U` bzw. `V` bestehen aus den Eigenvektoren von AA^t bzw. A^tA. Es gilt $A = USV^t$, also $S = U^tAV$.
Beispiel:

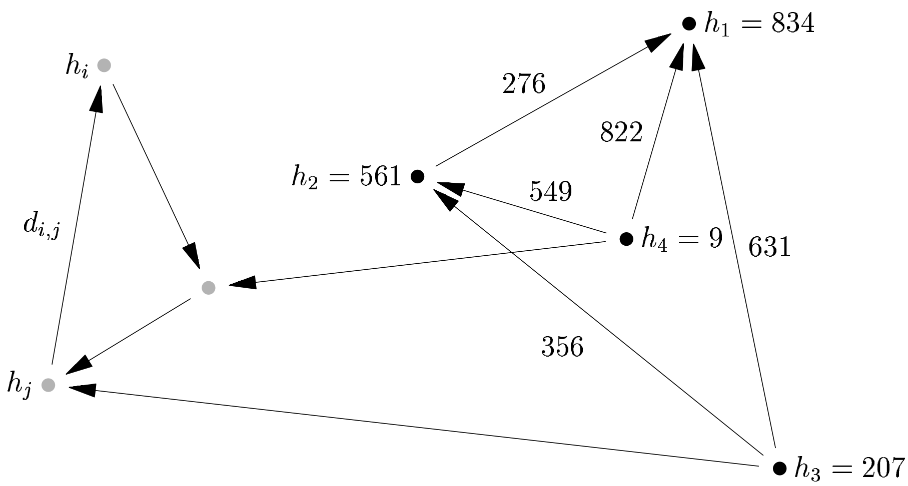
```
>> % Zu zerlegende Matrix
>> A = [0  1  1; -1  0  -1]
A =
     0     1     1
    -1     0    -1

>> [U,S,V]=svd(A)
U =
   -0.7071    0.7071
    0.7071    0.7071
S =
    1.7321         0         0
         0    1.0000         0
V =
   -0.4082   -0.7071   -0.5774
   -0.4082    0.7071   -0.5774
   -0.8165   -0.0000    0.5774

>> % Kontrolle der Zerlegung und U,V unitaer
>> U*S*V'-A
ans =
   1.0e-15 *
   -0.0679         0   -0.1110
    0.2220    0.1795    0.1110

>> U*U'-eye(2)
ans =
   1.0e-15 *
    0.2220   -0.0785
   -0.0785         0

>> V*V'-eye(3)
ans =
   1.0e-15 *
   -0.2220   -0.1825   -0.0818
   -0.1825   -0.1110   -0.1741
   -0.0818   -0.1741   -0.1110

>> % Kontrolle Eigenvektoren
>> [U,E1]=eig(A*A')
U =
   -0.7071   -0.7071
   -0.7071    0.7071
E1 =
     1     0
```

```
     0     3

>> [V,E2]=eig(A'*A)
V =
    0.5774    0.7071    0.4082
    0.5774   -0.7071    0.4082
   -0.5774         0    0.8165
E2 =
   -0.0000         0         0
         0    1.0000         0
         0         0    3.0000
```

Beispiel: Korrektur von Höhendaten / Pseudoinverse
Gemessen wurden topographische Höhendaten h_i und Höhendifferenzen $d_{i,j}$. Aufgrund von Messfehlern gilt in der Regel $d_{i,j} \neq h_i - h_j$. Gesucht werden Höhenkorrekturen x_i mit

$$\sum_{(i,j)\in\mathcal{J}} \Big(d_{i,j} - \big((h_i + x_i) - (h_j + x_j)\big)\Big)^2 \longrightarrow \min \ .$$

Zur Illustration der Vorgehensweise wird ein Modellproblem mit wenigen Daten mit Matlab gelöst.



Für die Höhen und Differenzwerte

```
>> h=[834, 561, 207, 9]';
>> d=[276, 631, 822, 356, 549]';
```

mit $d = (d_{1,2}, d_{1,3}, d_{1,4}, d_{2,3}, d_{2,4})^t$, sei

$$A := \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} , \qquad A(h+x) = d \ .$$

Das überbestimmte Gleichungssystem $Ax = d - Ah$ wird mit der Pseudoinversen gelöst.

```
>> A=[1 -1 0 0 ; 1 0 -1 0 ; 1 0 0 -1 ; 0 1 -1 0 ; 0 1 0 -1];
>> AP=pinv(A);
>> x=AP*(d-A*h)
x =
    1.0000
   -1.0000
   -3.0000
    3.0000
```

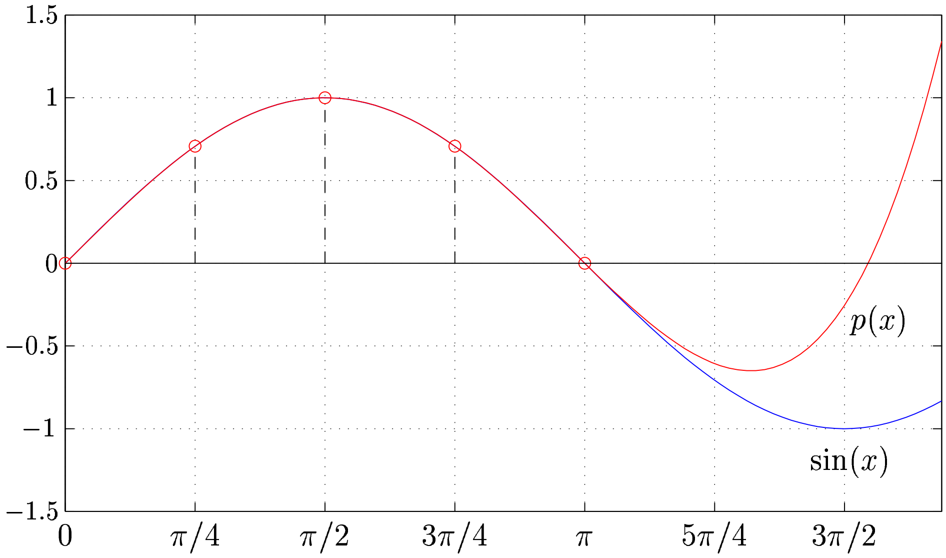
Man beachte, dass A keinen vollen Rang hat. Mit der Pseudoinversen wird dann die Ausgleichslösung minimaler Norm bestimmt. Dies ist für die betrachtete Anwendung sinnvoll, denn es soll eine möglichst kleine Korrektur bestimmt werden.

5.2 Analysis

5.2.1 Polynomiale Interpolation

`p=polyfit(x,y,n)` bestimmt zu den Datenvektoren `x` und `y` mit den Stützstellen (x_i, y_i) die Koeffizienten `p` $= (p_n, \ldots, p_0)$ des interpoliernenden Polynoms $p(x) = p_n x^n + \ldots + p_1 x + p_0$ vom Grad $\leq$`n`. Entspricht `n` der um 1 verminderten Länge von `x`, so ist die Interpolation exakt. Ist `n` um mindestens 2 kleiner als die Länge von `x`, so wird ein quadratisches Ausgleichsproblem gelöst.

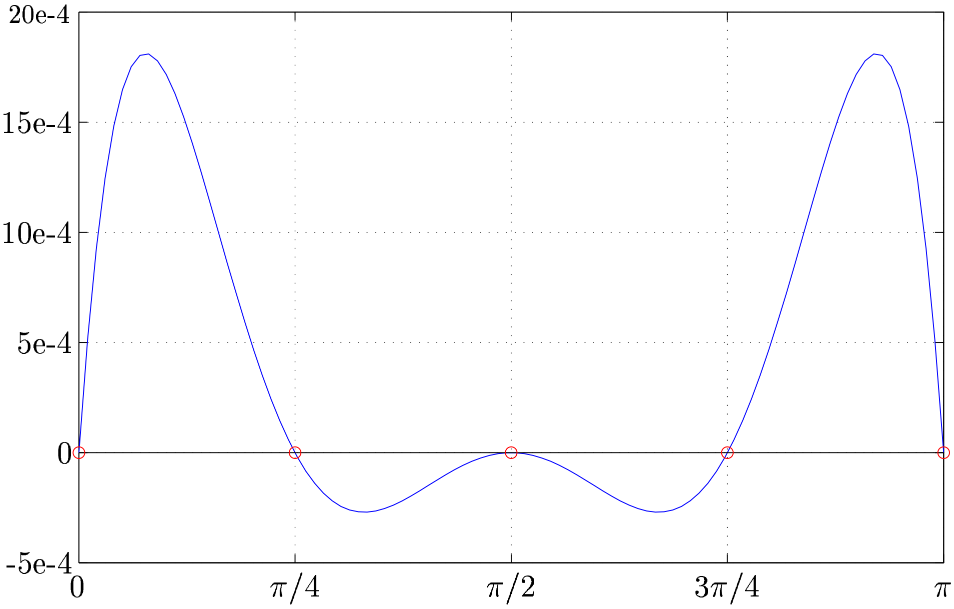
`y=polyval(p,x)` ermittelt die Funktionswerte `y` des Polynoms mit den Koeffizienten `p` an den Stellen `x`.
Beispiel:
Interpolation der Sinusfunktion bei `x`$= [0, \pi/4, \pi/2, 3\pi/4, \pi]$:




```
>> % Interpolationsdaten
>> x=[0:4]*pi/4;
>> y=[0, 1/sqrt(2), 1, 1/sqrt(2), 0];

>> % Interpoland ermitteln
```

```
>> p=polyfit(x,y,4)
p =
    0.0376   -0.2361    0.0583    0.9820    0.0000

>> % Interpoland auswerten
>> t=linspace(0,pi);
>> g=polyval(p,t);

>> % Interpolationsfehler sin(x)-p(x) darstellen
>> plot(t,sin(t)-g)
```

Interpolationsfehler:



5.2.2 Schnelle Fourier-Transformation

`FFT(X)` berechnet die diskrete Fourier-Transformation des Vektors `X`.
`IFFT(X)` bestimmt die inverse diskrete Fourier-Transformation des Vektors `X`.
Beispiel: Multiplikation von Polynomen
Bestimmung der Koeffzienten des Produktpolynoms

$$p(x) := \sum_{k=0}^{m+n} p_k x^k = \underbrace{\left(\sum_{k=0}^{m} u_k x^k\right)}_{:=u(x)} \cdot \underbrace{\left(\sum_{k=0}^{n} v_k x^k\right)}_{:=v(x)}$$

mit Hilfe der schnellen Fourier-Transformation.
Exemplarisch für die Koeffizientenvektoren $u = (3, 1)$ und $v = (-1, 0, 4)$:
Länge auf nächste Zweierpotenz 4 erweitern

```
>> u=[ 3  1  0  0 ];
>> v=[-1  0  4  0 ];
```

Polynome jeweils mit der schnellen Fouriertransformation an den Punkten $x_k = \exp(2\pi i k/4)$ auswerten

```
>> fu=fft(u)
fu =
   4.0000    3.0000 - 1.0000i    2.0000    3.0000 + 1.0000i
>> fv=fft(v)
fv =
     3    -5     3    -5
```

Funktionswerte punktweise multiplizieren und Koeffizienten des Produktpolynoms mit inverser Fourier-Transformation bestimmen

```
>> p=ifft( fu .* fv )
p =
    -3    -1    12     4
```

5.2.3 Integration

`F=trapz(X,Y)` bestimmt mit Hilfe der Trapezregel näherungsweise das Integral zu den in `X,Y` gespeicherten Funktionswerten (x_k, y_k) über dem Intervall $[\min(X), \max(X)]$. Der Abszissenvektor `X` muss aufsteigend sortiert sein.

`F=quad(function,xmin,xmax)` bestimmt mit Hilfe der adaptiven Simpson-Regel das Integral der durch `function` gegebenen Funktion im Intervall $[\mathrm{xmin}, \mathrm{xmax}]$. `function` kann dabei der Name einer Funktion oder ein Matlab-Ausdruck in einer Variablen sein und muss zu einem Vektor von Argumenten einen gleich langen Vektor von Funktionswerten zurückliefern.

`F=quadl(function,xmin,xmax)` verwendet die adaptive Lobatto-Quadratur.

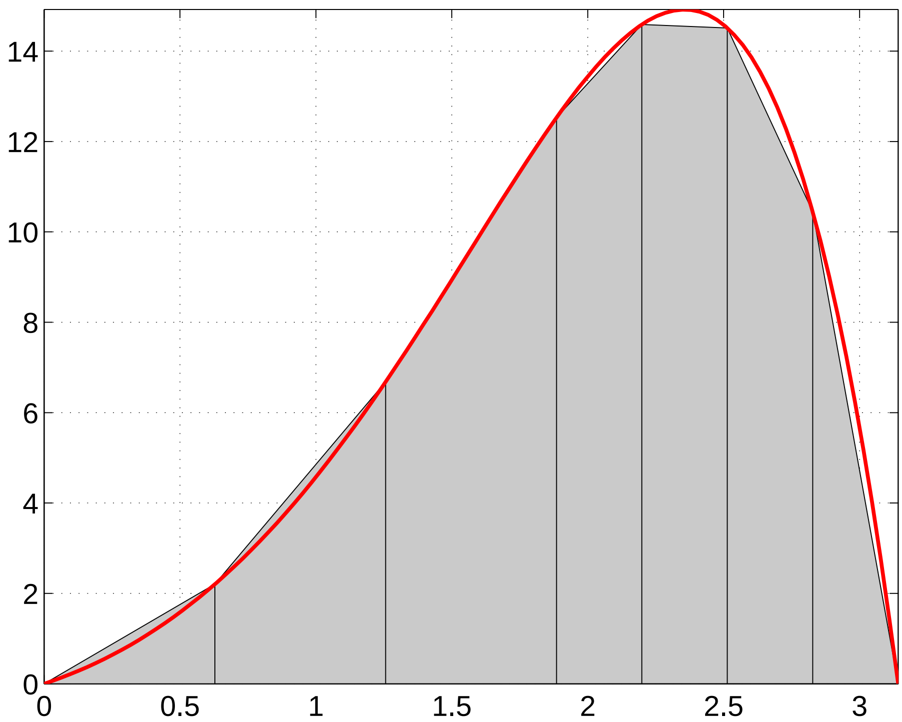
`V=dblquad(function,xmin,xmax,ymin,ymax)` bestimmt das zweidimensionale Integral über dem Gebiet $[\mathrm{xmin}, \mathrm{xmax}] \times [\mathrm{ymin}, \mathrm{ymax}]$.

Beispiel:
Numerische Bestimmung von

$$\int_0^\pi 2\sin(x)\mathrm{e}^x\,dx \quad = \quad \mathrm{e}^\pi + 1 \quad \approx \quad 24.1406926\,.$$

Trapezregel:

```
>> % Integrationsdaten
>> x=[ 0 .2 .4 .6:.1:1 ]*pi;
>> y=2*sin(x).*exp(x);

>> % relativer Fehler Trapezregel (rund 1%)
>> F=trapz(x,y)
F =
   23.9117
>> exakt=exp(pi)+1;
>> e=abs(F-exakt)/exakt
e =
    0.0095
```



Integrationsmethoden höherer Ordnung:

```
>> % relativer Fehler Simpson adaptiv
>> F=quad('2*sin(x).*exp(x)',0,pi)
F =
   24.1407
>> e=abs(F-exakt)/exakt
e =
   3.4291e-10

>> % relativer Fehler Lobatto adaptiv
>> F=quadl('2*sin(x).*exp(x)',0,pi)
F =
   24.1407
>> e=abs(F-exakt)/exakt
e =
   1.0743e-14
```

2D-Integration: Bestimmung der Fläche des Einheitskreises durch Integration der charakteristischen Funktion

$$\chi(x,y) := \begin{cases} 1 & \text{für } x^2+y^2 \leq 1 \\ 0 & \text{sonst} \end{cases}$$

auf dem Quadrat $[-1,1] \times [-1,1]$.

```
>> % Integration und relativer Fehler
>> F=dblquad('x.^2+y.^2<=1',-1,1,-1,1)
F =
    3.1416
>> e=abs(F-pi)/pi
e =
   5.9949e-06
```

5.2.4 Gewöhnliche Differenzialgleichungen

`[T,Y]=ode45(odefun,tspan,y0)` löst das in expliziter Form gegebene Anfangswertproblem

$$y' = f(t, y), \qquad y(t_0) = y_0 ,$$

auf dem durch `tspan` gegebenen Lösungsintervall $[t_0, t_1]$. `odefun` enthält den Namen oder den Zeiger der Funktion, welche den Funktionswert der rechten Seite f für die Parameterwerte t und y als Spaltenvektor zurückgibt, `y0` den Anfangswert.

In `T` und `Y` werden die Parameter- und die zugehörigen Funktionswerte zurückgegeben. Beim Aufruf ohne Rückgabeargumente wird die Lösung graphisch dargestellt.

Hinweis: Neben `ode45` existieren noch die syntaktisch gleichen Löser `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t` und `ode23tb`, die sich durch die verwendeten Lösungsalgorithmen unterscheiden.

Beispiel: Anfangswertproblem und Richtungsfeld

Lösung der Differenzialgleichung

$$y' = \frac{y}{2} \sin\left(t + t^2\right)$$

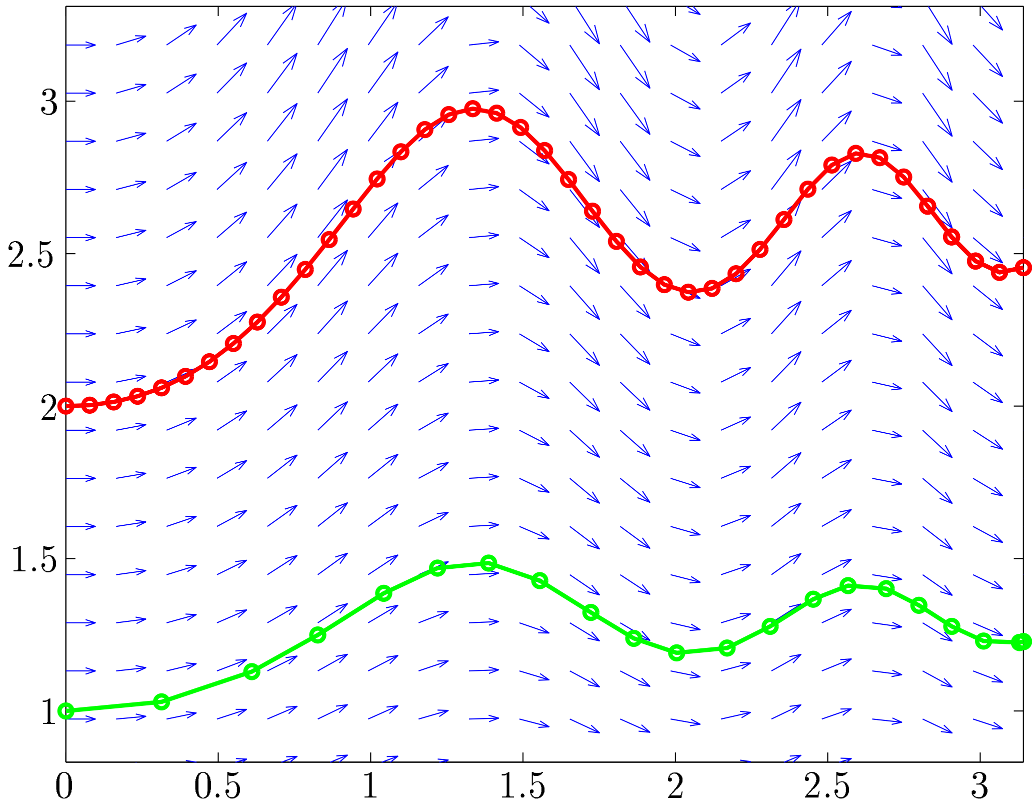
für die Anfangswerte $y(0) = 1$ und $y(0) = 2$:

Auswertungs-Funktion `odefun.m` der rechten Seite:

```
function ys=odefun(t,y)
ys=sin(t+t.^2).*y/2;
```

Visualisierung des Richtungsfeldes und der Lösungen der beiden Anfangswertprobleme für $t \in [0, \pi]$:

```
>> % Richtungsfeld plotten
>> [t,y]=meshgrid(linspace(0,pi,20),linspace(.5,3.5,20));
>> ys=odefun(t,y);
>> quiver(t,y,ones(size(ys)),ys);
>> hold on

>> % Anfangswertprobleme loesen
>> ode23(@odefun,[0,pi],1)
>> ode45('odefun',[0,pi],2)
```



Beispiel: Lorenz-Differenzialgleichung

Lösung der Lorenz-Differenzialgleichung

$$
\begin{aligned}
y_1' &= -10y_1 + 10y_2 \\
y_2' &= -y_1y_3 + 28y_1 - y_2 \\
y_3' &= y_1y_2 - 8/3y_3 \qquad .
\end{aligned}
$$

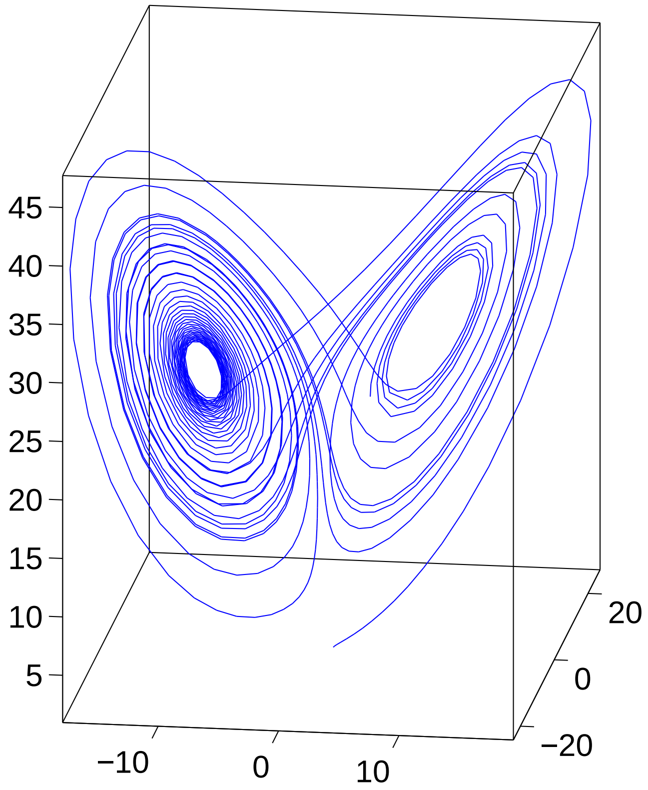
mit dem Anfangswert $y(0) = (1, 1, 1)$.

Auswertungsfunktion `odesys.m` der rechten Seite:

```
function ys=odesys(t,y)
ys=[ 10*(-y(1)+y(2));
    -y(1)*y(3)+28*y(1)-y(2);
     y(1)*y(2)-8/3*y(3) ];
```

Lösung für $t \in [0, 33]$ bestimmen und plotten:

```
>> [T,Y]=ode23(@odesys,[0,33],[1 1 1]);
>> plot3(Y(:,1),Y(:,2),Y(:,3))
>> view(8,16)
```


45
40
35
30
25
20
15
10
5
−10
0
10
20
0
−20