

---

# Projektseminar

---

Einführung in Matlab

Ali Kanso M.Sc.

ZeMA gGmbH

Zentrum für Mechatronik und Automatisierungstechnik

Saarbrücken, 15.05.2020

# Gliederung

---

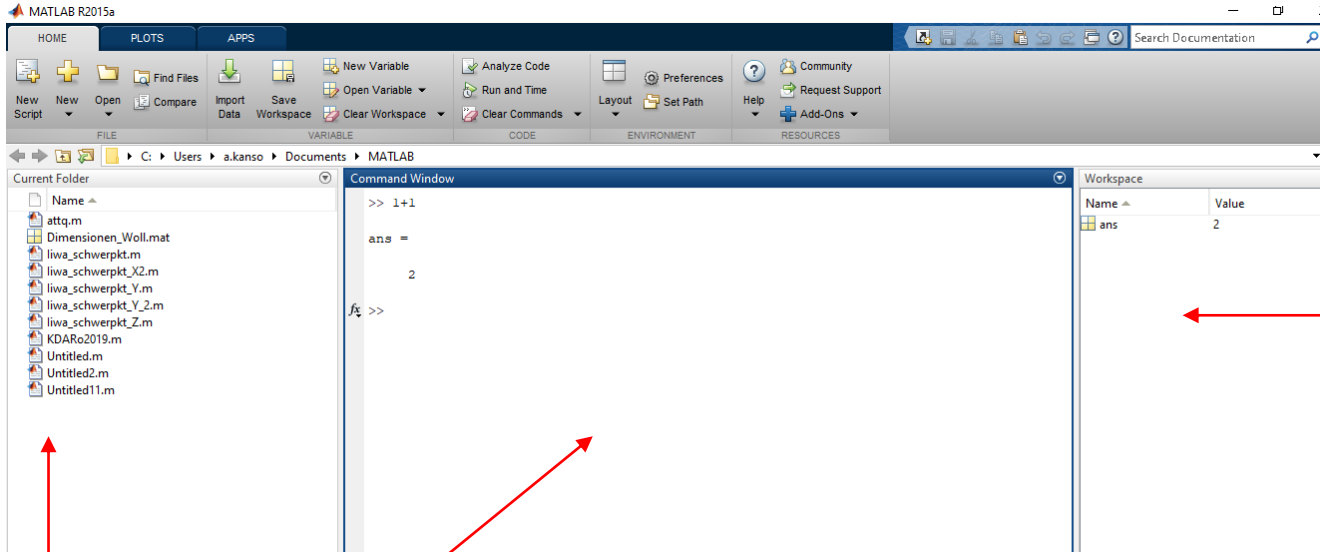
- 1** Einstieg in Matlab
- 2** Vektor bzw. Matrizen Rechnung
- 3** Indizierung und Doppelpunktnotation
- 4** Operatoren und Funktionen
- 5** Operatoren und Flusskontrolle
- 6** M-Dateien
- 7** Grafiken mit MATLAB erstellen
- 8** Graphical User Interface GUI

# Gliederung

---

- 1** Einstieg in Matlab
- 2** Vektor bzw. Matrizen Rechnung
- 3** Indizierung und Doppelpunktnotation
- 4** Operatoren und Funktionen
- 5** Operatoren und Flusskontrolle
- 6** M-Dateien
- 7** Grafiken mit MATLAB erstellen
- 8** Graphical User Interface GUI

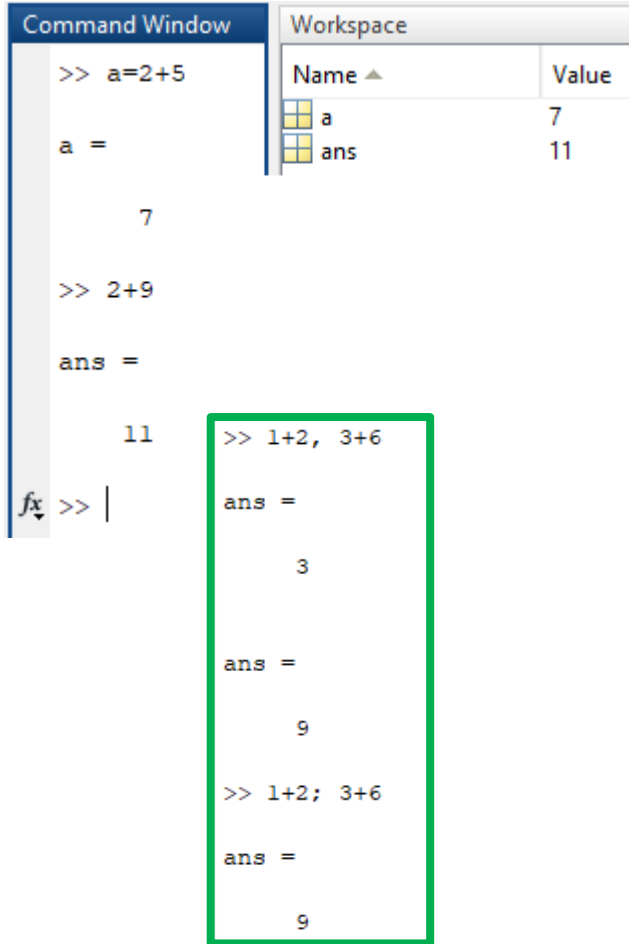
# Die Oberfläche



## ■ Drei Fenster

- **Command Windows** Hier werden die Befehle und Variablenzuweisungen eingegeben.
  - Test: `5+4`, `help ans`
- **Current Folder**: Das ist das aktuelle Verzeichnis, standardmäßig ist das MATLAB unter EigeneDateien. Beim ersten Arbeiten mit MATLAB ist das Verzeichnis üblicherweise leer. Hier findet man Dateien mit den Endungen `.m` (sogenannte *M-Files*) bzw. `.mat` (abgespeicherte Variable oder Parameter).
- **Workspace**: Hier sind die momentan benutzen Variablen und Parameter zu sehen.

# Grundrechenarten



The screenshot shows the MATLAB Command Window and Workspace. The Command Window displays the following commands and outputs:

```
>> a=2+5  
  
a =  
  
7  
  
>> 2+9  
  
ans =  
  
11
```

The Workspace window shows two variables:

Name	Value
a	7
ans	11

A green box highlights a sequence of commands and outputs in the Command Window:

```
>> 1+2, 3+6  
  
ans =  
  
3  
  
ans =  
  
9  
  
>> 1+2; 3+6  
  
ans =  
  
9
```

## ■ Grundrechenarten:

- Addition: +
- Subtraktion: -
- Multiplikation: \*
- Division: /
- Potenzbildung: ^

## ■ Variablen müssen nicht gesondert deklariert

- Sie werden durch eine Wertzuweisung erzeugt, und ihre Werte können anderen Variablen übertragen werden.

## ■ Das Ergebnis eines Befehls wird standardmäßig direkt ausgegeben. Dies kann unterdrückt werden durch Abschließen des Befehls mit einem Semikolon.

- → Wird das Ergebnis eines Befehls nicht in einer Variable gespeichert, wird es automatisch in die Variable ans geschrieben.

## ■ Mehrere Befehle können durch Kommata (mit Ausgabe der Ergebnisse) oder Semikolon (ohne Ausgabe der Ergebnisse) getrennt in eine Zeile geschrieben werden. Hierbei wird nur das letzte Resultat in ans gespeichert.

## ■ Bemerkung: Komplex Zahlen werden durch a+ib eingegeben

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Einfache Funktionen

```
Command Window
>> f = @(x) x^2 + sin(x^2+pi/2)

f =

@(x) x^2+sin(x^2+pi/2)
```

```
Command Window
>> help elfun
Elementary math functions.

Trigonometric.
sin      - Sine.
sind     - Sine of argument in degrees.
sinh     - Hyperbolic sine.
asin     - Inverse sine.
asind    - Inverse sine, result in degrees.
asinh    - Inverse hyperbolic sine.
cos      - Cosine.
cosd     - Cosine of argument in degrees.
cosh     - Hyperbolic cosine.
acos     - Inverse cosine.
acod     - Inverse cosine, result in degrees.
acosh    - Inverse hyperbolic cosine.
tan      - Tangent.
tand     - Tangent of argument in degrees.
tanh     - Hyperbolic tangent.
atan     - Inverse tangent.
atand    - Inverse tangent, result in degrees.
atan2    - Four quadrant inverse tangent.
atan2d   - Four quadrant inverse tangent, result in degrees.
atanh    - Inverse hyperbolic tangent.
sec      - Secant.
secd     - Secant of argument in degrees.
sch      - Hyperbolic secant.
asec     - Inverse secant.
asecd    - Inverse secant, result in degrees.
asech    - Inverse hyperbolic secant.
csc      - Cosecant.
csd      - Cosecant of argument in degrees.
sch      - Hyperbolic cosecant.
acsc     - Inverse cosecant.
acsd     - Inverse cosecant, result in degrees.
asch     - Inverse hyperbolic cosecant.
cot      - Cotangent.
cotd     - Cotangent of argument in degrees.
coth     - Hyperbolic cotangent.
acot     - Inverse cotangent.
acotd    - Inverse cotangent, result in degrees.
acoth    - Inverse hyperbolic cotangent.
hypot    - Square root of sum of squares.

Exponential.
exp      - Exponential.
expm1    - Compute exp(x)-1 accurately.
log      - Natural logarithm.
logh     - Compute log(1+x) accurately.
```

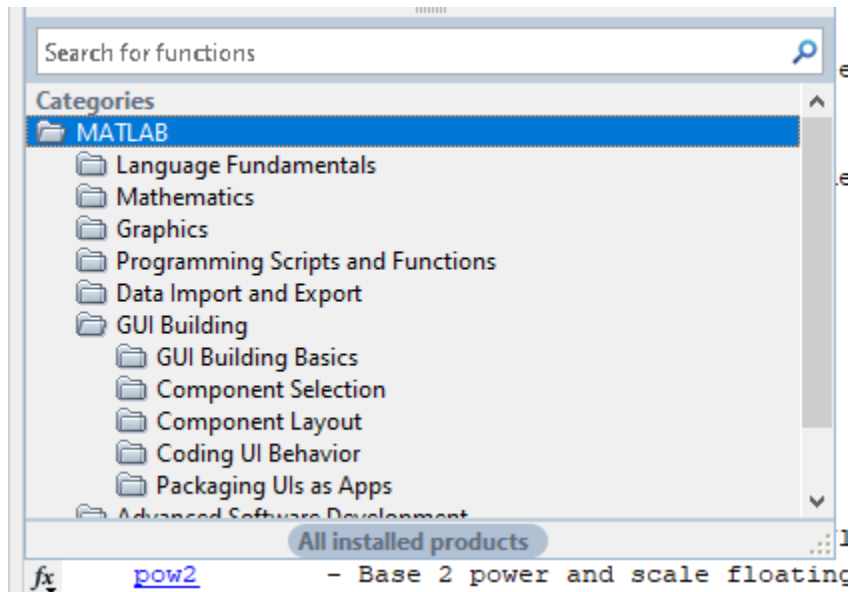
Einfache Funktionen sind dabei solche, die man schnell mal im Command Window auf Zahlen oder Vektoren anwenden kann, komplizierte Funktionen sind solche, die man besser separat in einer sogenannten .m-Datei erklärt, um sie erst dann im Command Window aufzurufen.

- Einfache Funktionen Bsp.:
  - Sinusfunktion:  $\sin(\pi/2)$
  - Kosinusfunktion:  $\cos(\pi)$ 
    - → bei trigonometrischen Funktionen ist den Winkel in Radian zu geben
  - Exponentialfunktion:  $\exp()$
  - Logarithmusfunktion:  $\log()$
  - Wurzelfunktion:  $\sqrt{\phantom{x}}$
- **Achtung: MATLAB unterscheidet zwischen Groß- und Kleinschreibung.**
- Weitere einfache Funktionen, die man im Command Window erklären und anwenden kann, sind Polynomfunktionen oder Komposita einfacher Funktionen. Solche Funktionen kann man im Command Window problemlos als sogenannte *anonyme Funktionen* erklären.
  - Bsp1.:  $f = @(x) x^2 + \sin(x^2+\pi/2)$ 
    - $f(0)=1$
  - Bsp2.:  $g = @(x,y) x^2 + \sin(y^2+\pi/2)$

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Einfache Funktionen

- Neben diesen elementaren Funktionen findet man viele weitere Funktionen durch Anklicken des Symbols *fx* links von der aktuellen Eingabezeile im Command Window.
  - Die Funktionsweise dieses *Function Browser*s erklärt sich von selbst; man erhält eine Beschreibung der Funktionen, indem man den Mauszeiger über die entsprechende Auswahl platziert.



Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Bemerkungen

---

```
>> format long
>> pi

ans =

    3.141592653589793

>> pi

ans =

    3.141592653589793

>> sin(0.5)

ans =

    0.479425538604203

>> format short
>> sin(0.5)

ans =

    0.4794

>> format rat
>> sin(0.5)

ans =

    501/1045

>> pi

ans =

    355/113
```

- clc: damit leert man das Command Window.
- clear: damit entfernt man alle Variablen im Workspace. Natürlich können auch einzelne Variable gelöscht werden. So entfernt etwa clear a die Variable a.
- Strg+C: bricht MATLAB (meistens) ab, falls Sie z. B. eine Endlosschleife erzeugt haben.
- help: damit ruft man die Hilfe auf, z. B. help sin oder help clear.
- Falls die help-Hilfe nicht ausreichend ist, so greife man auf doc zurück, z. B. doc sin.
- Zahlenformate in MATLAB: Mit format kann das Zahlenformat geändert werden:
  - format short: das ist standardmäßig voreingestellt, z. B. 0,3212.
  - format long: z. B. 0,321234276512387.
  - format rat: MATLAB rechnet mit rationalen Zahlen, z. B. e= 1457/536.

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

---



# Gliederung

---

- 1 Einstieg in Matlab
- 2 **Vektor bzw. Matrizen Rechnung**
  - 2.1 Doppelpunkt Operator
- 3 Indizierung und Doppelpunktnotation
- 4 Operatoren und Funktionen
- 5 Operatoren und Flusskontrolle
- 6 M-Dateien
- 7 Grafiken mit MATLAB erstellen
- 8 Graphical User Interface GUI

# Vektoren bzw. Matrizen erzeugen

```
>> zeros(3,4)
```

```
ans =
```

```
    0    0    0    0
    0    0    0    0
    0    0    0    0
```

```
>> ones(2,3)
```

```
ans =
```

```
    1    1    1
    1    1    1
```

```
>> eye(3)
```

```
ans =
```

```
    1    0    0
    0    1    0
    0    0    1
```

```
>> rand(2,3)
```

```
ans =
```

```
    0.2785    0.9575    0.1576
    0.5469    0.9649    0.9706
```

- Wir fassen Spaltenvektoren als einspaltige Matrizen und Zeilenvektoren als einzeilige Matrizen auf. Somit können wir uns auf Matrizen beschränken.
- Matrizen können auf mehrere Arten erzeugt werden. Viele Typen von Matrizen können direkt über eine MATLAB-Funktion generiert werden:
  - die Nullmatrix, die Einheitsmatrix und Einsmatrizen
- Können über die Funktionen `zeros`, `eye` und `ones` erzeugt werden.
- Alle haben die gleiche Syntax. Zum Beispiel erzeugt
  - `zeros(m,n)` oder `zeros([m,n])` eine  $m \times n$  Nullmatrix, während `zeros(n)` eine  $n \times n$  Nullmatrix erzeugt.
- Mit `rand` erzeugt man Matrizen mit Pseudozufallszahlen als Einträge. Die Syntax ist die gleiche wie bei `eye`. Ohne Argument gibt die Funktion eine einzelne Zufallszahl zurück.

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Vektoren bzw. Matrizen erzeugen

```
>> diag([2 3 4],1)
```

```
ans =
```

```
    0    2    0    0
    0    0    3    0
    0    0    0    4
    0    0    0    0
```

```
>> diag([2 3 4],-1)
```

```
ans =
```

```
    0    0    0    0
    2    0    0    0
    0    3    0    0
    0    0    4    0
```

```
>> diag([2 3 4],0)
```

```
ans =
```

```
    2    0    0
    0    3    0
    0    0    4
```

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

```
A =
```

```
    2    5    6
    1    2    3
    8    8    7
```

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

```
A =
```

```
    2    5    6
    1    2    3
    8    8    7
```

- Mit der Funktion `diag` können Diagonalmatrizen angelegt werden. Für einen Vektor `x` erzeugt `diag(x)` eine Diagonalmatrix mit der Diagonale `x`.
- Allgemeiner legt `diag(x,k)` `x` auf die `k`-te Diagonale:
  - $k > 0$  Diagonalen über der Hauptdiagonalen
  - $k < 0$  die darunter ( $k = 0$  bezeichnet die Hauptdiagonale).
- Matrizen können explizit über die Klammernotation (square bracket notation) erzeugt werden.
  - Innerhalb einer Zeile können einzelne Elemente über ein Leerzeichen oder Komma getrennt werden.
  - Achtung: bei der Angabe von Vorzeichen für die einzelnen Einträge kein Leerzeichen zwischen Vorzeichen und Element : → MATLAB interpretiert das Vorzeichen sonst als Plus oder Minus Operator.

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Vektoren bzw. Matrizen erzeugen

Command Window

```
>> B=[3 5; 2 8]
```

B =

```
3    5
2    8
```

```
>> C=[B zeros(2); eye(2) ones(2)]
```

C =

```
3    5    0    0
2    8    0    0
1    0    1    1
0    1    1    1
```

```
>> repmat(B,4,4)
```

ans =

```
3    5    3    5    3    5    3    5
2    8    2    8    2    8    2    8
3    5    3    5    3    5    3    5
2    8    2    8    2    8    2    8
3    5    3    5    3    5    3    5
2    8    2    8    2    8    2    8
```

- Häufig sehr praktisch ist das Erzeugen von Matrizen durch Angabe von Blöcken, anstatt der einzelnen Elemente.
- Blockdiagonalmatrizen können noch einfacher direkt über die Funktion `blkdiag` erzeugt werden

```
>> A=blkdiag(B,ones(2))
```

A =

```
3    5    0    0
2    8    0    0
0    0    1    1
0    0    1    1
```

- Für "getäfelte" Blockmatrizen eignet sich `repmat`: `repmat(A,m,n)` erzeugt eine Block- $m \times n$ -Matrix, in der jeder Block eine Kopie von A ist. Wird n weggelassen, wird der Wert standardmäßig auf m gesetzt.

# Spezielle Matrizen

```
>> a=hilb(3)

a =

    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000

>> b=invhilb(3)

b =

     9    -36     30
   -36    192   -180
     30   -180    180

>> b*a

ans =

    1.0000    0.0000    0.0000
    0.0000    1.0000   -0.0000
   -0.0000     0     1.0000
```

- MATLAB verfügt über Funktionen um ganz spezielle Matrizen zu erzeugen.
  - Bsp.:
    - Hilbermatrizen:  $a_{i,j} = \frac{1}{i+j-1}$
    - Die Matrix wird durch den Befehl `hilb` erzeugt, und ihre Inverse (die nur ganzzahlige Komponenten hat!) über `invhilb`.
- Quadratische  $n \times n$  Matrizen die nur aus den Zahlen  $1, \dots, n^2$  bestehen, deren Zeilen- und Spaltensummen und Summe der Einträge der Diagonalen gleich ist nennt man *magische Quadrate*.
  - Diese können mit MATLAB durch die Funktion `magic` erzeugt werden.

```
>> magic(3)

ans =

     8     1     6
     3     5     7
     4     9     2
```

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Gliederung

---

- 1** Einstieg in Matlab
- 2** Vektor bzw. Matrizen Rechnung
  - 2.1** Doppelpunkt Operator
- 3** Indizierung und Doppelpunktnotation
- 4** Operatoren und Funktionen
- 5** Operatoren und Flusskontrolle
- 6** M-Dateien
- 7** Grafiken mit MATLAB erstellen
- 8** Graphical User Interface GUI

# Doppelpunkt Operator

```
>> 1:1:6

ans =

     1     2     3     4     5     6

>> 10:-1.3:6

ans =

 10.0000   8.7000   7.4000   6.1000

>> 1:2:7

ans =

     1     3     5     7

Did you mean:
>> linspace(0,1,3)

ans =

     0   0.5000   1.0000

>> linspace(0,1,4)

ans =

     0   0.3333   0.6667   1.0000
```

- Der Doppelpunkt Operator ist einer der wichtigsten Operatoren in MATLAB und findet in vielen Fällen Verwendung.
- Punkte gleichen Abstands zwischen a und b. Der Standardwert für n ist 100. Mit seiner Hilfe können spezielle Zeilenvektoren erzeugt werden, die z.B. bei der Indizierung in for Schleifen oder beim Plotten verwendet werden.
  - Dabei wird ausgehend von einer Zahl solange eine Einheit addiert und in dem Vektor gespeichert, bis ein vorgegebenes Ende erreicht oder überschritten wurde. Die allgemeine Syntax ist:
    - `<Start>:<Ende>` oder `<Start>:<Increment>:<Ende>`.
- `linspace(a,b,n)` erzeugt n Punkte gleichen Abstands zwischen a und b. Der Standardwert für n ist 100.

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Gliederung

---

- 1 Einstieg in Matlab
- 2 Vektor bzw. Matrizen Rechnung
- 3 Indizierung und Doppelpunktnotation**
- 4 Operatoren und Funktionen
- 5 Operatoren und Flusskontrolle
- 6 M-Dateien
- 7 Grafiken mit MATLAB erstellen
- 8 Graphical User Interface GUI



# Indizierung und Doppelpunktnotation

- Die Indizierung von Feldern erfolgt in MATLAB durch runde Klammern und startet beim Index 1.
  - Bei Matrizen steht der erste Index für die Zeilen-, der zweite für die Spaltennummer des Elements.
  - **Es gibt bei MATLAB keine nicht positiven Indizes!**
- Ein Vorteil von MATLAB gegenüber anderen Programmiersprachen ist, dass nicht nur auf die einzelnen Elemente eines Feldes zugegriffen werden kann, sondern auf beliebige Teilblöcke.

```
>> A=10*round(rand(4,4)*10)/10
```

```
A =
```

7	2	5	1
7	9	10	10
5	2	1	0
1	8	4	8

```
>> B=A([1,2],3)
```

```
B =
```

5
10

```
>> B=A([1,2],[2,3])
```

```
B =
```

2	5
9	10

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

```
B =
```

2	1
9	10

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

```
B =
```

2	1
9	10

```
>> B=A([1:3],[2:end])
```

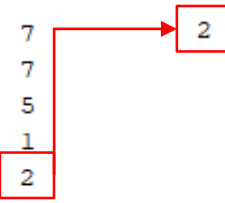
```
B =
```

2	5	1
9	10	10
2	1	0

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Indizierung und Doppelpunktnotation

```
A =  
  
7     2     5     1  
7     9    10    10  
5     2     1     0  
1     8     4     8  
  
>> A(:)    >> A(5)  
  
ans =      ans =  
  
7  
7  
5  
1  
2  
9  
2  
8  
5  
10  
1  
4  
1  
10  
0  
8
```



- MATLAB speichert alle Felder intern als Spaltenvektor, Matrizen spaltenweise von der ersten zur letzten Spalte. Auf diese Repräsentation kann durch Angabe nur eines Indizes zugegriffen werden, dies wird häufig *lineares Indizieren* genannt.
- Wird A(:) auf der linken Seite einer Zuweisung benutzt, so wird damit A gefüllt, unter Beibehaltung der Struktur von A.
  - Mit dieser Notation ergibt sich eine Möglichkeit, auf folgende Weise eine 3 × 3 – Matrix der ersten 9 Primzahlen zu erzeugen:

```
>> A=zeros(3)    >> A(:)=primes(23)  
  
A =              A =              >> A=primes(23)  
  
0     0     0      2     7     17    A =  
0     0     0      3    11     19  
0     0     0      5    13     23      2     3     5     7    11    13    17    19    23
```

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Indizierung und Doppelpunktnotation

---

A =

9	4	2	9
6	1	2	9
4	2	4	5
5	1	0	5

```
>> A(sub2ind(size(A),1:4,1:4))=10
```

A =

10	4	2	9
6	10	2	9
4	2	10	5
5	1	0	10

- Eine weitere Funktion die beim Indizieren hilfreich ist, ist `sub2ind`. Diese berechnet aus Matrixindizes lineare Indizes.
  - Hiermit können wir z.B. die Diagonale der obigen Matrix A mit 10 überschreiben.

# Gliederung

---

- 1 Einstieg in Matlab
- 2 Vektor bzw. Matrizen Rechnung
- 3 Indizierung und Doppelpunktnotation
- 4 Operatoren und Funktionen**
- 5 Operatoren und Flusskontrolle
- 6 M-Dateien
- 7 Grafiken mit MATLAB erstellen
- 8 Graphical User Interface GUI

# Operatoren

```
A =  
  
    10     4     2     9  
     6    10     2     9  
     4     2    10     5  
     5     1     0    10  
  
>> A'  
  
ans =  
  
    10     6     4     5  
     4    10     2     1  
     2     2    10     0  
     9     9     5    10
```

- Um ein Feld zu transponieren, stellt MATLAB den Operator ' zur Verfügung
- Rechenoperatoren:
  - MATLAB unterstützt das Rechnen mit Vektoren und Matrizen. So können zwei Felder gleicher Dimensionen einfach durch + addiert und mit - subtrahiert werden
  - MATLAB interpretiert den Multiplikationsoperator \* als Matrixprodukt oder als Multiplikation mit einem Skalar. Bei ersterem muss die Anzahl der Spalten des ersten Arguments gleich der Anzahl der Zeilen des zweiten Argumentes sein. Daneben gibt es noch den elementweisen Multiplikationsoperator.\*.

```
>> x=1:3  
  
x =  
  
     1     2     3  
  
>> y=2:4  
  
y =  
  
     2     3     4
```

```
>> x*y  
Error using *  
Inner matrix dimensions must agree.  
  
>> x.*y  
  
ans =  
  
     2     6    12  
  
>> 4*x  
  
ans =  
  
     4     8    12
```

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Operatoren

```
X =  
     6     1     2     4  
  
>> X./X  
ans =  
     1     1     1     1  
  
>> 3./X  
ans =  
  0.5000  3.0000  1.5000  0.7500  
  
>> X./3  
ans =  
  2.0000  0.3333  0.6667  1.3333
```

- Auch das Potenzieren  $^$  wird im Sinne des Matrixprodukt interpretiert, analog zur Multiplikation gibt es auch die „gepunktete“ Version  $.^$ .
- Die Division gibt es in zwei Ausführungen: den Slash  $/$  und den Backslash  $\backslash$ .
  - Beide entsprechen dem (ggf. approximativen) Lösen eines linearen Gleichungssystems.  $B/A$  steht ungefähr für  $BA^{-1}$ ,  $A\backslash B$  für  $A^{-1}B$ .
  - Oder  $BA^{-1} = B \cdot \text{inv}(A)$
  - Natürlich gibt es auch wieder das elementweise dividieren  $./$ , dieses muss auch eingesetzt werden, wenn ein Skalar durch einen Vektor geteilt wird

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Basisfunktion

---

```
>> X=round(rand(1,4)*10)
```

```
X =
```

```
      8      0      0      2
```

```
>> length(X)
```

```
ans =
```

```
      4
```

```
>> size(X)
```

```
ans =
```

```
      1      4
```

```
>> numel(X)
```

```
ans =
```

```
      4
```

- Um Eigenschaften wie die Länge oder Dimensionen von Feldern abzufragen, gibt es in MATLAB eine Hand voll Funktionen:

- length Gibt die Länge eines Vektors zurück, bei einer Matrix wird die größere der beiden Dimensionen zurückgegeben.
- size Gibt die Dimensionen einer Matrix zurück.
- numel Gibt die Anzahl der Elemente einer Matrix zurück,  $\text{numel}(A) == \text{prod}(\text{size}(A)) == \text{length}(A(:))$

# Funktionen

```
>> x=linspace(-pi/2, pi/2,5)

x =

    -1.5708    -0.7854         0     0.7854     1.5708

>> sin(x)

ans =

    -1.0000    -0.7071         0     0.7071     1.0000
```

```
>> X=round(rand(1,4)*10)    >> diff(X)

X =                          ans =

     2     2     2     4         0     0     2

>> X=round(rand(1,4)*10)    >> prod(X)

X =                          ans =

     3     9     4     2         360

>> X=round(rand(1,4)*10)    >> cumsum(X)

X =                          ans =

     9    10     4     1         9    19    23    24

>>
>> sum(X)

ans =

    24
```

- MATLAB verfügt über unzählige Funktionen die Vektoren und Matrizen als Eingabe erwarten.
  - Grundsätzlich verändert eine Funktion in MATLAB kein Eingabeargument (Eingabeargumente werden als Wert und nicht als Referenz übergeben).
  - Viele dieser Methoden kann man einer von drei Gruppen zuordnen:
    - Skalare Funktionen
    - Vektorfunktionen
    - Matrixfunktionen
- Vektorwertige Funktionen operieren auf Vektoren und geben ein Skalar oder einen Vektor zurück.
  - sum, max, prod, diff, cumsum, sort
- Viele vektorwertige Funktionen unterstützen die Übergabe eines zweiten Parameters dim, der die Dimension (1=Spalten, 2=Zeilen) angibt, über welche die Funktion ausgeführt werden soll.
  - Damit ist es also z. B. auch einfach möglich, die Summe der Zeilen von A zu berechnen: sum(A,2)
  - Möchte man diese Funktion nicht auf die Spalten einer Matrix A sondern auf die ganze Matrix -als Vektor aufgefasst- anwenden, übergibt man A(:) als Eingabeargument sum(A(:))

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung



# Funktionen

- Matrixfunktionen erwarten als Eingabe eine Matrix. Beispiele sind `norm`, `chol`, `svd`, `triu` und `tril`. Die letzten
  - beiden Funktionen extrahieren aus einer gegebenen Matrix eine obere, bzw. untere Dreiecksmatrizen.
  - Allgemein gibt `tril(A,k)` (bzw. `triu(A,k)`) die Elemente auf und unter (bzw. ober) der k-ten Diagonale zurück.
  - `chol`: Cholesky factorization
  - `svd`: singular value decomposition

X =

1	1	5	9
3	7	8	3
3	1	7	7
7	7	9	2

>> tril(X)

ans =

1	0	0	0
3	7	0	0
3	1	7	0
7	7	9	2

>> tril(X,4)

ans =

1	1	5	9
3	7	8	3
3	1	7	7
7	7	9	2

>> tril(X,0)

ans =

1	0	0	0
3	7	0	0
3	1	7	0
7	7	9	2

>> tril(X,1)

ans =

1	1	0	0
3	7	8	0
3	1	7	7
7	7	9	2

>> triu(X,0)

ans =

1	1	5	9
0	7	8	3
0	0	7	7
0	0	0	2

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Matrizen manipulieren

---

- Die Funktion `reshape` ändert die Dimensionen einer Matrix: `reshape(A,m,n)` erzeugt eine  $m \times n$  Matrix, deren Elemente spaltenweise aus `A` entnommen werden.
- Die Notation `[]` steht für eine leere  $0 \times 0$ -Matrix. Weist man einer Zeile oder Spalte einer Matrix den Wert `[]` zu, so wird sie aus der Matrix gelöscht.
- An einen vorhandenen Vektor können einfach weitere Elemente angehängt werden, indem man einem Index der größer als die Länge des Vektors ist, einen Wert zuweist. Der Vektor wird daraufhin automatisch bis zu dem entsprechenden Index verlängert und mit Nullen gefüllt.

# Gliederung

---

- 1 Einstieg in Matlab
- 2 Vektor bzw. Matrizen Rechnung
- 3 Indizierung und Doppelpunktnotation
- 4 Operatoren und Funktionen
- 5 Operatoren und Flusskontrolle**
- 6 M-Dateien
- 7 Grafiken mit MATLAB erstellen
- 8 Graphical User Interface GUI

# Relationale Operatoren

```
>> B=round(10*rand(3))
```

```
B =
```

```
     1     3     8
     5     6     3
    10     2     5
```

```
>> A=round(10*rand(3))
```

```
A =
```

```
     7     5     3
     9     1     8
    10     1     3
```

```
>> A==B
```

```
ans =
```

```
     0     0     0
     0     0     0
     1     0     0
```

- Die relational Operatoren von Matlab sind:
  - ==: gleich; ~=: ungleich
  - <: weniger als; >: größer als
  - <= kleiner oder gleich; >= größer oder gleich
- Beachten Sie, dass in MATLAB ein einzelnes Gleichheitszeichen = eine Zuweisung angibt und nie auf Gleichheit testet.
- Vergleiche zwischen Skalaren erzeugen logisch 1 wenn die Relation wahr und logisch 0 wenn sie falsch ist.
  - Vergleiche sind auch zwischen Matrizen gleicher Dimension definiert und zwischen Matrizen und Skalaren, deren Ergebnis in beiden Fällen eine Matrix mit Nullen und Einsen ist.
  - Für Matrix-Matrix Vergleiche werden die entsprechenden Paare von Elementen verglichen, während für Matrix-Skalar Vergleiche der Skalar mit jedem Matrixelement verglichen wird.
- Um zu testen, ob zwei Matrizen A und B gleich sind, kann der Ausdruck `isequal(A,B)` verwendet werden.
  - Für eine Liste aller dieser Funktionen rufen sie in MATLAB doc is auf. Die Funktion `isnan` ist besonders wichtig, da der Test `x == NaN` immer das Ergebnis 0 (falsch) liefert, selbst wenn `x = NaN`

# Logische Operatoren

---

```
>> A>0 & B>0
```

```
ans =
```

```
1     1     1
1     1     1
1     1     1
```

```
>> all(A(:)==B(:))
```

```
ans =
```

```
0
```

- Die logische Operatoren von Matlab sind:
  - &: logisches und
  - |: logisches oder
  - ~: logisches nicht
  - xor: logisches exklusive oder
  - all: wahr wenn alle Elemente eines Vektor von Null verschieden sind
  - any: wahr wenn wenigstens ein Element eines Vektors von Null verschieden ist
- Wie die relationalen Operatoren produzieren &,| und ~ Matrizen von Nullen und Einsen, falls eines der Argumente eine Matrix ist.
  - Mit all(A(:)==B(:)) können die beiden Matrizen A und B auf Gleichheit getestet werden.

# Flusskontrolle

---

```
>> if a<10, a=a+1, end
```

```
a =
```

```
1
```

```
>> a=0;
```

```
>> b=-1;
```

```
>> if a==0 & b<0
```

```
c=10
```

```
elseif a==0&b>0
```

```
c=20
```

```
end
```

```
c =
```

```
10
```

## ■ Matlab hat vier Strukturen für die Flußkontrolle:

- Die if-Abfrage
- Die for-Schleife
- Die while-Schleife
- Den switch-Befehl

## ■ Die If-Abfrage

- if expression  
statements
- end
- Folgen auf einen if-Befehl auf der gleichen Zeile weitere Befehle, so müssen diese durch ein Komma getrennt werden, um das if-Kommando vom nächsten Befehl zu trennen
- Befehle die nur ausgeführt werden sollen, wenn expression falsch ist, können nach else eingefügt werden
- Schließlich kann mit elseif ein weiterer Test hinzugefügt werden mit (beachte, dass zwischen else und if kein Leerzeichen stehen darf)
- Bei einer if-Abfrage der Form if Bedingung 1 & Bedingung 2 wird Bedingung 2 nicht ausgewertet, wenn Bedingung 1 falsch ist (dies wird "early return" if Auswertung genannt).

---

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Flusskontrolle

```
>> y=0;  
>> for i=1:3  
y=y+i  
end
```

```
y =  
  
1
```

```
y =  
  
3
```

```
y =  
  
6
```

```
>> x=0;  
while 1,...,end  
x=x+1  
break
```

```
y=2  
end
```

```
z=3
```

```
x =  
  
1
```

```
z =  
  
3
```

## ■ Die for-Schleife

- for variable = ausdruck  
statements
- end

## ■ Normalerweise ist ausdruck ein Vektor der Form i:s:j. Die Befehle werden für jedes Element von ausdruck ausgeführt, wobei variable dem entsprechenden Element von ausdruck zugeordnet ist.

- Mehrere for-Schleifen können verschachtelt werden.

## ■ Die while-Schleife hat die Form

- While variable = ausdruck  
statements
- end
- Die Befehle werden ausgeführt, solange ausdruck war ist
- Eine while-Schleife kann mit dem Befehl break beendet werden, der die Kontrolle an den ersten Befehl nach dem entsprechenden end zurückgibt. Eine unendliche Schleife kann mit while 1, ..., end erzeugt werden.
- Der Befehl continue setzt die Ausführung sofort in der nächsten Iteration der for- oder while-Schleife fort, ohne die verbleibenden Befehle in der Schleife auszuführen.

# Flusskontrolle

```
>> p=2
x=1:4
switch p
case 1
y = sum(abs(x));
case 2
y = sqrt(x*x')
case inf
y = max(abs(x));
otherwise
error('p must be 1, 2 or inf.')
end

p =

     2

x =

     1     2     3     4

y =

    5.4772
```

## ■ Den switch-Befehl

- switch Ausdruck  
case Ausdruck Befehl  
otherwise Ausdruck (optional)
- end

- Der Ausdruck nach case kann eine Liste von Werten sein, die in geschweiften Klammern eingeschlossen ist (ein Cell-Array). In diesem Fall kann der switch-Ausdruck zu jedem Element der Liste passen.

```
clc
x = input('Enter a real number: '); switch x
case {inf,-inf}
disp('Plus or minus infinity')
case {1,2,3}
disp('1,2,3')
case 0
disp('Zero')
otherwise
disp('Nonzero and finite')
end
Enter a real number: 2
1,2,3
```

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung



# Gliederung

---

- 1 Einstieg in Matlab
- 2 Vektor bzw. Matrizen Rechnung
- 3 Indizierung und Doppelpunktnotation
- 4 Operatoren und Funktionen
- 5 Operatoren und Flusskontrolle
- 6 M-Dateien**
- 7 Grafiken mit MATLAB erstellen
- 8 Graphical User Interface GUI

# M-Dateien

- Eine M-Datei ist eine Textdatei mit der Dateiendung .m, die MATLAB Befehle enthält. Es gibt zwei Arten:
  - **Skriptdateien** (oder Kommandodateien) keine Ein- oder Ausgabeargumente und operieren auf Variablen im Workspace.
  - **Funktionsdateien** enthalten eine function Definitionszeile und akzeptieren Eingabeargumente und geben Ausgabeargumente zurück, und ihre internen Variable sind lokal auf die Funktion beschränkt

```
1 - close all
2 - clear all
3 - clc
4
5 - off=90
6 %% Input
7 %% p1
8 - q1=95*540/255 + 254*540/(255*255)+off ;% °
9
10 - q2=30*270/255 + 112*270/(255*255)-135;% °
11 - d3=10;% m
12 - q=[q1 ;q2; 0]; % Gelenk variable Degree °
13
14 - run Manipulator_defintion.m
15 - [RPY,D,r,T0_1,T0_2,T0_3,T0_4,T0_5,T0_6]=forwardkine
16 - T_P1=T0_2;
17 - q11=q1;
```

```
1 function [RPY,D,r,T0_1,T0_2,T0_3,T0_4,T0_5,T0_6]=forwardkinematic(DHParameter)
2
3 %% intialization
4 - Dz=eye(4,4);
5 - Sz=eye(4,4);
6 - Dx=eye(4,4);
7 - Sx=eye(4,4);
8 - T=eye(4,4);
9 - Ai = zeros(4,4,size(DHParameter,1));
10 - T_link=zeros(4,24);
11
12 %% Forward calculation
13 - p=1;
14 - for i = 1:size(DHParameter,1)
```

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Skriptdateien

---

- Ein Skript sammelt eine Folge von Befehlen, die wiederholt verwendet werden sollen oder in Zukunft noch gebraucht werden.
- Sobald MATLAB auf ein % trifft ignoriert es den Rest der Zeile. Dies erlaubt das Einfügen von Text, der das Skript für Menschen leichter verständlich macht.

```
3 %% initialization
4 - Dz=eye(4,4);
5 - Sz=eye(4,4);
6 - Dx=eye(4,4);
7 - Sx=eye(4,4);
8 - T=eye(4,4);
9 - Ai = zeros(4,4,size(DHParameter,1));
10 - T_link=zeros(4,24);
11
12 %% Forward calculation
13 - p=1;
14 - for i = 1:size(DHParameter,1)
15
```

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Funktionsdateien

```
1 function a= add2num(b,c)
2 -     a=b+c;
3 -     end
```

```
1 -     a= add2num(2,5)

Command Window

>> Untitled2

a =

    7
```

- Selbst geschriebene Funktionsdateien erweitern den Umfang von MATLAB.
- Sie werden auf die gleiche Weise verwendet wie die bereits existierenden MATLAB Funktionen wie sin, eye, size usw.
- Die erste Zeile fängt mit dem Schlüsselwort function an, gefolgt vom Ausgabeargument y, und dem Gleichheitszeichen. Rechts von = kommt der Funktionsname Myfunction, gefolgt vom Eingabeargument A in Klammern.
  - Im Allgemeinen kann es beliebig viele Ein- und Ausgabeargumente geben.
  - Der Funktionsname muss der gleiche sein wie der Name der M-Datei, die die Funktion enthält - in diesem Fall muss die Datei Myfunction heißen.
- Bsp: die Funktion add2num addiert zwei Zahlen
  - function a=add2num(b,c)
  - a=b+c;
  - end

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Optimierung von M-Files

```
1 - n = 5e5;  
2 - x = randn(n,1);  
3 - tic, s = 0;  
4 - for i=1:n  
5 -     s = s+x(i)^2;  
6 - end, toc  
7
```

Command Window

```
>> Untitled2  
Elapsed time is 0.487028 seconds.
```

```
1 - close all  
2 - clear all  
3 - clc  
4 - n = 5e5;  
5 - x = randn(n,1);  
6 - tic  
7 - t=sum(x.^2);  
8 - toc  
9  
10
```

Command Window

```
Elapsed time is 0.001971 seconds.
```

- Bei großen Problemen ist es sinnvoll und notwendig, Algorithmen effizient zu implementieren.
  - Durch die Beachtung einiger Regeln kann man enorme Zeitgewinne erreichen (siehe Bsp. links).
  - Offensichtlich wird in beiden Fällen die Summe der Quadrate der Elemente von x berechnet.
  - Der Zeitgewinn durch die vektorielle Implementierung ist allerdings phänomenal.

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Preallokierung von Speicher

```
1 - close all
2 - clear all
3 - clc
4
5 - n=1e7;
6 - tic
7 - x(1:2) = 1;
8 - for i=3:n
9 -     x(i)=0.25*x(i-1)^2-x(i-2);
10 - end
11 - toc
```

Command Window

Elapsed time is 2.372357 seconds.

```
12 - tic
13 - x=ones(n,1);
14 - for i=3:n
15 -     x(i)=0.25*x(i-1)^2-x(i-2);
16 - end
17 - toc
```

Command Window

Elapsed time is 0.250924 seconds.

- Eine bequeme Eigenschaft von MATLAB besteht darin, dass Arrays (Vektoren) vor ihrer Benutzung nicht deklariert werden müssen und man sich ins Besondere die Festlegung auf eine Größe (Höchstzahl der Elemente) spart.
  - Werden an ein vorhandenes Array neue Elemente angehängt, wird die Dimension automatisch angepasst. Dafür muss immer neuer Speicher belegt werden – in Extremfällen kann dieses Vorgehen deshalb zu Ineffizienz führen:
- Es ist zu erkennen, dass die arithmetischen Operationen im Vergleich zur Speicher-Allokierung einen verschwindend geringen Anteil an der Rechenzeit haben.

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Rekursives Programmieren

```
1 function ret_val=fak(n)
2 -   if (n<=1)
3 -       ret_val=1;
4 -   else
5 -       ret_val=n*(fak(n-1));
6 -   end
```

Command Window

```
>> fak(4)
```

```
ans =
```

```
24
```

- Als Rekursion bezeichnet man einen Aufruf einer Funktion durch sich selbst.
  - Am besten erklärt sich das an Hand eines Beispiels. Die Fakultät einer natürlichen Zahl  $n$  ist definiert durch:
    - $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$
    - In Zeile 5 ruft sich die Funktion 'fak' selbst auf, allerdings mit einem um 1 verminderten Argument.
- Rekursive Programmierung ist allerdings auch mit Bedacht zu wählen:
  - Speicheraufwendig
  - Kann langsamer als iterative Methoden sein.

# Gliederung

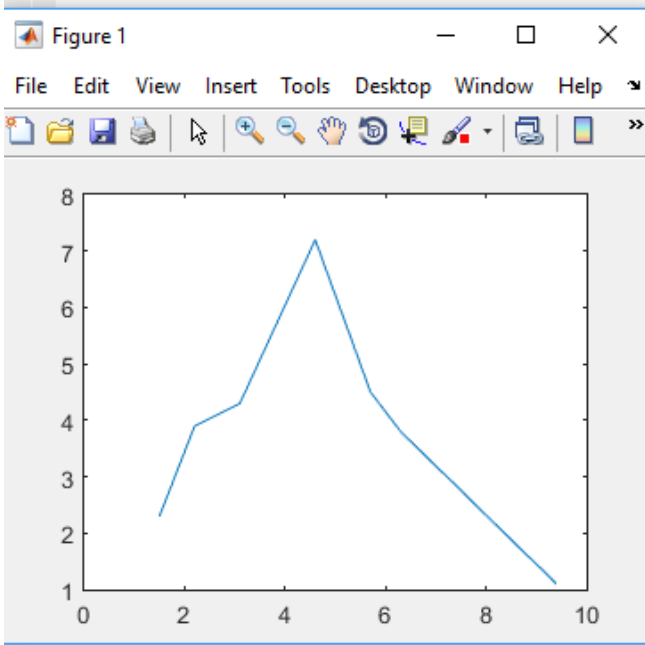
---

- 1 Einstieg in Matlab
- 2 Vektor bzw. Matrizen Rechnung
- 3 Indizierung und Doppelpunktnotation
- 4 Operatoren und Funktionen
- 5 Operatoren und Flusskontrolle
- 6 M-Dateien
- 7 Grafiken mit MATLAB erstellen**
- 8 Graphical User Interface GUI



# 2-dimensionale Plots

```
1 - close all
2 - clc
3 - clear
4 - x = [1.5 2.2 3.1 4.6 5.7 6.3 9.4];
5 - y = [2.3 3.9 4.3 7.2 4.5 3.8 1.1];
6 - plot(x,y)
```

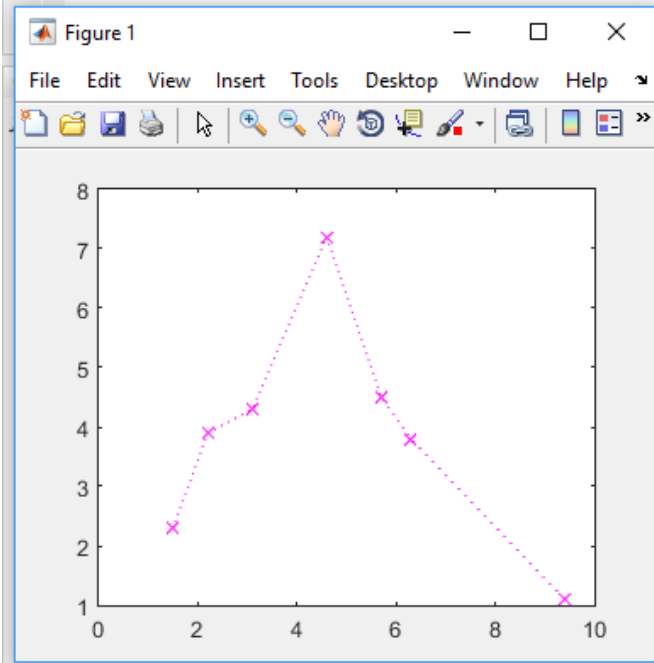


- Die einfachste Variante zweidimensionale Plots zu erzeugen, besteht darin zwei Vektoren mit gleichen Dimensionen zu koppeln.
  - Hat man zwei solcher Vektoren (x und y) gegeben, werden mit dem Befehl `plot(x,y)` die jeweiligen  $x(i)$  und  $y(i)$  als zusammengehörig erkannt und ein dementsprechender Plot erzeugt.
- Mit zusätzlichen Angaben kann man das Aussehen modifizieren. Ein allgemeinerer Aufruf der Funktion `plot` hat die Form `plot(x,y,String)`, wobei sich der String aus bis zu drei Angaben (Farbe, Knoten, Linienart) zusammensetzt.

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

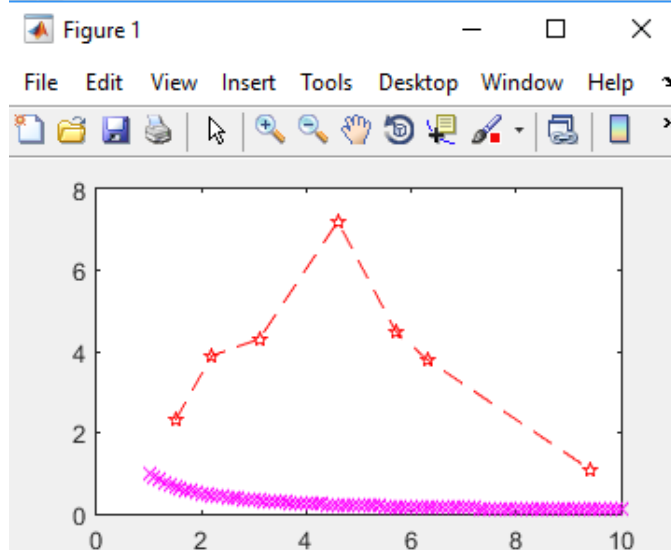
# 2-dimensionale Plots

```
1 - close all
2 - clc
3 - clear
4 - x = [1.5 2.2 3.1 4.6 5.7 6.3 9.4];
5 - y = [2.3 3.9 4.3 7.2 4.5 3.8 1.1];
6 - plot(x,y,'m:x')
```



- Ein typischer Aufruf hätte die Form `plot(x,y,'mx:')`, das gleiche Resultat erhält man auch mit `plot(x,y,'m:x')`, woran man sieht, dass die Reihenfolge der Angabe irrelevant ist.
- Es ist auch möglich mehr als einen Graphen in das Koordinatensystem zu legen:

```
4 - x = [1.5 2.2 3.1 4.6 5.7 6.3 9.4];
5 - y = [2.3 3.9 4.3 7.2 4.5 3.8 1.1];
6 - a = 1:1:10;
7 - b = 1./a;
8 - plot(x,y,'rp--',a,b,'mx:')
```



Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Parameter für das Aussehen eines Plots

---

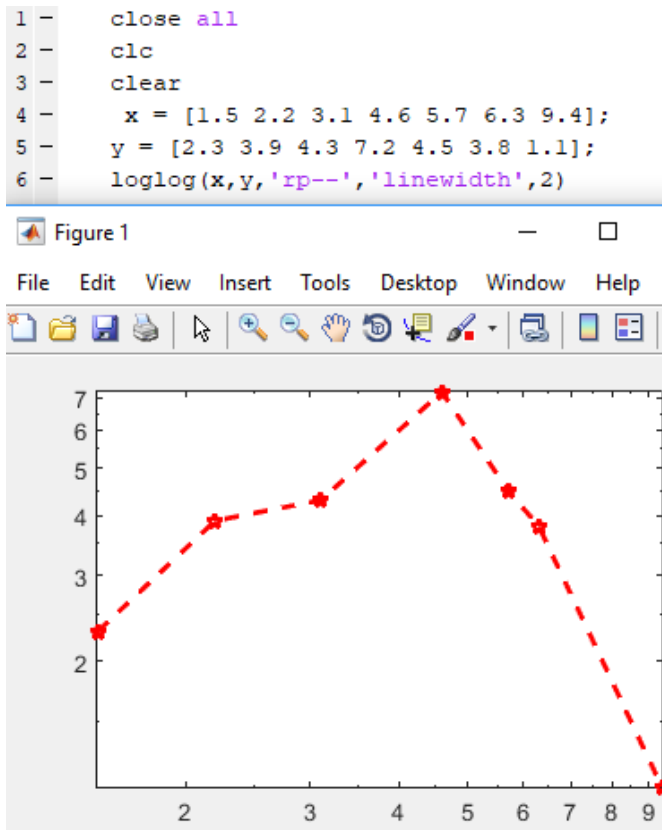
Farbe	
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

Marker	
o	Kreis
*	Stern
.	Punkt
+	Plus
x	Kreuz
s	Quadrat
d	Diamant
^	Dreieck nach oben
v	Dreieck nach unten
>	Dreieck nach rechts
<	Dreieck nach links
p	Fünfpunktstern
h	Sechspunktstern

Linienart	
-	durchgezogene Linie
-	gestrichelte Linie
:	gepunktete Linie
-.	gepunktet und gestrichelte Linie

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# 2-dimensionale Plots

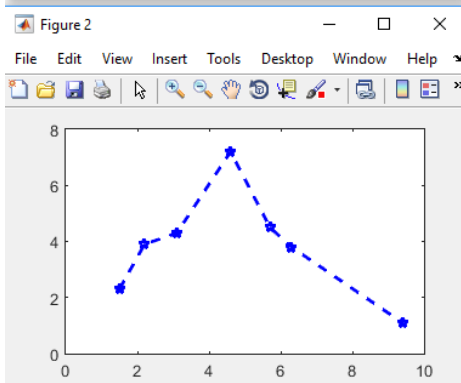
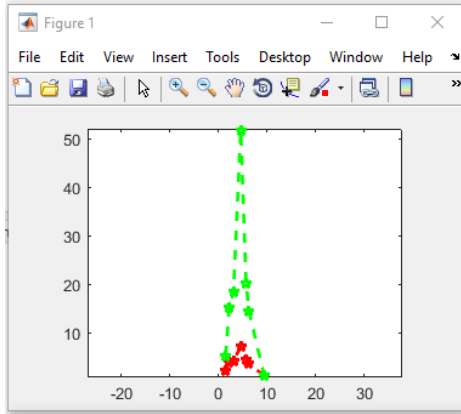


- Es werden allerdings nicht nur Vektoren als „Datenquellen“ akzeptiert, sondern auch Matrizen.
  - Wenn  $x$  ein  $m$ -dimensionaler Vektor und  $Y$  eine  $m \times n$ -Matrix ist, bewirkt `plot(x,Y)` das gleiche wie `plot(x,Y(:,1),x,Y(:,2),...,x,Y(:,n))`.
  - Wenn zusätzlich  $X$  auch eine  $m \times n$  Matrix ist, kann man mit `plot(X,Y)` die jeweils korrespondierenden Spalten der beiden Matrizen zu einem Graph verbinden, es handelt sich also um eine verkürzte Schreibweise von `plot(X(:,1),Y(:,1),...,X(:,n),Y(:,n))`.
- In Fall der Komplexen Zahlen wird der imaginäre Teil ignoriert. Allerdings gibt es eine Ausnahme:
  - Übergibt man `plot` als einziges Argument eine komplexe Matrix  $Z$ , erhält man das identische Resultat wie mit dem Kommando `plot(real(Z),imag(Z))`.
- `plot` können weitere Attribute übergeben werden, beispielsweise `LineWidth`: `plot(x,y,'linewidth',2)`
- Für manche Anwendungen ist es sinnvoller, die Achsen logarithmisch zu skalieren. Dazu verwendet man statt `plot` eine der folgenden Anweisungen:
  - `semilogx(x,y)`, die  $x$ -Achse wird logarithmisch skaliert
  - `semilogy(x,y)`, die  $y$ -Achse wird logarithmisch skaliert
  - `loglog(x,y)`, beide Achsen werden logarithmisch skaliert

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Nützliche Befehle

```
x = [1.5 2.2 3.1 4.6 5.7 6.3 9.4];  
y = [2.3 3.9 4.3 7.2 4.5 3.8 1.1];  
figure(1)  
plot(x,y,'rp--','linewidth',2)  
hold on  
plot(x,y.^2,'gp--','linewidth',2)  
axis equal  
figure(2);  
plot(x,y,'bp--','linewidth',2)  
hold on
```

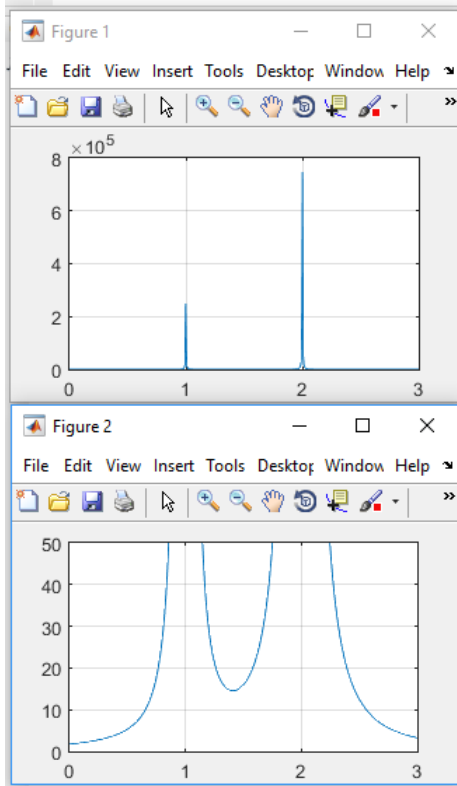


- **hold on:** sorgt dafür, dass beim Plotten von weiteren Funktionen die bisherigen erhalten bleiben und nicht - wie standardmäßig eingestellt bzw. bei hold off – zunächst das figure-Fenster „geleert“ und erst dann der neue Plot eingefügt wird.
- Mit dem Befehl figure wird ein neues figure-Fenster geöffnet, mit figure(n) kann man ein beliebiges figure-Fenster aktivieren, so dass sich alle folgenden (Grafik-) Befehle auf dieses beziehen.
- Viele Eigenschaften der Achsen können mit dem axis-Kommando beeinflusst werden:
  - axis([xmin xmax ymin ymax])
  - axis auto
  - axis equal
  - axis off
  - axis square; axis tight

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Nützliche Befehle

```
4 - x = linspace(0,3,500);
5 - figure
6 - title('myTitle')
7 - plot(x,1./(x-1).^2+3./(x-2).^2)
8 - grid on
9 - figure
10 - title('myTitle')
11 - plot(x,1./(x-1).^2+3./(x-2).^2)
12 - ylim([0 50])
13 - grid on
```



- **Titel:** `title('myTitel')` nach `plot` Befehl
- **Für den Fall, dass man nur die Grenzen einer Achse festlegen möchte:**
  - stehen die Befehle `xlim([xmin,xmax])` und `ylim([ymin,ymax])` zur Verfügung.
  - Sollte man sich an einem Ende des Intervalls nicht festlegen wollen oder können, sorgt die Angabe von `-inf` bzw. `inf` dafür, dass sich MATLAB darum kümmert.
- **Um Text einzufügen:**
  - `text(x,y,'string')`, wobei  $(x, y)$  die Koordinate festlegt, an der der Text beginnen soll.
- **Legend:**
  - `legend('string1', ..., 'stringn', Position)`, die Reihenfolge und die Anzahl der übergebenen Strings sollte mit der im `plot`- bzw `loglog`- Befehl übereinstimmen.
  - Mögliche Werte für Position:
    - -1: rechts des Plots
    - 0: MATLAB entscheidet
    - 1: rechts oben; 2: links oben
    - 3: links unten; 4: rechts unten

Quelle: Christian Karpfinger und Boris von Loesh, Matlab – Eine Einführung

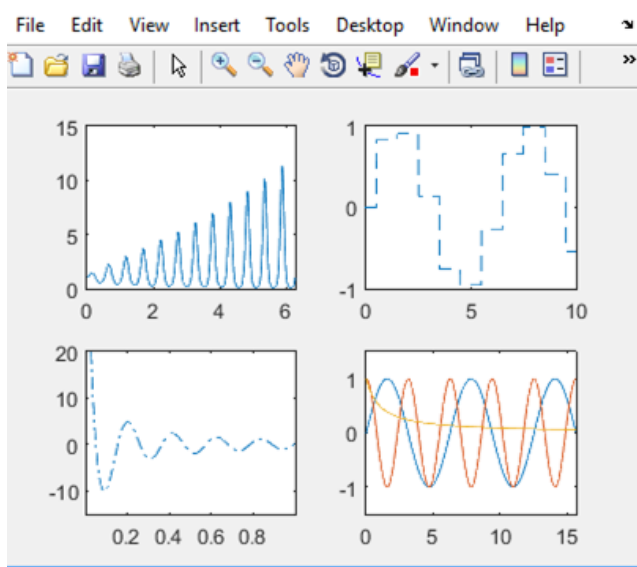
# Nützliche Befehle

---

- Es ist möglich (und teilweise auch notwendig) in Strings TEX-Code zu benutzen
- Um zum Beispiel folgenden Text am Punkt  $(-0.1, 0.2)$  in einer Grafik einzufügen:
  - $(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$  schreibt man idealerweise:
  - `text(-.1,.2,'(n+1)P_{n+1}(x)=(2n+1)xP_n(x)-nP_{n-1}(x)')`

# Mehrere Plots in einem Fenster

```
subplot(2,2,1), fplot('exp(sqrt(x)*sin(12*x))',[0 2*pi]);  
subplot(2,2,2), fplot('sin(round(x))',[0 10], '--');  
subplot(2,2,3), fplot('cos(30*x)/x',[0.01 1 -15 20], '-.');  
subplot(2,2,4), fplot('sin(x), cos(2*x), 1/(1+x)',[0 5*pi -1.5 1.5])
```



- Der Befehl `subplot(m,n,p)` teilt das figure-Fenster in `m` Zeilen und `n` Spalten auf. `p` ist ein Skalar und bezeichnet das aktive Feld.

– Auch eine unregelmäßige Aufteilung des figure-Feldes ist möglich:

```
x = linspace(0,15,100);  
subplot(2,2,1), plot(x,sin(x))  
subplot(2,2,2), plot(x,round(x))  
subplot(2,2,3:4), plot(x, sin(round(x)))
```

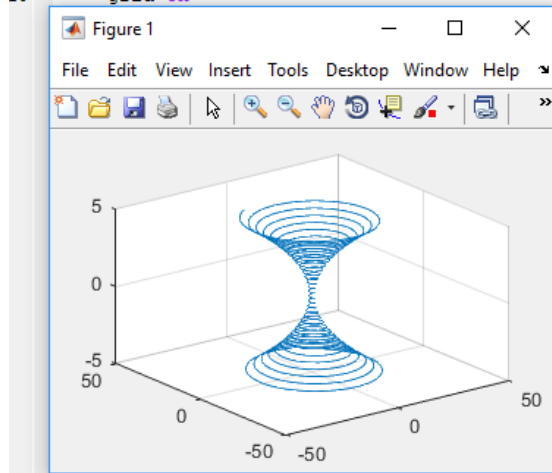
- `fplot`: MATLAB wertet dabei die Funktion an „genug“ Stellen aus, um einen realitätsnahen Graph zu erhalten
- Natürlich können auch `fplot` weitere Parameter übergeben werden. Mehr dazu findet man - wie zu jeder anderen Funktion auch - unter `doc fplot`

Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

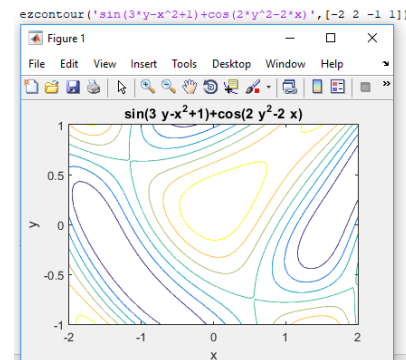


# 3-dimensionale Plots

```
5 - t = -5:.005:5;  
6 - x = (1+t.^2).*sin(20*t);  
7 - y = (1+t.^2).*cos(20*t);  
8 - z = t;  
9 - plot3(x,y,z)  
10 - grid on
```



- Analog zum Befehl `plot` im zweidimensionalen existiert auch der Befehl `plot3` zum Plotten von Kurven in 3-D. Im Grunde funktioniert er genauso.
- `plot3` kann nur zur Visualisierung von Kurven  $f: \mathbb{R} \rightarrow \mathbb{R}^3$  genutzt werden, nicht aber von Funktionen  $g: \mathbb{R}^2 \rightarrow \mathbb{R}$ , anschaulich gesprochen werden ausschließlich Linien und nie Flächen dargestellt.
- Einen ersten Ausweg aus diesem Dilemma bietet der Befehl `ezcontour`, der für eine Funktion  $f: [xmin, xmax] \times [ymin, ymax] \rightarrow \mathbb{R}$  eine Höhenkarte erzeugt, man `ezcontour('f',[xmin xmax ymin ymax]);` ausführt.

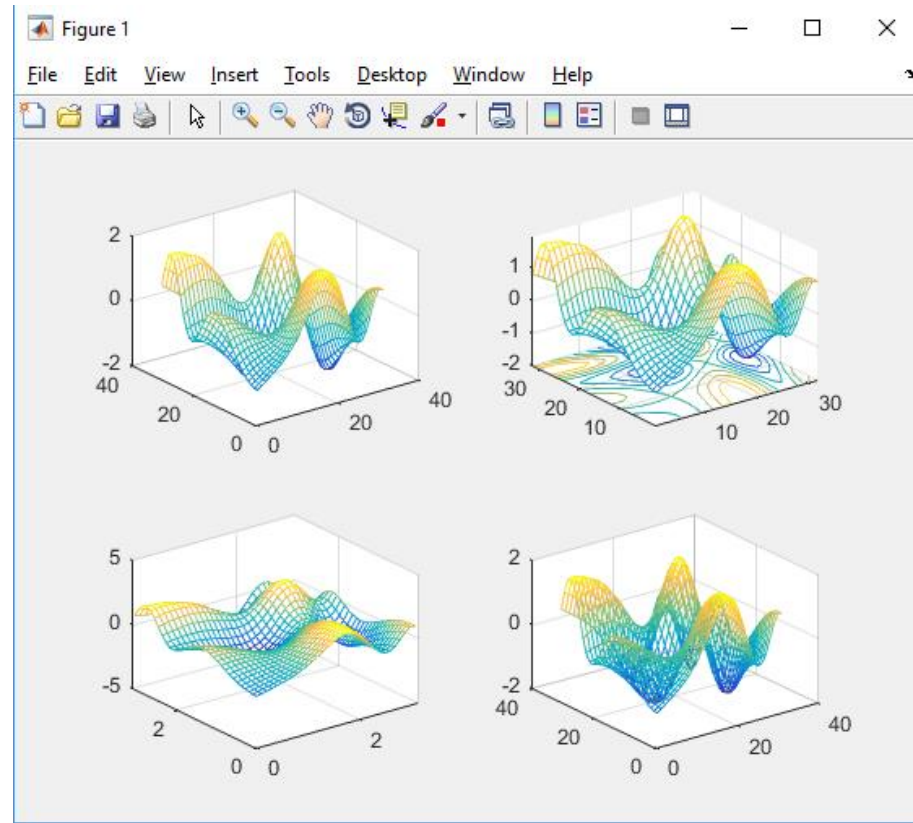


Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# 3-dimensionale Plots

- Die Befehle `mesh` und `meshc` liefern dreidimensionale Plots, wie wir sie uns vorstellen, `meshc` legt in die x-y-Ebene zusätzlich eine Höhenkarte

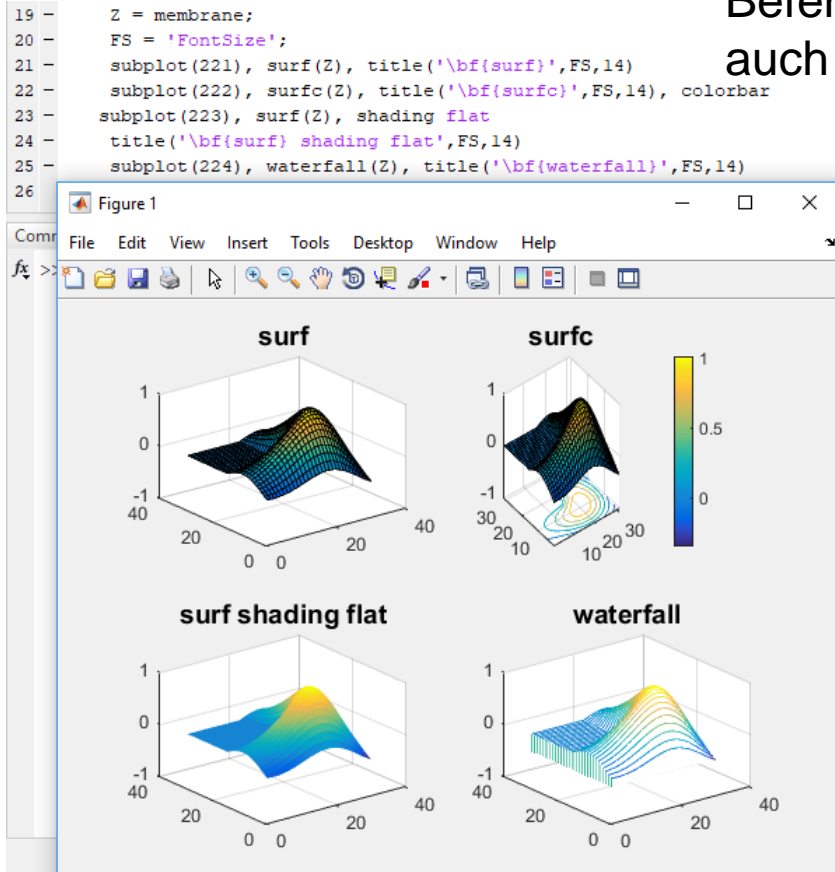
```
5 - x = 0:.1:pi; y = 0:.1:pi;  
6 - [X,Y] = meshgrid(x,y);  
7 - Z = sin(Y.^2+X) - cos(Y-X.^2);  
8 - subplot(221)  
9 - mesh(Z)  
10 - subplot(222)  
11 - meshc(Z)  
12 - subplot(223)  
13 - mesh(x,y,Z)  
14 - axis([0 pi 0 pi -5 5])  
15 - subplot(224)  
16 - mesh(Z)  
17 - hidden off
```



Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# 3-dimensionale Plots

- Es fällt auf, dass nur netzartige Strukturen entstanden sind. Ausgefüllt bekommt man die Graphen durch die Befehle surf bzw. surfc. Ein ähnliches Resultat liefert auch waterfall



Quelle: Christian Karpfinger und Broris von Loesh, Matlab – Eine Einführung

# Modellierung von Koordinatensysteme

```
close all
clear all
clc

%% KS1
plot3([0,1],[0,0],[0,0],'r')
hold on
plot3([0,0],[0,1],[0,0],'g')
hold on
plot3([0,0],[0,0],[0,1],'b')
hold on
text(0,0,0,'KS1')

%% Verbindungslinie 1-->2
plot3([0,2],[0,2],[0,2],'m')
text(2,2,2,'KS2')
hold on

%% KS2
plot3([2,2],[2,2],[2,3],'r')
hold on
plot3([2,2],[2,3],[2,2],'g')
hold on
plot3([2,3],[2,2],[2,2],'b')
```

