Ostfalia
Hochschule für angewandte Wissenschaften

Einführung in die Modellierung

Dipl.-Ing. Irina Ikkert, M.Eng.

Salzgitter
Suderburg
Wolfenbüttel
Wolfsburg

Organisatorisches

- 2 SWS LV: 1 SWS Vorlesung + 1 SWS Übung
- Vorlesung: Montags **11:45 – 13:15, HS E**
- Übung: Montags (jede zweite Woche) 09:45 – 11:15 (P3/P4)
  11:45 – 13:15 (P3/P4)
  Gruppenarbeit : 2 Studenten pro Gruppe.
- Insgesamt 13 Termine: 7 VL + 6 Übung am Rechner
- Ausfälle: keine

  Übung 16.10 Betreuung durch Hiwi
- Klausur 9.01, 9:00 – 10:00, im Rechenzentrum
  Hilfsmittel: - Formelsammlung (1 Blatt, handgeschrieben)
  - Vorlesungsfolien (am PC)

Vorlesungsinhalt

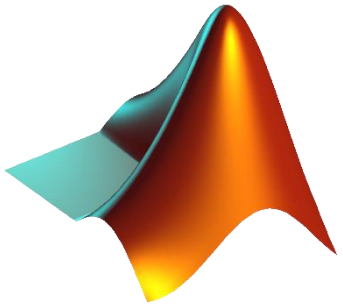
- Einführung
  - Erste Schritte
  - Desktop
  - Matlab – Hilfe
- Definitionen
  - Zahlen
  - Variablen, Vektoren und Matrizen
  - Strukturen und Cell Arrays
  - Character-Arrays
  - Ein- und Ausgabe
  - Import und Export von Daten
- Mathematische Berechnungen
  - Grundrechenarten. Skalar
  - Matrizen und Vektoren
  - Trigonometrische Funktionen
  - Komplexe Zahlen
  - Vergleichsoperatoren
  - Logische Operatoren
- Grafische Darstellungen, 2D
  - Grafik zeichnen
  - Grafikeigenschaften
  - Farbwerte, Punkt- und Linientypen
  - Mehrere Kurven in einem Diagramm
  - Diagrammtypen
- Grafische Darstellungen, 3D
  - 3D Funktion zeichnen
  - Diagrammtypen
  - Interaktives Plotten
  - 3D Objekte durch Rotation
  - Hilfreiche Befehle
- Kontrollstrukturen
  - Definition
  - for – Schleife
  - while – Schleife
  - If-elseif-else – Verzweigung
  - switch-case-otherwise
  - Hilfreiche Befehle

Dipl.-Ing. Irina Ikkert, M.Eng.

Vorlesungsinhalt

- Funktionen
  - Definition
  - Funktionsaufruf
  - Variable Anzahl der Ausgabeparameter
  - Aufgaben (Fakultät, lineare Funktion)
  - Variable Anzahl der Eingabeparameter (Mittelwert)
  - Funktionen als Eingabeparameter
  - Aufgaben (Grafik, Ableitung)
  - Nullstellen einer Funktion
- Polynome und Interpolation
  - Definition
  - Polynomoperationen
  - Partialbruchzerlegung
  - Interpolation (Lagrange, Chebyshev)
- Differential- und Integralrechnung
  - Numerische Differentiation
  - Symbolische Differentiation
  - Numerische Integration
  - Symbolische Integration
  - Auswertung symbolischer Berechnungen
- Symbolische Berechnung von Gleichungen, GS und DGL‘s
  - Gleichungen
  - Gleichungssysteme
  - Differentialgleichungen

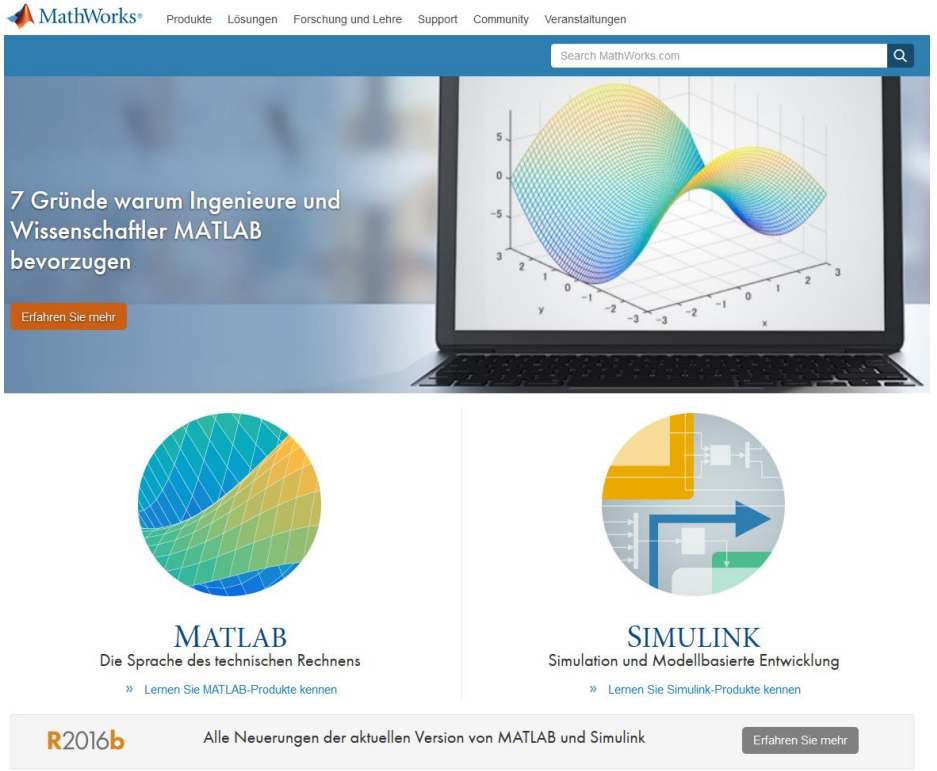
Einführung



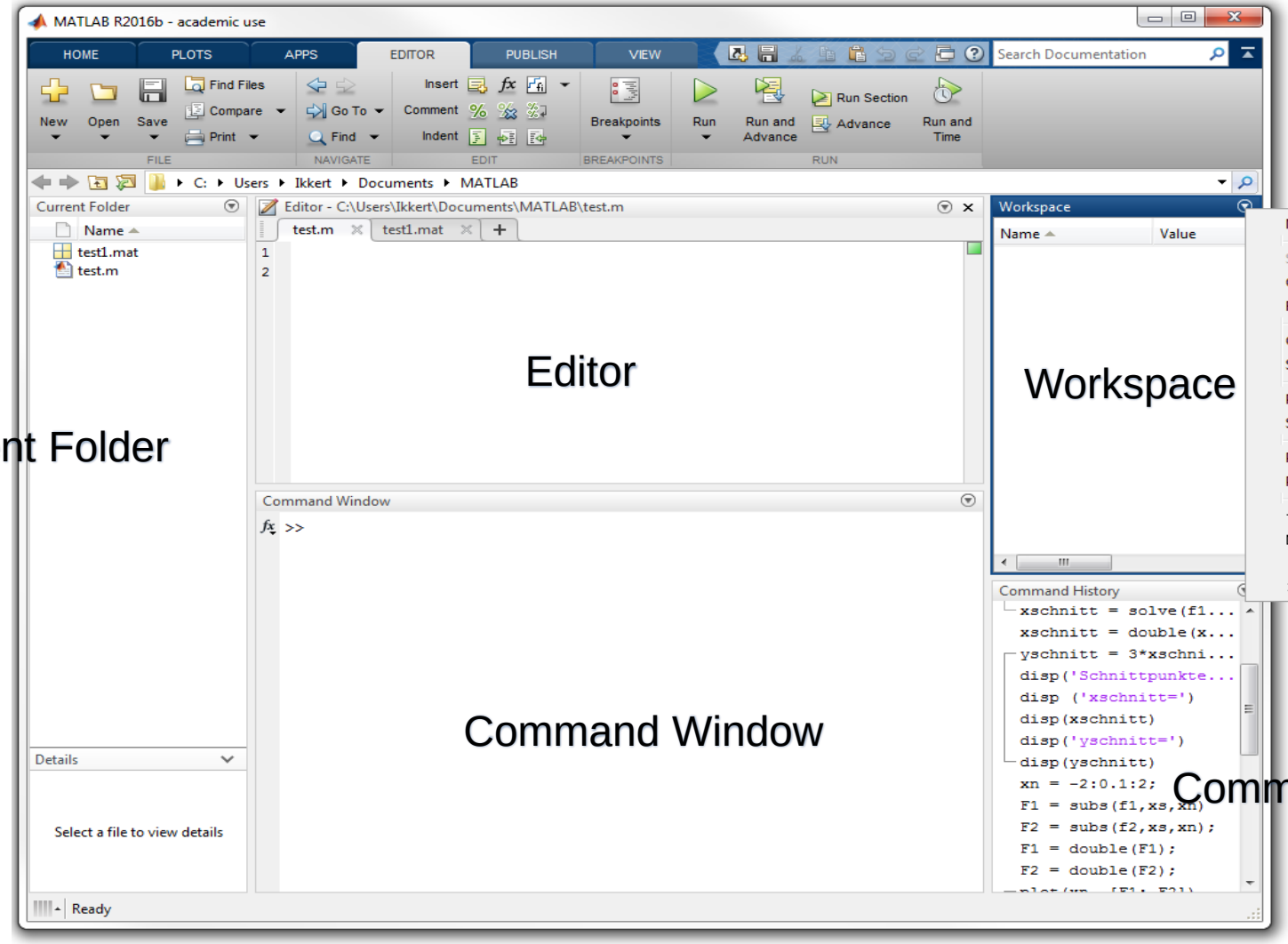
MATLAB - **Mat**rix **Lab**oratory

Installationspfad:

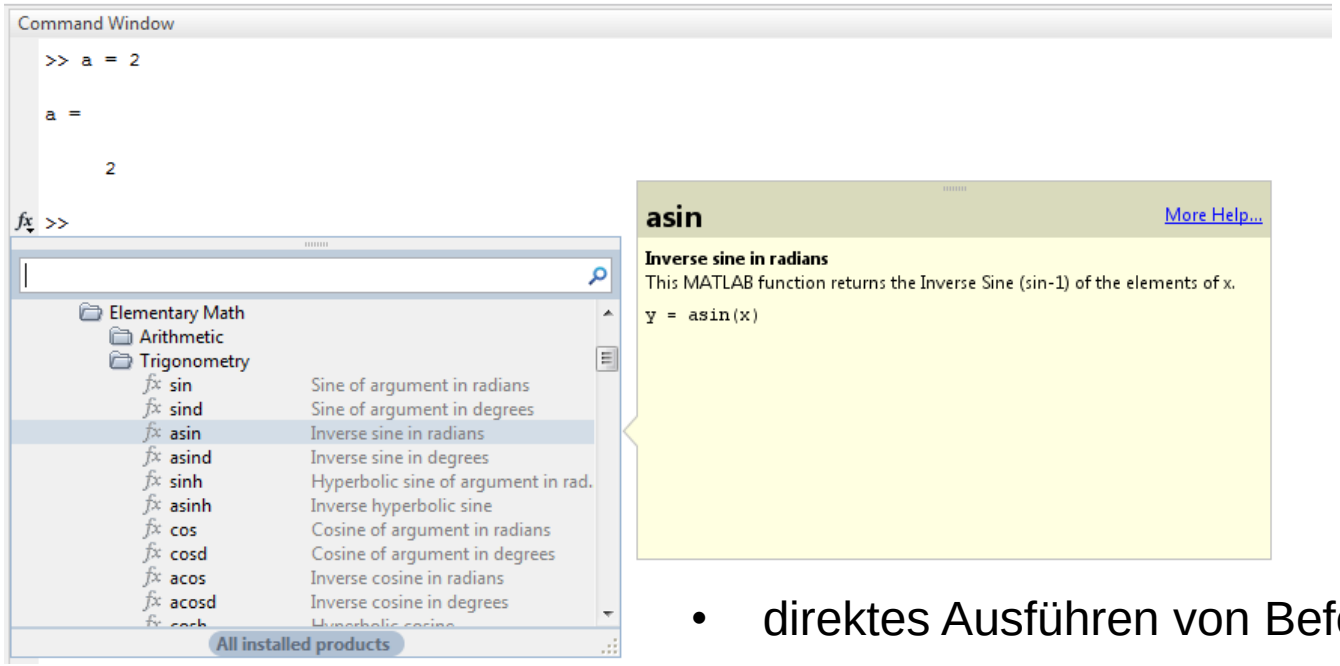
www.ostfalia.de → Portal → Allgemeine Dienste → Software → MathWorks



Einführung. Desktop



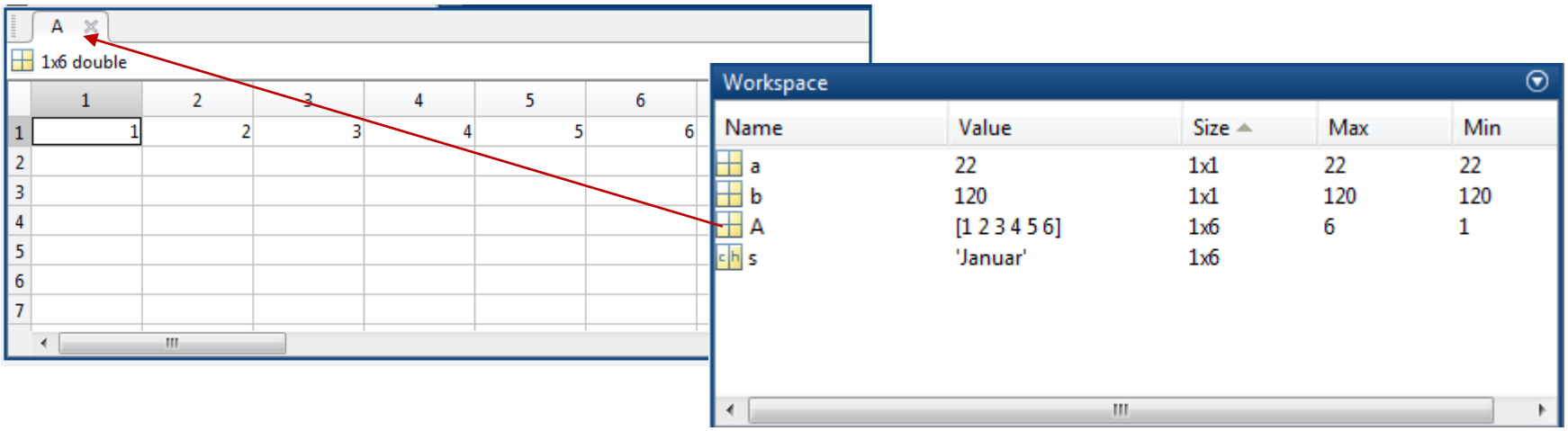
Command Window



- direktes Ausführen von Befehlen
- interaktive Ausgabe der Ergebnisse

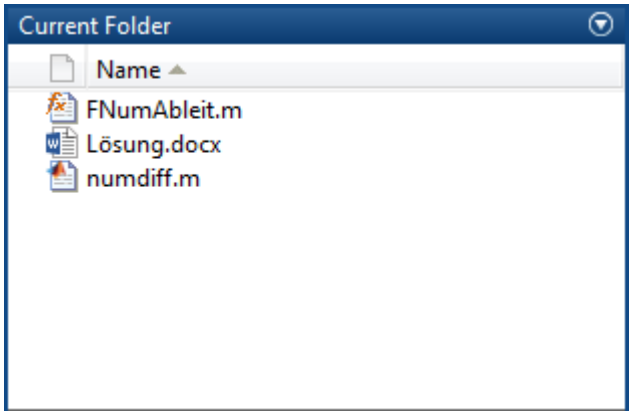
clc	löscht den Inhalt des Command Window
clear all	löscht alle Größen aus dem Workspace
clear x y	löscht Variablen x und y aus dem Workspace

Workspace - aktueller Arbeitsspeicher



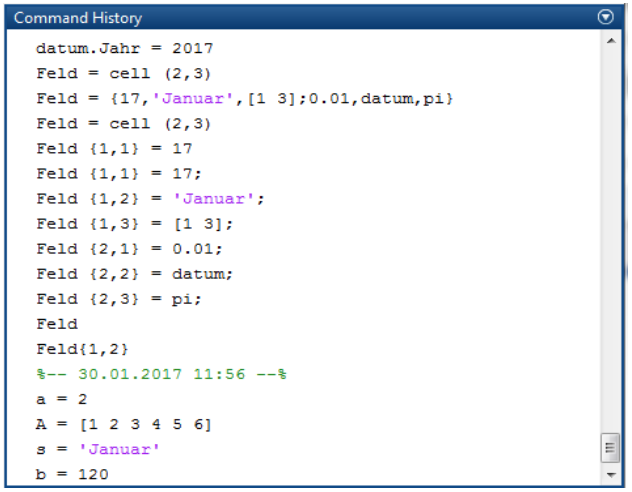
- Liste momentan im Arbeitsspeicher abgelegter Variablen
- Darstellung: Name, Größe, Dimension und Datentyp
- Variablen durch Doppelklick veränderbar
- weitere Eigenschaften über das Kontextmenü

Current Folder - aktueller Arbeitsverzeichnis



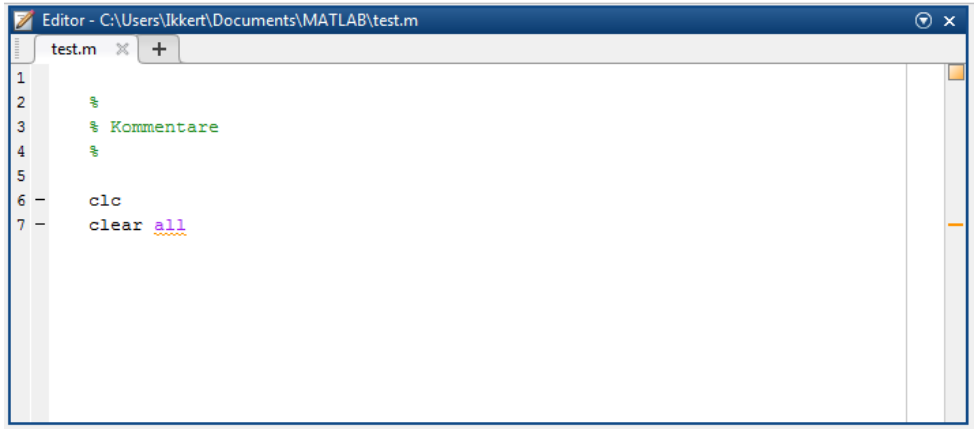
- kann eingesehen oder verändert werden
- ein neues Verzeichnis kann erstellt werden
- erzeugte Matlab Dateien sind sichtbar
- zum Öffnen der Dateien
- zum Ausführen der Funktionen

Command History – Chronik der Befehle



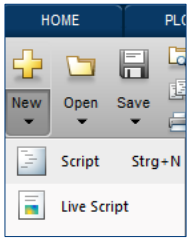
- speichert eingegebene Befehle geordnet ab
- Befehle können durch Doppelklick wiederholt werden

Editor. M-File.

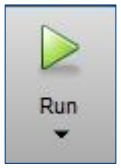


M-File erstellen:

Home → New Script



M-File ausführen:



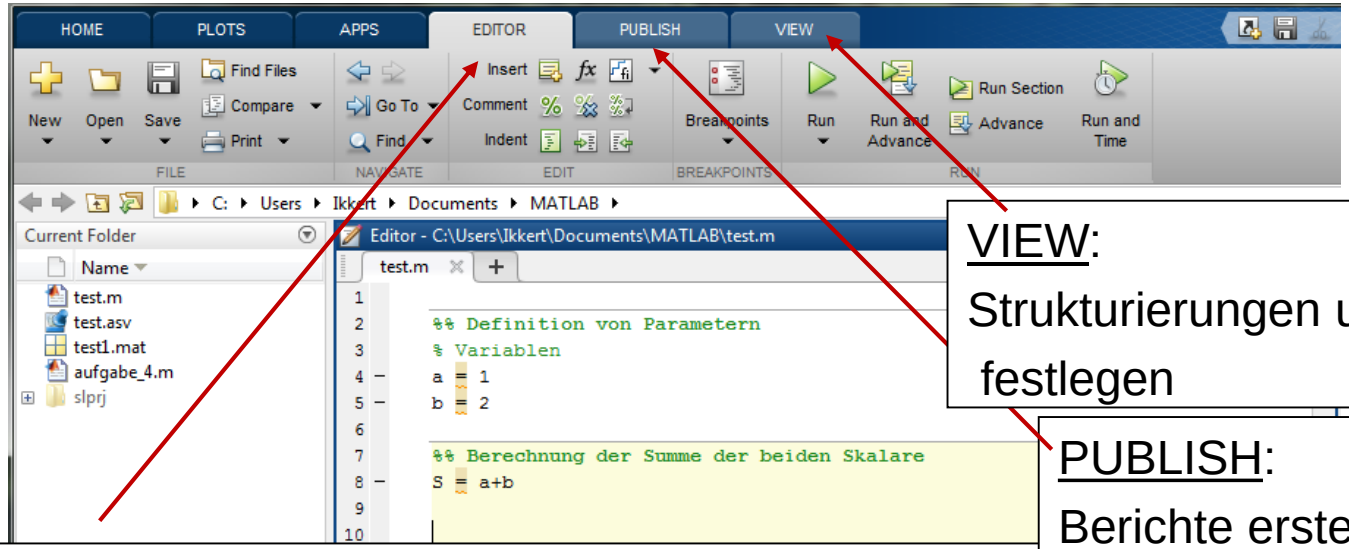
oder F5

Hier können Skripte und Funktionen erstellt und bearbeitet werden.

Vorteile:

- Speichern vom Programmcode
- Schrittweise Abarbeitung
- Syntax – Prüfung
- Debuggern von Fehlern
- Arbeiten mit Kommentaren

Editor. M-File.



VIEW:
Strukturierungen und Anordnungen festlegen

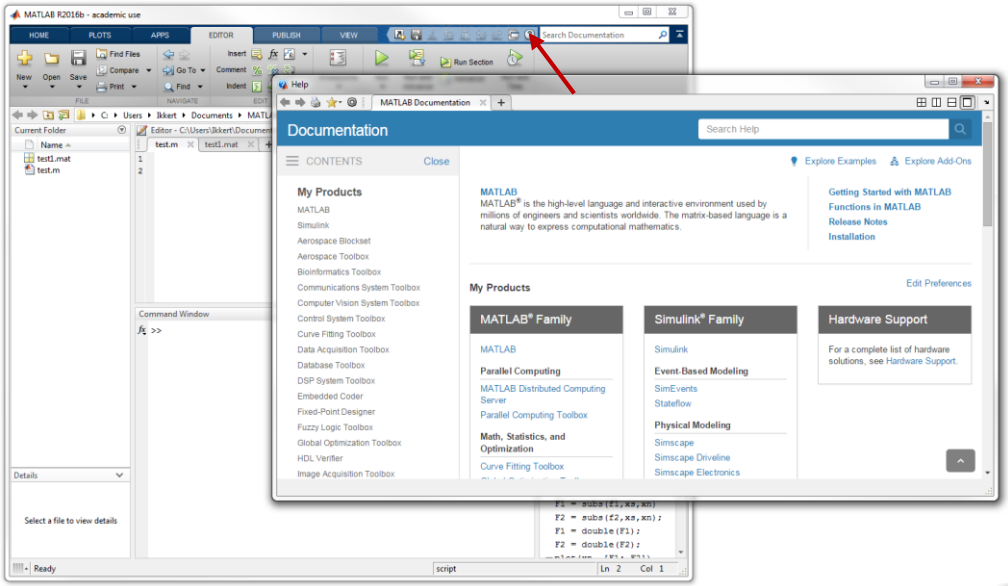
PUBLISH:
Berichte erstellen

EDITOR:

- FILE – erstellen, öffnen oder speichern von Dateien
- NAVIGATE – Textsuche und -navigation
- EDIT – Kommentare und Sektionen
- BREAKPOINTS – aufspüren von Laufzeitfehlern
- RUN – ausführen der Datei

Einführung. Matlab - Hilfe

- Hilfe - Browser

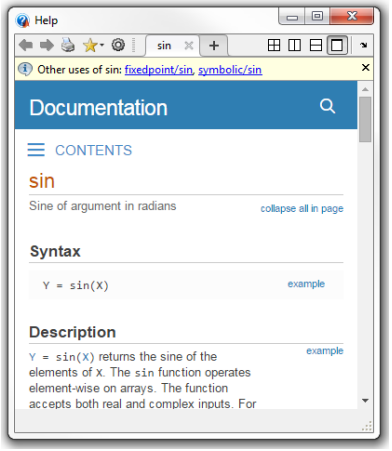


- help (Befehl)

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

    See also asin, sind.
```

- doc (Befehl)

```
>> doc sin
```



Definitionen

Darstellung von Zahlen

Zahlenbereich	10^{-308} bis 10^{308}
Unendlich ∞ (<10^{-308} oder > 10^{308})	Inf
Dezimalzahlen	9 -37 0.015 1.2E-20 9e73
Imaginäre Einheit $\sqrt{-1}$	i oder j
Komplexe Zahlen	3+6i oder 3+6j
Kreiszahl π	pi
Eulersche Zahl e	exp(1)
Ungültiges Ergebnis (Not a Number)	NaN

Datentypen

double(x)	Fließkommazahl, doppelte Genauigkeit (64 bit)
single(x)	Fließkommazahl, einfache Genauigkeit (32 bit)
int8(x); auch 16, 32, 64	8bit Festkommazahl, vorzeichenbehaftet (auch 16,32,64)
uint8(x); auch 16, 32, 64	8bit Festkommazahl, vorzeichenlos (auch 16,32,64)
logical(x)	logisches Ergebnis (1 oder 0)
char(x)	Zeichenkette (16 bit)

Typumwandlung

- single(a)
- int32(12.65) = 13

Runden von Fließkommazahlen

fix, round, ceil oder floor

Zahlenformate

Format	Darstellung der Zahl 1/7
short (short e)	0.1429 (1.4286e-01)
long (long e)	0.142857142857143 (1.428571428571429e-01)
hex	3fc2492492492492
rat	1/7

weitere Formate: bank, loose, compact

Umwandlung

```
>> format long
>> 1/7
ans = 0.142857142857143
>> format
```

Umrechnung von Zahlen in andere Formate

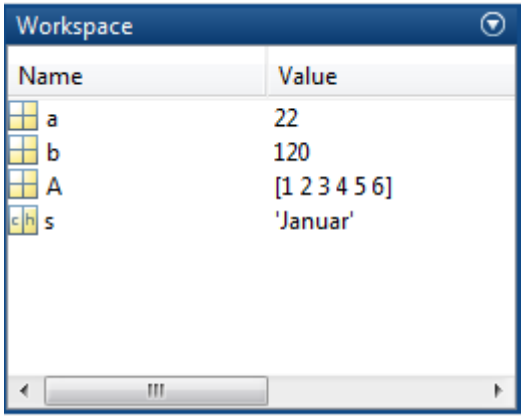
- dec2hex(a) bzw. hex2dec('str')
- num2hex(a) bzw. hex2num('str')
- dec2bin(a) bzw. bin2dec('str')
- dec2base(a,basis) bzw. base2dec('str',basis)

mit dem Befehl *format* werden die Standardeinstellungen wiederhergestellt

Variablen

Variablenname:

- keine Sonderzeichen (Unterstrich erlaubt)
- Anfangszeichen - immer Buchstabe
- Nicht länger als 63 Zeichen
- Klein- und Großschreibung beachten!



Vorteile:

- speichern
- überschreiben
- verändern
- weiterverarbeiten

Wertzuweisung:

- einfach

```
>> a = 9
a = 9

>> a = 9;
```

- mehrfach

```
>> a = 9, b = 12
```

Vektoren

Spaltenvektor	>> **v = [1;2;3]**	v = 1 2 3
Zeilenvektor	>> **v = [1 2 3]** oder >> **v = [1, 2, 3]**	v = 1 2 3
Zahlenbereich Anfangswert : Schrittweite : Endwert	>> **v = 1:5** (Schrittweite=1) >> **v = 1:2:6** >> **v = 0:0.1:1**; (Schrittweite<1)	v = 1 2 3 4 5 v = 1 3 5 v = 0 0.1 0.2 0.3 …
linear verteilter Wertebereich linspace(Anfangswert, Endwert, Anzahl der Werte)	>> **linspace(0,10,6)** >> **v = linspace(0,10)** (autom. 100 Werte)	v = 0 2 4 6 8 10 v = 0 0.101 0.202 0.303……
logarithmisch verteilter Wertebereich, (Basis 10) logspace(Exponent d. Anfangswerts, Exponent d. Endwerts, Anzahl der Werte)	>> **v = logspace(-2,2,5)**	v = 0.01 0.1 1.0 10.0 100.0

Matrizen

Wertzuweisung	>> **A = [1 2 3; 4 5 6; 7 8 9]** oder >> **A = [1, 2, 3; 4, 5, 6; 7, 8, 9]**	A = 1 2 3 4 5 6 7 8 9
Aus Vektoren bilden	>> v1 = [1;2;3], v2 = [4;5;6]; >> **A = [v1 v2]**	A = 1 4 2 5 3 6
	>> v1 = [1 2 3], v2 = [4 5 6]; >> **A = [v1; v2]**	A = 1 2 3 4 5 6

Spezielle Matrizen

(m,n): m – Anzahl der Zeilen, n – Anzahl der Spalten
(m): quadratische Matrix mxm

- **Nullmatrix**

A = zeros(m,n) oder (m)

>> **A = zeros(2,3)**

A =

0 0 0
0 0 0

- **Einheitsmatrix**

A = eye(m,n) oder (m)

>> **A = eye(3)**

A =

1 0 0
0 1 0
0 0 1

- **Matrix aus Einsen**

A = ones(m,n) oder (m)

>> **A = ones(3,2)**

A =

1 1
1 1
1 1

- **Zufallsmatrix (Wertebereich 0:1)**

A = rand(m,n) oder (m)

>> **A = rand(2)**

A =

0.9649 0.9706
0.1576 0.9572

Zugriff auf einzelne Elemente, Zeilen oder Spalten

Befehl	Bedeutung	Beispiel
A(x,y)	Element aus Zeile x und Spalte y	>> **a = A(3,1)** a = 7 A = 1 2 3 4 5 6 7 8 9
A(x1:x2,y)	Elemente x1 bis x2 aus Spalte y	>> **v = A(2:3,3)** v = 6 9 A = 1 2 3 4 5 6 7 8 9
A(x,y1:y2)	Elemente y1 bis y2 aus Zeile x	>> **v = A(2,1:2)** v = 4 5 A = 1 2 3 4 5 6 7 8 9
A(:,y)	Alle Elemente aus Spalte y	>> **v = A(:,1)** v = 1 4 7 A = 1 2 3 4 5 6 7 8 9
A(x,:)	Alle Elemente aus Zeile x	>> **v = A(3,:)** v = 7 8 9 A = 1 2 3 4 5 6 7 8 9
A(:)	Alle Elemente aus Matrix A	Das Ergebnis wird als Spaltenvektor ausgegeben

- Matrix aus Untermatrizen zusammensetzen

$$A = \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

```
>> B = [A  (A+10); 2*A  -A]
>> B =
```

A	1	2	11	12	(A+10)
	3	4	13	14	
2A	2	4	-1	-2	-A
	6	8	-3	-4	

- Untermatrix aus einer Matrix expandieren

```
>> C = B(2:3,3:4)
>> C =
```

13	14
-1	-2

1	2	11	12
3	4	13	14
2	4	-1	-2
6	8	-3	-4

B(2:3 , 3:4)

2 und 3 Zeile

3 und 4 Spalte

- Teil einer Matrix überschreiben

```
>> A(2,1) = C(1,1)
>> A =  1   2
       13   4
```

Strukturen

- **struct('name1',wert1,'name2',wert2,……)**

```
>> daten_beispiel = struct('Vektor', v, 'Matrix', A);
>> daten_beispiel.Matrix
ans =
     4     7
     3     6
```

- **mit Punkt (.) als Separator**

```
>> datum.Tag = 27;
>> datum.Monat = 1;
>> datum.Jahr = 2017;

datum =
  struct with fields:

      Tag: 27
    Monat: 1
     Jahr: 2017
```

Cell Arrays

- **cell(m,n)**

```
>> Feld = cell (2,3)

Feld =
  2×3 cell array
    []    []    []
    []    []    []

>> Feld {1,1} = 17; >> Feld {1,2} = 'Januar';
>> Feld {1,3} = [1 3]; >> Feld {2,1} = 0.01;
>> Feld {2,2} = datum; >> Feld {2,3} = pi;
>> Feld

Feld =
  2×3 cell array

    [    17]    'Januar'        [1×2 double]
    [0.0100]    [1×1 struct]    [    3.1416]
```

- **Geschweifte Klammern {}**

```
>> Feld = {17,'Januar',[1 3]; 0.01,datum,pi}

Feld =
  2×3 cell array

    [    17]    'Januar'        [1×2 double]
    [0.0100]    [1×1 struct]    [    3.1416]

>> Feld{1,2}

ans =
Januar
```

Cell Arrays können Daten unterschiedlicher Datentypen enthalten!

Text in Matlab

- ‘text‘ – wird mit Hilfe von Anführungszeichen erzeugt

```
>> 'text'
>> ans = text
```

- string = ‘text‘ – kann Variablen (Zeilenvektor) zugewiesen werden

```
>> string = 'Das ist ein String';
>> whos string
Name        Size            Bytes  Class    Attributes
string      1x18               36  char
```

- string = [‘text1‘,‘text2‘] – kann aus mehreren Texten zusammengesetzt werden

```
>> SommerSemester = ['März  ','April  ','Mai  ','Juni  ']

SommerSemester =  März  April  Mai  Juni
```

Eingabe in Matlab

- Abfrage von Daten : **variable = input(Begleittext)**

```
>> Zahl = input(['Geben Sie eine ganze Zahl ein \n', 'Zahl:']);   % Datenabfrage
Geben Sie eine Zahl ein                                          % System - Antwort
Zahl:17                                                          % Zahleneingabe
```

- Abfrage vom Text : **string = input(Begleittext,‘s‘)**

```
>> paritaet = input('Ist diese Zahl gerade oder ungerade? ','s')  % Textabfrage
Ist diese Zahl gerade oder ungerade?  Ungerade                    % System – Antwort + Eingabe
```

Ausgabe in Matlab

– Ausgabe mit **disp**:

```
>> disp('Die Zahl lautet:  ')      % Ausgabe eines Textes
>> disp(Zahl)                      % Ausgabe einer Variable
>> disp('Parität ist: ')           % Ausgabe eines Textes
>> disp(paritaet)                  % Ausgabe einer char-Variable
Die Zahl lautet: 17                %System – Antwort
Parität ist:  gerade               %System – Antwort
```

– Formatierte Ausgabe vektorieller Daten:

a) **string = sprintf(string,variable)**

```
>> ausgabe = sprintf('Hat die nächste Zahl %d auch eine %s Parität? ',(Zahl+1),paritaet);%Formatierung
>> disp(ausgabe) %Ausgabe
Hat die nächste Zahl 18 auch eine ungerade Parität? %System – Antwort
```

b) **num2str(variable,'format')**

```
>> disp(['Hat die nächste Zahl ',num2str(Zahl+1,'%d'),' auch eine ',paritaet,' Parität?'])%Formatierte Ausgabe
Hat die nächste Zahl 18 auch eine ungerade Parität? %System – Antwort
```

Sonderzeichen	Bedeutung	Sonderzeichen	Bedeutung
\n	Zeilenumbruch	%d	Ganze Zahl
\t	Tabulator	%x	Ganze Zahl hexadezimal
\\	Backslash	%f	Fließkomma – Zahl
%%	Prozentzeichen	%e	Exponenten Schreibweise
“	Anführungszeichen	%s	String

Definitionen. Import und Export von Daten

	Befehl	Beschreibung
Speichern	**save** dateiname	Speichern von Workspace Variablen in die Datei dateiname.mat
	save dateiname v1 v2 …	Speichern v1,v2… Variablen in die Datei dateiname.mat
	save datei.endung –ascii v1 v2	Speichern in ASCII – File datei
	xlswrite(‘datei.xlsx‘,[v1 v2])	Speichern in Excel-File
	save Workspace (über Home→Variable)	Interaktives speichern
Laden	**load** dateiname	Laden von Variablen aus der Datei dateiname.mat
	load dateiname v1 v2 …	Laden v1,v2… Variablen aus der Datei dateiname.mat
	load datei.endung	Laden aus ASCII – File
	xlsread(‘datei.xlsx‘)	Laden aus Excel - File
	Import Data (über Home → Variable)	Interaktives laden

Dipl.-Ing. Irina Ikkert, M.Eng.

Mathematische Berechnungen

Grundrechenarten. Skalar

Operation	Syntax	Beispiel
Addition	a+b	>> **7+15** ans = 22
Subtraktion	a-b	>> **7-15** ans = -8
Multiplikation	a*b	>> **7*15** ans = 105
Division	a/b	>> **7/15** ans = 0.4667
Potenzieren	a^b	>> **7^15** ans = 4.7476e+12
Quadratwurzel	sqrt(a)	>> **sqrt(7)** ans = 2.6458
Betrag	abs(a)	>> **abs(-7)** ans = 7
Exponentialfunktion, e^a	exp(a)	>> **exp(7)** ans = 1.0966e+03
Logarithmus, Basis e	log(a)	>> **log(7)** ans = 1.9459
Logarithmus, Basis 10	log10(a)	>> **log10(7)** ans = 0.8451

Ostfalia
Hochschule für angewandte Wissenschaften
Wolfenbüttel

Matrizen und Vektoren

$$a = 2 \qquad A = \begin{matrix} 4 & 9 \\ 4 & 5 \end{matrix} \qquad B = \begin{matrix} 2 & 3 \\ 1 & 5 \end{matrix}$$

* Für Vektoren gelten gleiche Regeln wie für die Matrizen

Operation		Syntax	Beispiel
Addition	Matrix+Skalar	B+a	>> **B+a** = 4 5 3 7 >> **A+B** = 6 12 5 10
	Matrix+Matrix	A+B	
Subtraktion	Matrix-Skalar	B-a	>> **B-a** = 0 1 -1 3 >> **A-B** = 2 6 3 0
	Matrix-Matrix	A-B	
Multiplikation	Matrix*Skalar	B*a	>> **B*a** = 4 6 2 10 >> **A*B** = 17 57 13 37 >> **A.*B** = 8 27 4 25
	Matrix*Matrix	A*B	
	Matrix.*Matrix (elementweise)	A.*B	
Division	Matrix/Skalar	B/a aber a./B	>> **B/a** = 1 1.5 0.5 2.5 >> **a./B** = 1 0.7 2 0.4 >> **A./B** = 2 3 4 1
	Matrix./Matrix (elementweise)	A./B	
Potenzieren	$\text{Matrix}^{\text{Skalar}}$	A^a	>> **A^a** = [52 81; 36 61]
	$\text{Matrix}^{\text{Skalar}}$ (el.)	A.^a	>> **A.^a** = [16 81; 16 25]

Matrizen und Vektoren

- Berechnen Sie die Multiplikation zweier Matrizen:

$$A = \begin{matrix} 8 & 6 \\ 9 & 0 \\ 12 & -2 \\ 3 & 5 \end{matrix} \qquad B = \begin{matrix} 9 & -6 & 4 & 5 & -2 & 0 \\ 7 & 4 & -1 & -8 & 8 & 9 \end{matrix}$$

% A=(4*2) und B=(2*6)

% (4*2)*(2*6) = (4*6)

```
>> A = [8 6;9 0;12 -2;3 5];  % Matrix A

>> B = [9 -6 4 5 -2 0;7 4 -1 -8 8 9]; % Matrix B

>> A*B  % Matrizenmultiplikation
```

A*B =

114	-24	26	-8	32	54
81	-54	36	45	-18	0
94	-80	50	76	-40	-18
62	2	7	-25	34	45

Matrizen und Vektoren

- Berechnen Sie die Lösung folgendes linearen Gleichungssystems:

$10x_1 - 3x_2 = 4$
$4x_1 + 5x_2 = 3$

Matrizenform:

A*X = B

$$\begin{pmatrix} 10 & -3 \\ 4 & 5 \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \end{pmatrix}$$

A^(-1)*A*X = A^(-1)*B
X = A^(-1)*B

>> A = [10 -3;4 5] % Koeffizientenmatrix

>> B = [4; 3] % rechte Seite

>> **X = A^(-1)*B** % Matrizenmultiplikation

X = 0.4677 → x_1 = 0.4677
0.2258 → x_2 = 0.2258

Matrizen und Vektoren

- Berechnen Sie die elementweise Multiplikation zweier Matrizen:

$$C = \begin{matrix} 8 & 6 \\ 9 & 0 \\ 12 & -2 \end{matrix} \qquad D = \begin{matrix} 9 & -6 \\ 7 & 4 \\ 3 & 5 \end{matrix}$$

```
%  C=(3*2) und D=(3*2)

%  (3*2).*(3*2) = (3*2)
```

```
>> C = [8 6;9 0;12 -2];  % Matrix C

>> D = [9 -6;7 4;3 5];   % Matrix D

>> C.*D % elementweise Matrizenmultiplikation
```

C.*D =
```
72  -36
63    0
36  -10
```

Matrizen und Vektoren

- Berechnen Sie die Funktionswerte für die gegebenen Punkte:

x	x1	x2	x3	……..	xn
y	y1	y2	y3	…….	yn

$$y = (x + 2) \cdot ln(x)$$

```
% Wahl des Definitionsbereichs
>> x = 1:0.5:4
x =
   1.0000   1.5000   2.0000   2.5000   3.0000   3.5000   4.0000

% Berechnung der Funktionswerte
>> y = (x+2).*log(x) % elementweise Matrizenmultiplikation

y =
         0   1.4191   2.7726   4.1233   5.4931   6.8902   8.3178
```

Matrizen und Vektoren

- Eingabe von Funktionen

$$y = \left|\sqrt{x^2 - 2x}\right| \cdot e^{\frac{1}{x}}$$

```
>> x = 2:10;
>> y = abs(sqrt(x.^2 - 2*x)).*exp(1./x)
```

Potenzieren eines Vektors

Skalar durch einen Vektor

- Berechnen von Logarithmen

```
>> v = [1 10 100 1000 10000]
>> log10(v) =    0    1    2    3    4
```

- Matrix – Operationen

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} \qquad B = \begin{matrix} 1 & 2 & 1 \\ 4 & 3 & 2 \end{matrix}$$

- Berechnen $A \cdot B$

```
>> A*B
Error using  *
Inner matrix dimensions must agree.
```

- Berechnen $A \cdot B$ elementweise

```
>> A.*B

ans =
     1    4    3
    16   15   12
```

Matrizen und Vektoren

$$v = [1\ 3\ 8\ ] \qquad A = \begin{matrix} 4 & 7 \\ 3 & 6 \end{matrix}$$

Operation	Syntax	Beispiel
Transponieren	T = A’	>> **v’** = 1 3 8 >> **A’** = 4 3 7 6
Inverse Matrix	I = inv(A)	>> **inv(A)** = 2.0000 -2.3333 -1.0000 1.3333
Quersumme	S = sum(A)	>> **sum(v)** = 12 >> **sum(A)** = 7 13
Größe	s = size(A)	>> **size(v)** = 1 3 >> **size(A)** = 2 2
Länge	L = length(A)	>> **length(v)** = 3 >> **length(A)** = 2
Max und Min	max(A), min(A)	>> **max(v)** = 8 >> **max(A)** = 4 7
Eigenwerte, -vektoren	eig(A), [V,D]=eig(A)	>> **eig(A)** = 0.3096 9.6904
Determinante	det(A)	>> **det(A)** = 3

Trigonometrische Funktionen

Argument	Sinus	Kosinus	Tangens
x, in radiant	sin(x)	cos(x)	tan(x)
x, in Grad	sind(x)	cosd(x)	tand(x)
Hyperbelfunktionen	sinh(x)	cosh(x)	tanh(x)

Arcus-Funktionen: Inverses der obigen Funktionen durch zusätzliches „a“ vor dem Befehl

- asin(x), acos(x), atan(x) Ergebnis in rad
- asind(x), acosd(x), atand(x) Ergebnis in Grad
- asinh(x), acosh(x), atanh(x) Arcushyperbolikus, rad

Umrechnung :

$$\boldsymbol{x, rad} = \frac{x(grad) \cdot \pi}{180°} \qquad \boldsymbol{x, grad} = \frac{x(rad) \cdot 180°}{\pi}$$

Komplexe Zahlen

Imaginäre Einheit in Matlab: i oder j , $i^2 = -1$

- Kartesische Form

$$z = x + i \cdot y \quad \text{oder} \quad z = complex(x, y)$$

- Polar- oder Trigonometrische Form

$$z = r \cdot \left(e^{i\varphi}\right) = r \cdot (\cos\varphi + i \cdot \sin\varphi), \qquad r = |z| = \sqrt{x^2 + y^2} \quad \text{und} \quad \varphi = \arg(z)$$

Operation	Syntax	Operation	Syntax
Betrag	abs(z)	Realteil	real(z)
Winkel, (rad)	angle(z)	Imaginärteil	imag(z)
Konjugiert komplex	conj(z)	Polardarstellung	compass(z)

Komplexe Zahlen

$$z = 4 + i \cdot 3$$

- Geben Sie die Zahl z in Polarform an

>> r = **abs(z)** = 5 % Betrag von z

>> phi = **angle (z)** = 0.6435 % Phasenwinkel von z in rad

>> **(phi*180)/pi** = 36.8699 % Umrechnung ins Gradmaß

Die Zahl z in Polarform lautet: $z = 5 \cdot e^{37i}$

- Bestimmen Sie Real- und Imaginärteil von z und die konjugiert komplexe Zahl

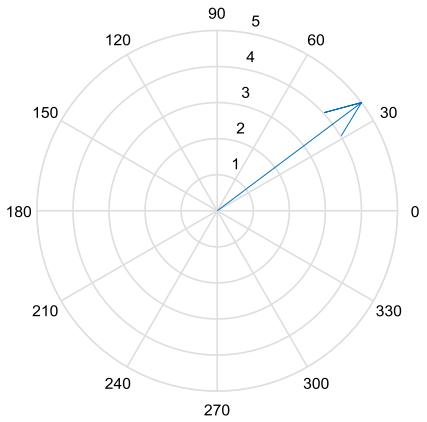
>> **real(z)** = 4 % Realteil von z

>> **imag(z)** = 3 % Imaginärteil von z

>> **conj(z)** = 4.0000 - 3.0000i % konjugiert komplex zu z

- Stellen Sie die Zahl in Polarform dar

>> **compass(z)**



Vergleichsoperatoren

Operation	Befehl	Operation	Befehl
gleich	==	größer als	>
ungleich	~=	kleiner oder gleich	<=
kleiner als	<	größer oder gleich	>=

Ergebnis: logische 0 – für eine falsche Aussage
1 – für eine richtige Aussage

- **Vergleich zweier Zahlen**

```
>> a=2, b=3
>> a <= b
ans =
     logical
     1
```

- **Vergleich Matrix mit einer Zahl**

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

>> A >= 4
ans =
     2x2 logical array

     0  0
     0  1
```

- **Vergleich zweier Matrizen**

```
>> A=[1 2; 3 4]
>> B=[1 3; 3 0]

>> A == B
ans =
     2x2 logical array

     1  0
     1  0
```

Logische Operatoren

Logische Operation	Befehl	Beispiel
und	&	>> **0&1 ans = 0**
oder	|	>> **0|1 ans = 1**
nicht	~	>> **~1 ans = 0**
entweder oder	xor	>> **xor(1,1) ans = 0**

Ergebnis:
0 – für eine falsche Aussage
1 – für eine richtige Aussage

$x = [-2\ 3\ 1\ 0\ 4],\quad y = [9\ 0\ 7\ 0\ 0]\quad und\quad z = [-4\ 6\ 2\ 0\ 8]$

- >> **x > y**

>> ans = 0 1 0 0 1

- >> **x == -2*z**

>> ans = 0 0 0 1 0

- >> **y(x<=1)**

>> ans = 9 7 0

- >> **x & (~y)**

>> ans = 0 1 0 0 1

- >> **x>2 & x<8 & y<=0**

>> ans = 0 1 0 0 1

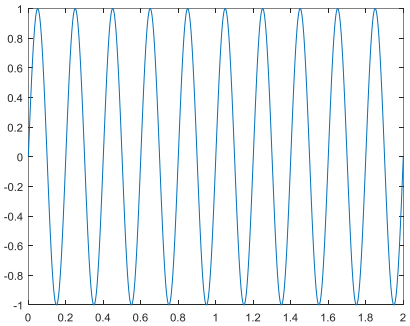
- >> **z((x<=2)|(y>=4))**

>> ans = -4 2 0

Grafische Darstellungen. 2D

Grafik zeichnen

- plot(x,y) - zeichnet die y-Werte über den x-Werten
  - x: $[x_1, x_2 \ldots x_n]$ Vektor der Abszisse-Werte
  - y: $[y_1, y_2 \ldots y_n]$ Vektor der Ordinate-Werte
  - x und y müssen gleich lang sein




```
>> x=0:0.001:2;          % Festlegung des x-Wertebereichs
>> y=sin(2*pi*5*x);      % Berechnung der Funktionswerte (y-Wertebereich)
>> plot(x,y)             % Grafische Darstellung der Funktion y(x)
```

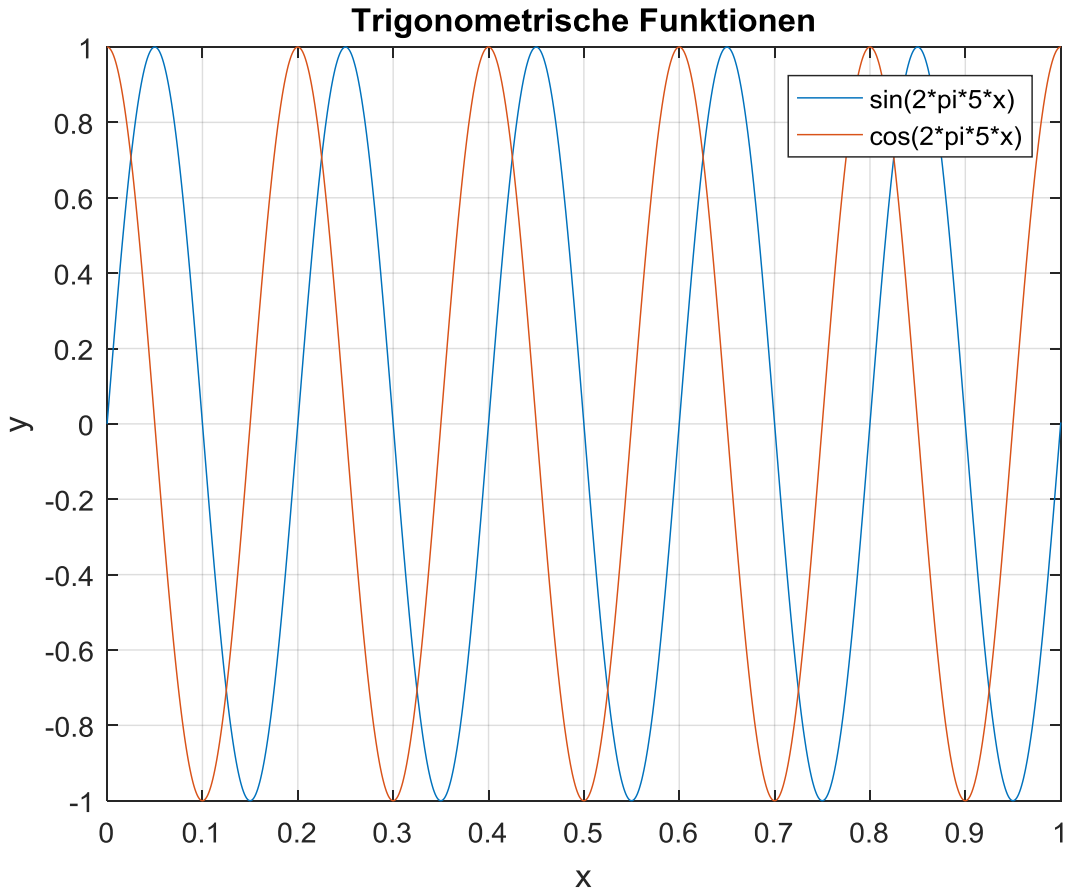
- plot(y) - zeichnet die y-Werte über der Folge (1,2,3….)

- plot(x1,y1,x2,y2…) - zeichnet mehrere Funktionen in einem Diagramm

Grafikeigenschaften

Befehl	Beschreibung
figure	Neues Grafikfenster
grid on (grid off)	Anzeigen (Ausblenden) von Gitternetzlinien
legend (‘ *text* ‘)	Einfügen einer Legende
title(‘ *text* ‘)	Einfügen eines Diagrammtitels
xlabel(‘ *text* ‘)	Beschriftung der x-Achse
ylabel(‘ *text* ‘)	Beschriftung der y-Achse
axis([xmin,xmax,ymin,ymax])	Skalierung der Achsen
plot(x,y,‘y--‘)	Optionen für Farbe und Stil der Linien
close (all)	Das aktuelle Grafikfenster wird geschlossen
axis equal	Gleiche Achsenmaßstäbe

Grafikeigenschaften




```
>> close
>> figure
>> x=0:0.001:2;
>> y1=sin(2*pi*5*x);
>> y2=cos(2*pi*5*x);
>> plot(x,y1,x,y2)
>> axis([0,1,-1,1]);
>> grid on
>> legend('sin(2*pi*5*x)','cos(2*pi*5*x)');
>> title('Trigonometrische Funktionen');
>> xlabel('x');
>> ylabel('y');
```

Farbenwerte, Punkt- und Linientypen

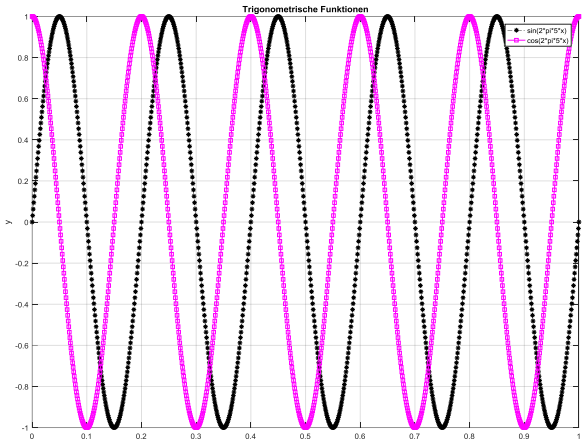
Farbe	Linientyp	Punkttyp	
b – blau	– durchgezogen	. Punkt	v,∧ Dreieck
g – grün	: gepunktet	o Kreis	<,> Dreieck
r – rot	-. Strich – Punkt	x X-Marker	p Pentagramm
c – cyan	-- gestrichelt	+ Plus	h Hexagramm
m – magenta		* Stern	
y – gelb		s Quadrat	
k – schwarz		d Raute	

Weitere Befehle:

- LineWidth Liniendicke
- MarkerSize Größe von Markern
- MarkerEdgeColor Farbe der Markerumrandungen
- MarkerFaceColor Markerfüllungen

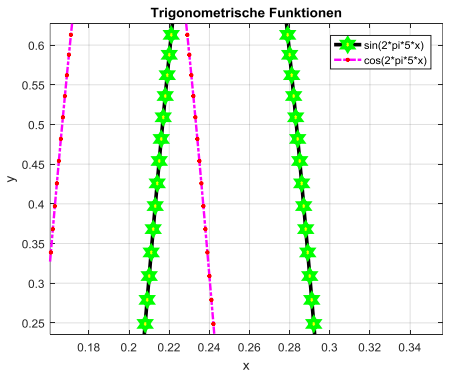
Können nur in separaten **plot** – Befehlen definiert werden

Farbenwerte, Punkt- und Linientypen



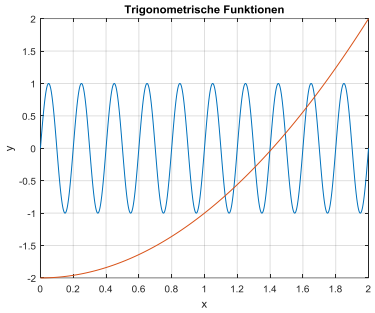

```
.
.
>> y1=sin(2*pi*5*x);
>> y2=cos(2*pi*5*x);
>> plot(x,y1,'k-.*',x,y2,'m-s');
>> axis([0,1,-1,1]);
.
.
.
```

```
.
>>
>> plot(x,y1,'k-h','LineWidth',3,'MarkerSize',10,'MarkerEdgeColor','g','MarkerFaceColor','y');

>> hold on      % damit die Sinuskurve nicht durch den plot-Befehl der Kosinuskurve
                  überschrieben wird

>> plot(x,y2,'m-.*','LineWidth',2,'MarkerSize',3,'MarkerEdgeColor','r');

>> hold off
>> axis([0,1,-1,1]);
>> grid on
```



Mehrere Kurven in einem Diagramm

- **hold** Die im Grafikfenster bereits bestehende Kurve wird nicht durch einen neuen plot-Befehl überschrieben

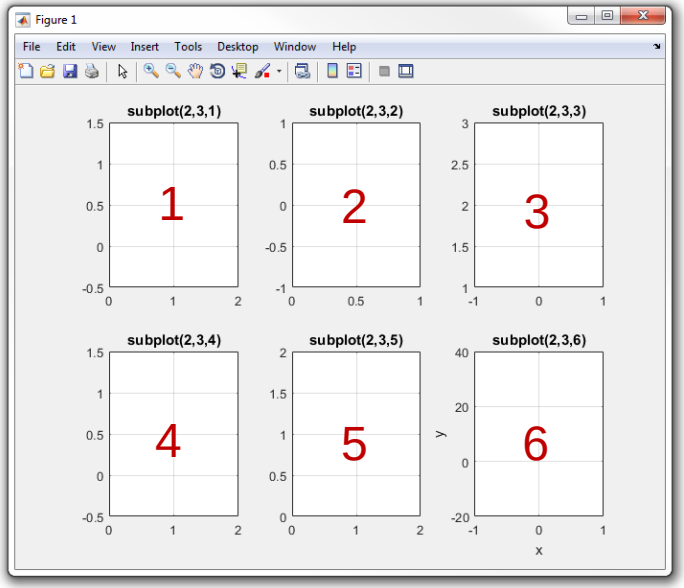



```
>>x=0:0.001:2;
>> y1=sin(2*pi*5*x);
>> y3 = x.^2-2;
>> plot(x,y1);
>> hold on
>> plot(x,y3);
>> hold off
```

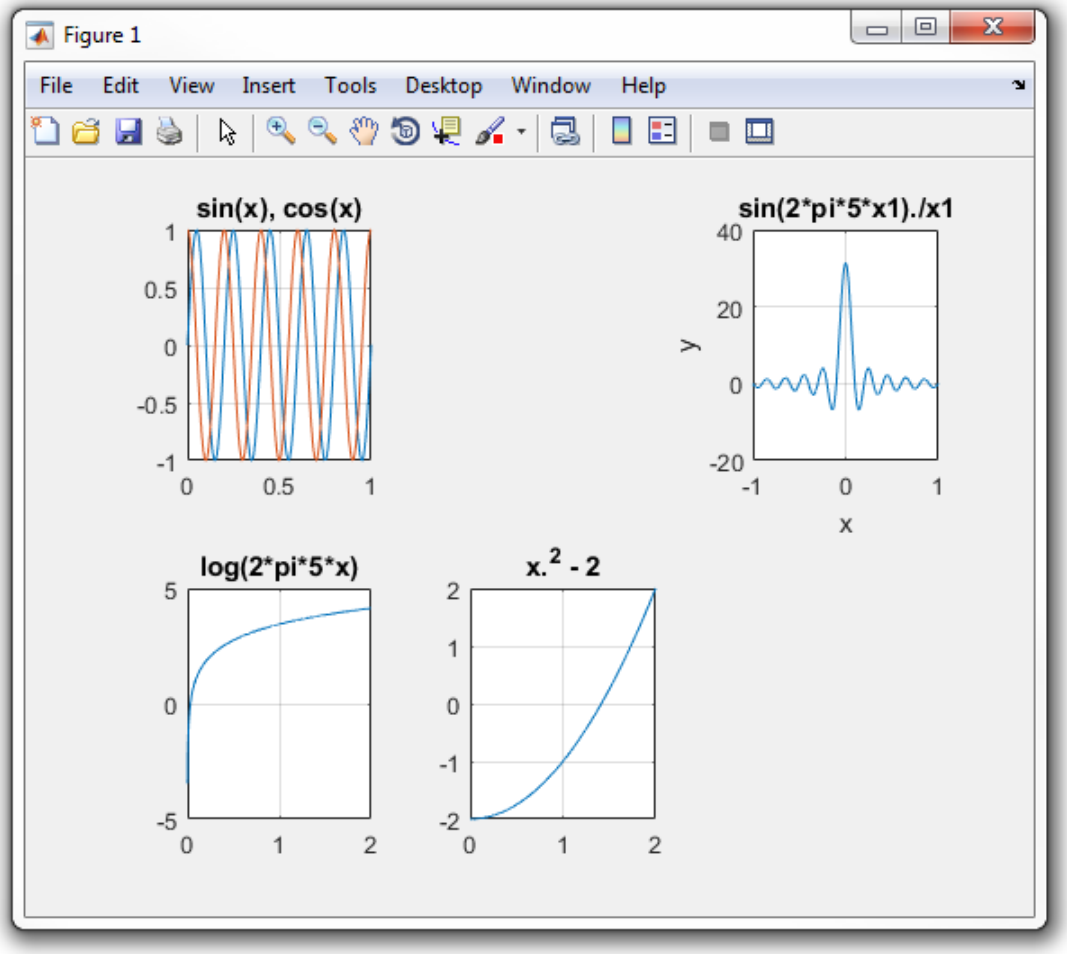
- **subplot(m,n,k)**

(m,n,k): m – Anzahl der Zeilen, n – Anzahl der Spalten,
k – die Position, an der das Diagramm platziert werden soll.

- Reihenfolge ist beliebig
- Felder können überschrieben werden
- Felder müssen nicht belegt sein
- Für jedes Unterdiagramm **subplot + plot** Befehle erforderlich
- Jedem Unterdiagramm kann Titel, Legende, Gitternetzlinien zugeordnet werden



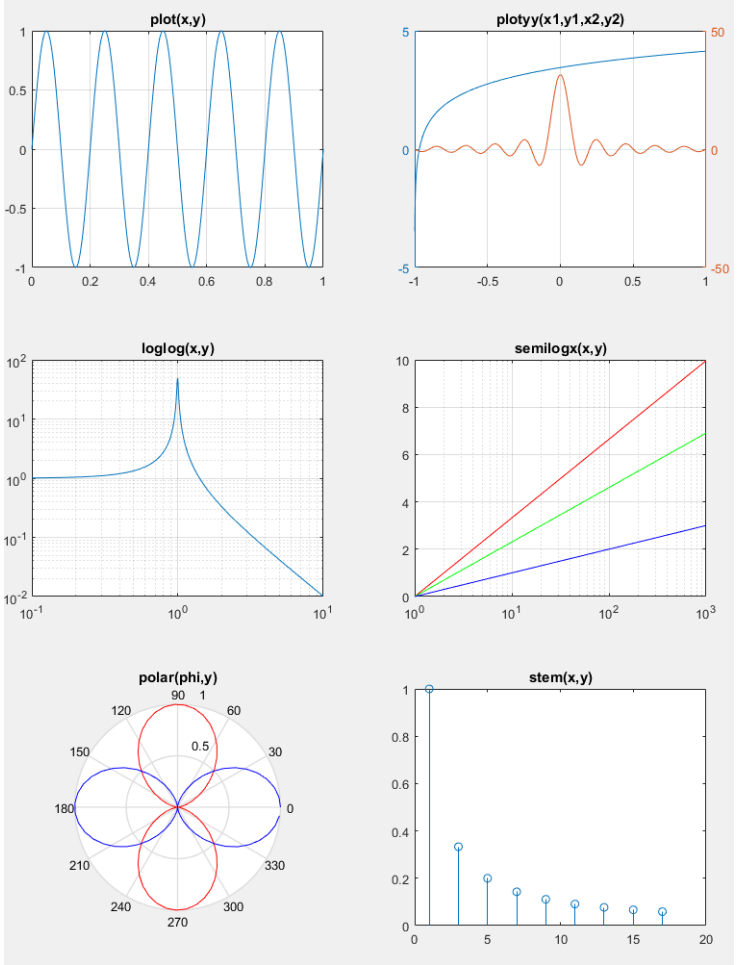
Mehrere Kurven in einem Diagramm




```
>>subplot(2,3,1)
>> plot(x,y1);
>> hold on
>> plot(x,y);
>> axis([0,1,-1,1]);
>> title('sin(x), cos(x)');
>> grid on
>> hold off
>> subplot(2,3,4)
>> plot(x,y2);
>> title('log(2*pi*5*x)');
>> grid on
>> subplot(2,3,5)
>> plot(x,y3);
>> title('x.^2 - 2');
>> grid on
>> subplot(2,3,3)
>> plot(x,y4);
>> title('sin(2*pi*5*x)./x');
>> axis([-1,1,-20,40]);
>> grid on
>> xlabel('x');
>> ylabel('y');
```

Diagrammtypen

Typ	Befehl	Beschreibung
linear	plot(y)	y über (1,2,3…)
	plot(x,y)	y über x
	plotyy(x1,y1,x2,y2)	zwei y-Achsen
logarithmisch	loglog(x,y)	beide Achsen logarith.
	semilogx(x,y)	x-Achse logarithmisch
	semilogy(x,y)	y-Achse logarithmisch
radial	polar(winkel,radius)	Polarkoordinaten
	compass(z)	Zeiger, komplex
Balken	stem(x,y)	y als senkrechte Linien über x



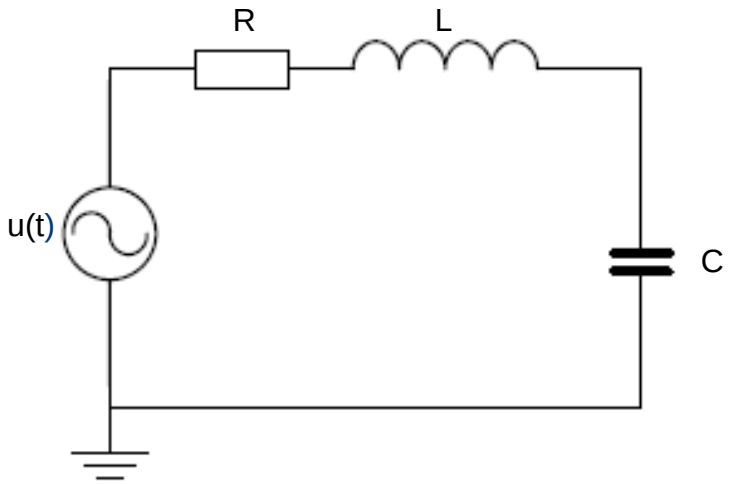
weitere Diagramme: stairs, contour, bar, pie etc.

RLC Reihenschwingkreis

Resonanzfall:

$$\boldsymbol{\omega = \omega_0}$$

ω - Anregungsfrequenz
ω_0 - Eigenfrequenz des Reihenschwingkreises



Gesamtimpedanz:

$$\underline{Z} = R + j\omega L + \frac{1}{j\omega C} = \left|\underline{Z}\right| \cdot e^{j\theta}$$

Strom- und Spanungsverläufe:

$$\hat{I} = \frac{\hat{U}}{\left|\underline{Z}\right|} \qquad \underline{U}_L = \hat{I} \cdot j\omega L \qquad \underline{U}_C = \frac{\hat{I}}{j\omega C}$$

Phasenverschiebung zwischen Strom und Spannung:

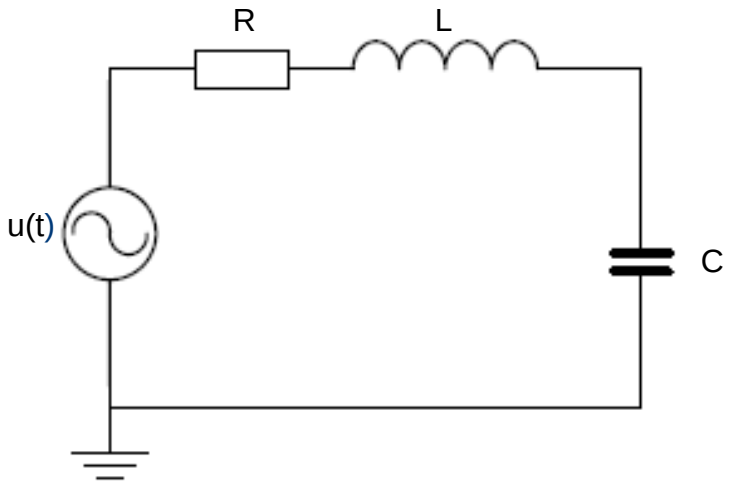
$$\theta = \arctan\left(\frac{\omega L - \frac{1}{\omega C}}{R}\right)$$

RLC Reihenschwingkreis

Resonanzfall:

$$\boldsymbol{\omega = \omega_0}$$

ω - Anregungsfrequenz
ω_0 - Eigenfrequenz des Reihenschwingkreises



Gesamtimpedanz:

$$\underline{Z} = R + j\omega L + \frac{1}{j\omega C} = \left|\underline{Z}\right| \cdot e^{j\theta}$$

Strom- und Spanungsverläufe:

$$\hat{I} = \frac{\hat{U}}{\left|\underline{Z}\right|} \qquad \underline{U}_L = \hat{I} \cdot j\omega L \qquad \underline{U}_C = \frac{\hat{I}}{j\omega C}$$

Phasenverschiebung zwischen Strom und Spannung:

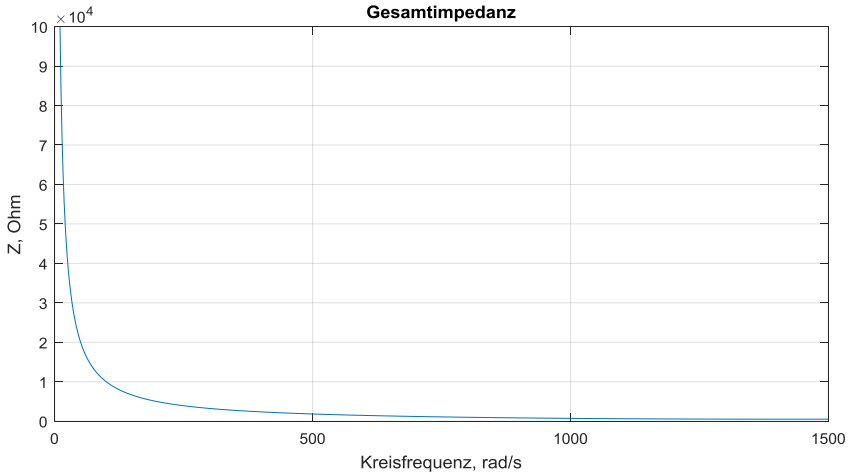
$$\theta = \arctan\left(\frac{\omega L - \frac{1}{\omega C}}{R}\right)$$

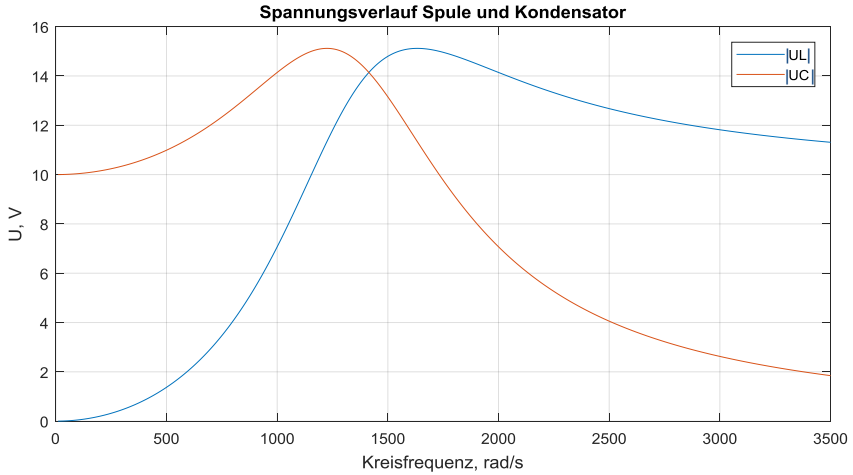
RLC Reihenschwingkreis

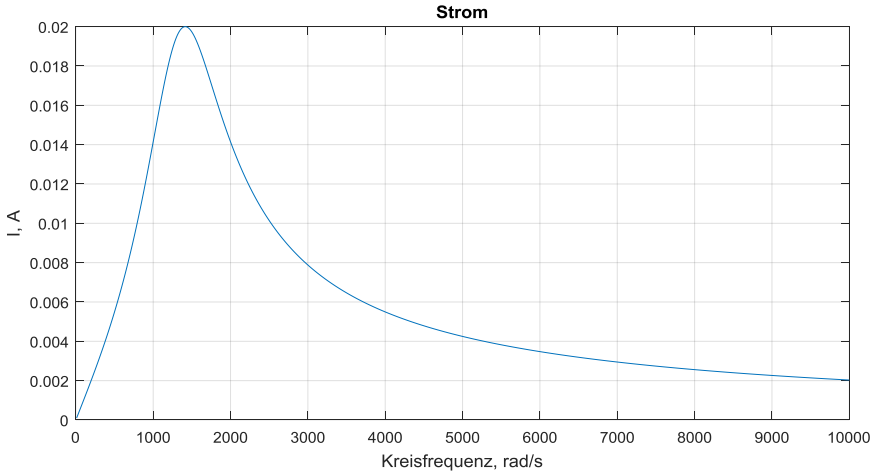
```
% Definition von Konstanten:
U = 10;
R = 500;
L = 0.5;
C = 0.000001;
w = 10:2:10000;

% Berechnungen
Z = abs(R+(w.*L-1./(w.*C))*j); % Gesamtimpedanz
I = U./Z; % Amplitude des Stromes
UL = I.*w.*L; % Verlauf der Spulenspannung
UC = I./(w.*C); % Verlauf der Kondensatorspannung
tan_phi = (w.*L-1./(w.*C))/R; % Phasenverschiebungswinkel

subplot(221) % mehrere Kurven in einem Diagrammen
plot(w,Z) % Verlauf der Gesamtimpedanz zeichnen
title('Gesamtimpedanz'); % Titel für den Verlauf im subplot(221)
xlabel('Kreisfrequenz, rad/s'); % Beschriftung der X Achse
ylabel('Z, Ohm'); % Beschriftung der Y Achse
xlim([0,1500]); % Darstellungsbereich der X Achse eingrenzen
grid% Gitternetzlinien einfügen
```

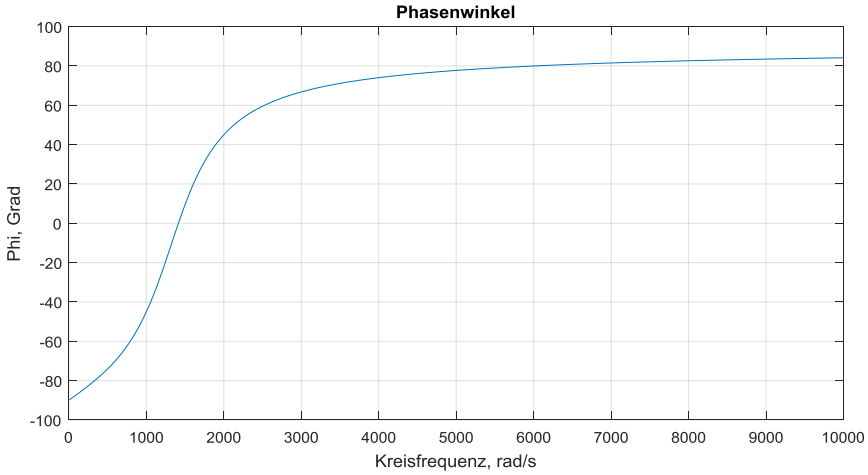
```
subplot(222)
plot(w,UL)
hold
plot(w,UC)
title('Spannungsverlauf Spule und Kondensator');
xlabel('Kreisfrequenz, rad/s');
ylabel('U, V');
xlim([0,3500]);
legend('UL','UC');
grid

subplot(223)
plot(w,I)
title('Strom');
xlabel('Kreisfrequenz, rad/s');
ylabel('I, A');
grid

subplot(224)
plot(w,atan(tan_phi)*180/pi)
title('Phasenwinkel');
xlabel('Kreisfrequenz, rad/s');
ylabel('Phi, Grad');
grid
```

RLC Reihenschwingkreis







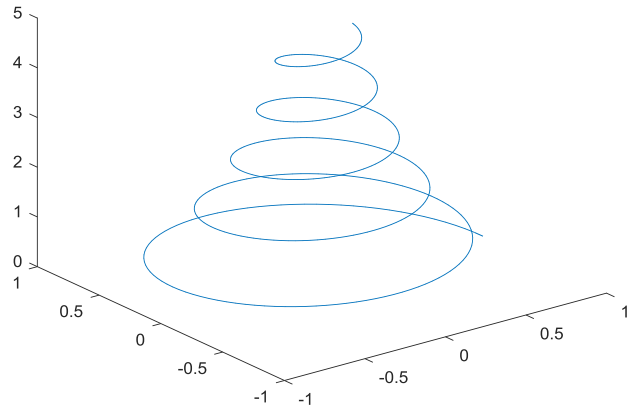


Grafische Darstellung. 3D

3D Funktion zeichnen

- **plot3(x,y,z)** – Linienplot

```
>> t = 0:0.01:5;
>> x = exp(-t/3).*cos(2*pi*t);
>> y = exp(-t/3).*sin(2*pi*t);
>> plot3(x,y,t)
```



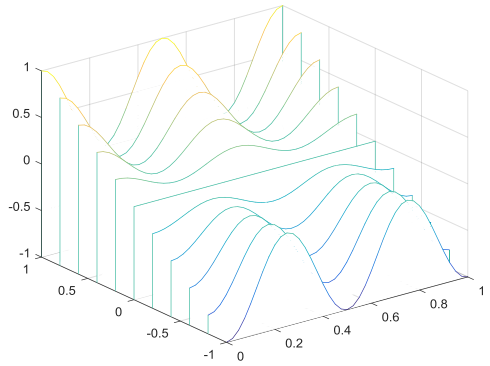
- **[X,Y] = meshgrid(x,y)** Gittermatrix

Dem Definitionsbereich der x und y werden Gitterpunkte, in denen die Funktion ausgewertet wird, entnommen.

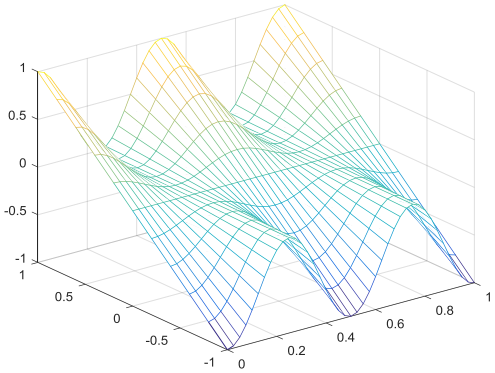
```
>> x = 0:0.025:1;          % Definitionsbereich für x
>> y = -1:0.2:1;           % Definitionsbereich für y
>> [X,Y] = meshgrid(x,y);  % Gittermatrix
>> Z = Y.*cos(2*pi*X).^2;  % Funktionsberechnung für ausgewählte Gitterpunkte
>> surf(X,Y,Z)             % Grafik zeichnen in 3D (surf als Beispiel)
```

Diagrammtypen

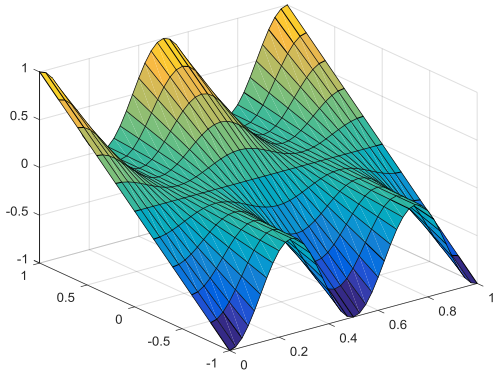
- **waterfall(X,Y,Z)**
  Liniennetz mit parallelen Linien



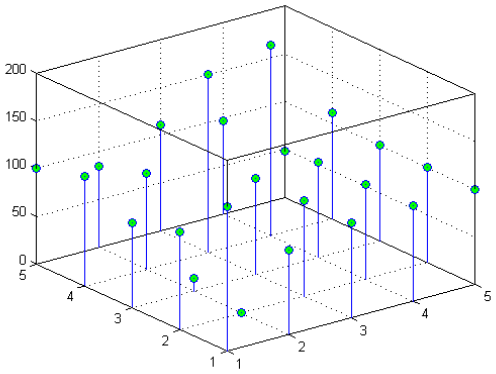
- **mesh(X,Y,Z)**
  Maschennetzdiagramm mit Gitternetzlinien



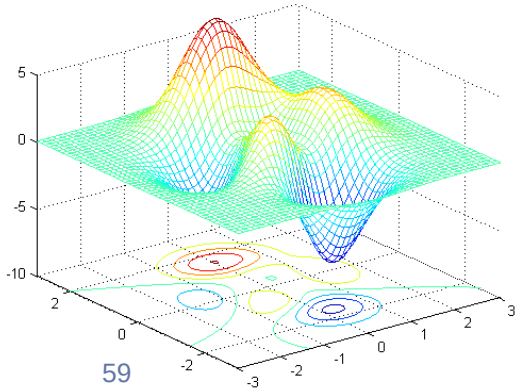
- **surf(X,Y,Z)**
  Oberflächendiagrammen mit Farbschattierungen



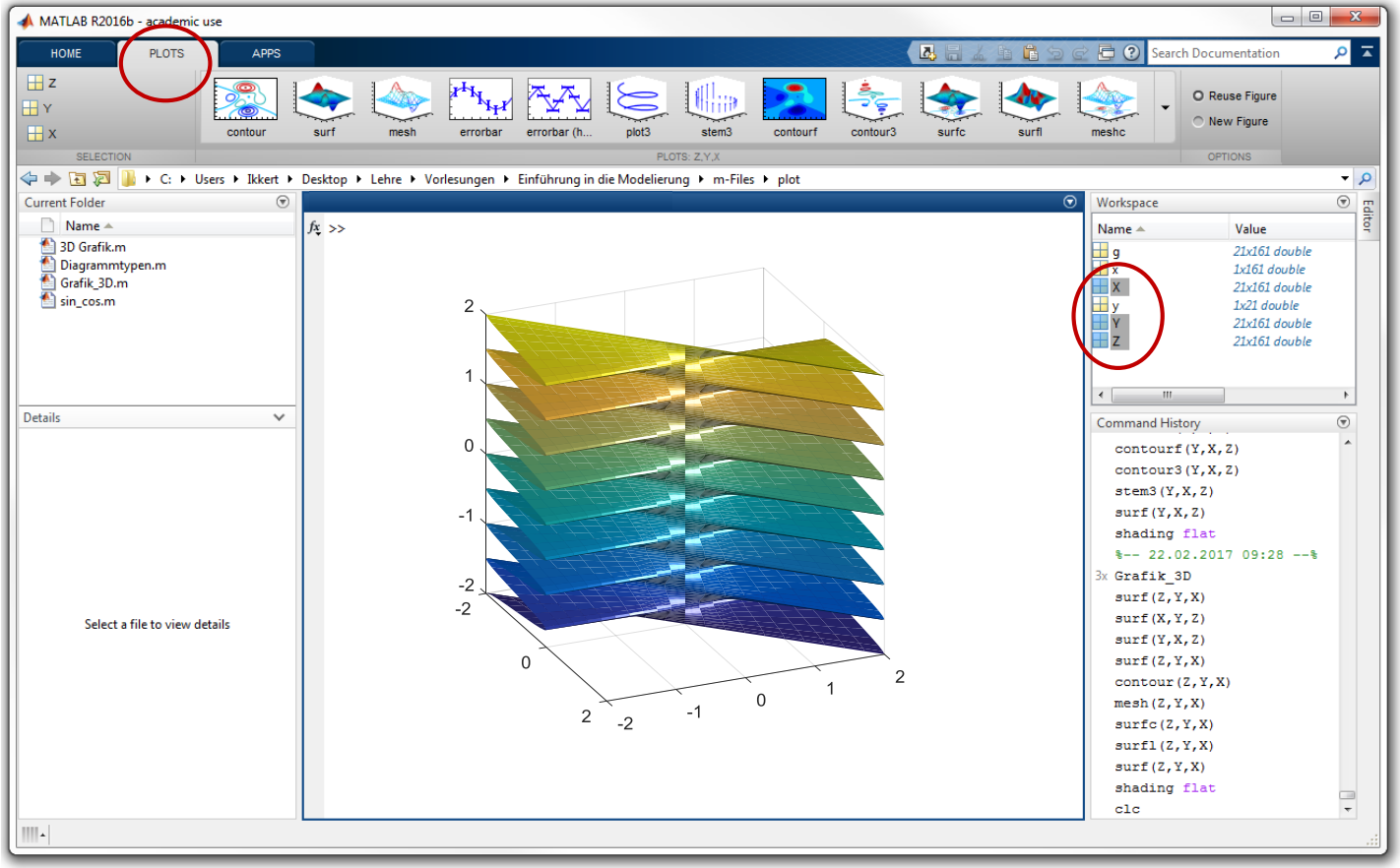
- **stem3(X,Y,Z)**
  Linien in Z-Richtung über x, y



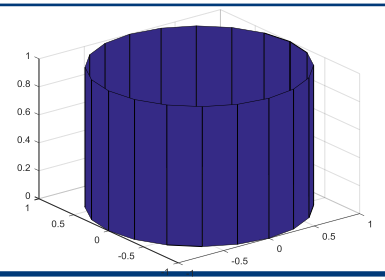
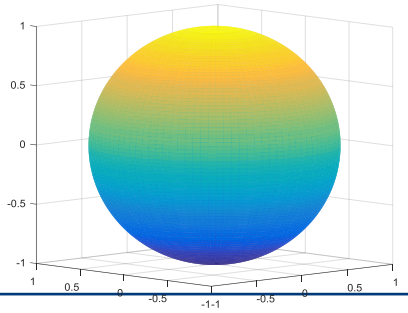
- **meshc(X,Y,Z)**
  Maschennetzdiagramm Gitternetz- plus Isolinien



Interaktives Plotten

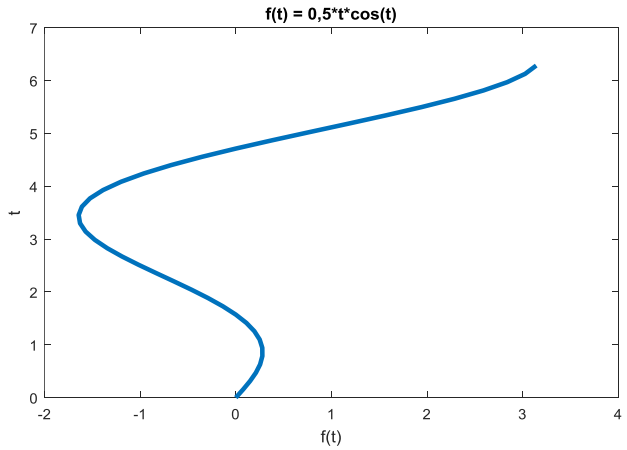


3D Objekte durch Rotation

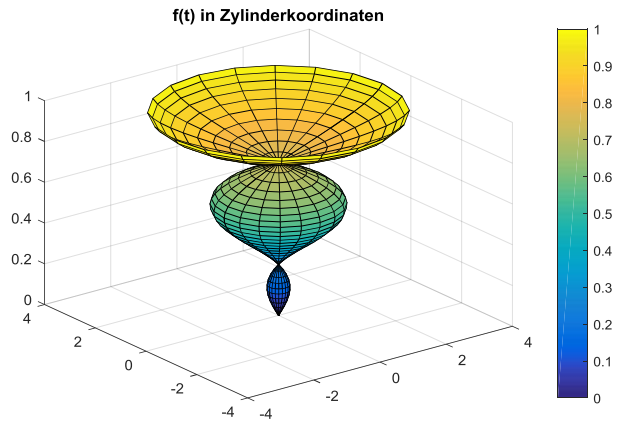
Objekt	Befehl	Beschreibung	
Zylinder	[X,Y,Z] = cylinder [X,Y,Z] = cylinder(r)	Erzeugt im Grafikfenster ein Zylinder mit dem Radius r	
Kugel	[X,Y,Z] = sphere [X,Y,Z] = sphere(n)	Erzeugt im Grafikfenster eine Kugel aus nxn Flächen	

- Die Objekte werden mit **mesh(X,Y,Z)** oder **surf(X,Y,Z)** grafisch ausgegeben
- Befehle **cylinder** oder **sphere** zeichnen Objekte in vordefinierter Größe

3D Objekte durch Rotation




```
>> t = 0:pi/20:2*pi;
>> f = 0.5*t.*cos(t);

>> subplot(2,1,1)
>> plot(f,t,'LineWidth',3);
>> xlabel('f(t)');
>> ylabel('t');
>> title('f(t) = 0,5*t*cos(t)');
```




```
>> subplot(2,1,2)
>>[X,Y,Z] = cylinder(0.5*t.*cos(t));
>> surf(X,Y,Z);
>> title('f(t) in Zylinderkoordinaten');
>> colorbar
```

Hilfreiche Befehle

Befehl	Funktion	Beschreibung
view(phi,r)	definiert den Beobachtungspunkt	durch Winkel phi und Höhe r
view([x,y,z])		durch x-, y- und z - Koordinaten
axis([x_{min},x_{max},y_{min},y_{max}])	Achsen Skalierung	Achsenbegrenzung durch max- und min-Werte
axis equal		alle Achsen gleich skaliert
axis tight		für minimale Grafikfläche
shading interp	Schattierung	weiche Farbübergänge, interpoliert
shading flat		Kein sichtbarer Farbübergang
box on	Diagramm Umrandung	zeichnet die Umrandung
box off		löscht bereits vorhandene Umrandung
colormap(name)	Farben einstellen	Wahl der Farbtabelle
caxis(farbe_min,farbe_max)		Skalierung der Farbe

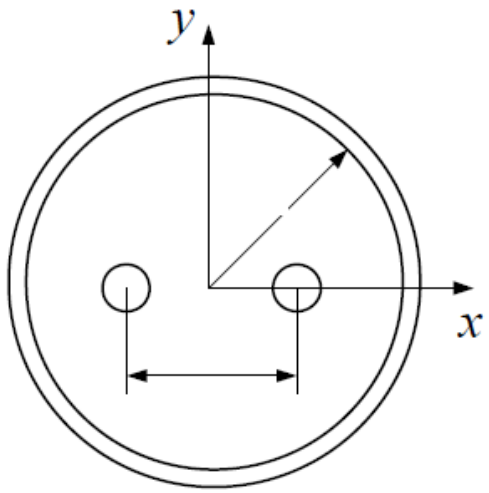
Weitere Einstellungen und Befehlsübersicht: View (Grafikfenster) oder help graph3d

Potentialfeld eines zweiadrigen Kabels

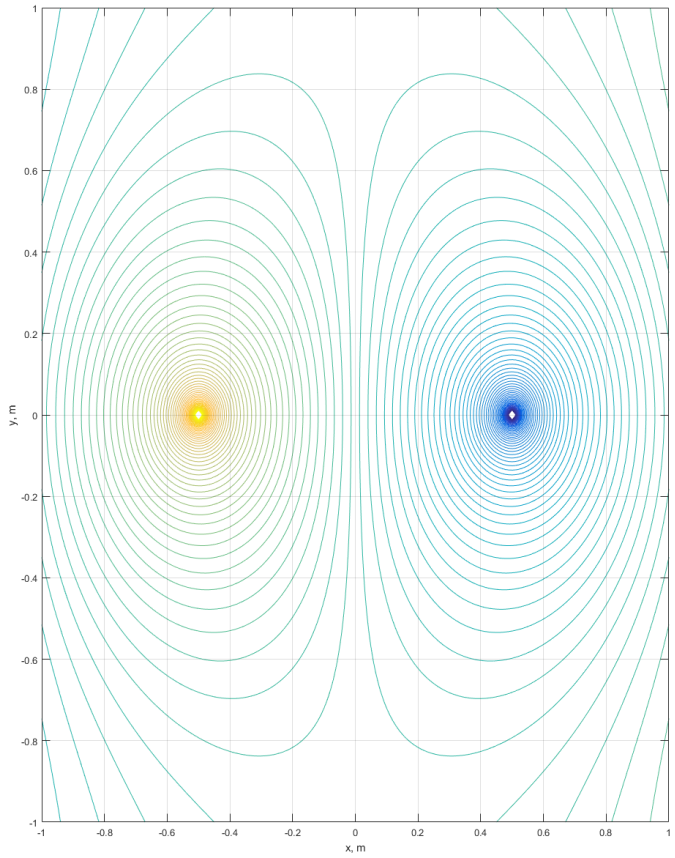
Potentialfeld eines beliebigen Punktes:

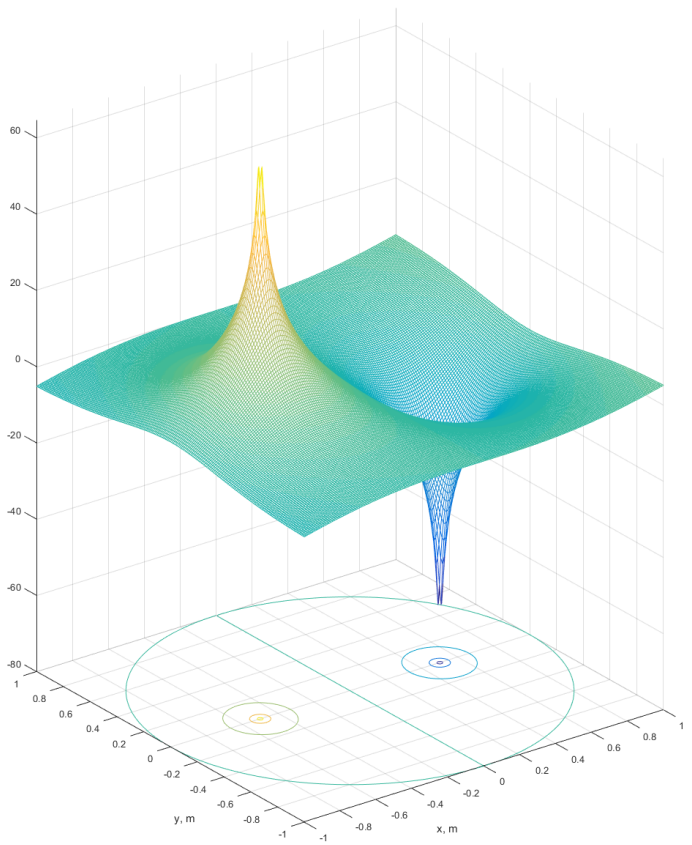
$$\Phi = A \cdot \left[ln \frac{\left((x-a)^2+y^2\right)}{(x+a)^2+y^2} + ln \frac{\left((x+b)^2+y^2\right)}{(x-b)^2+y^2} \right], \varphi(x=0) = 0$$

```
clc
x = -1:0.01:1; % Definitionsbereich für x
y = -1:0.01:1; % Definitionsbereich für y
A = 7.9; a = 0.5; b = 2;
[X,Y]= meshgrid(x,y); % Gittermatrix

arg1 =((X-a).^2+Y.^2)./((X+a).^2+Y.^2);
arg2 =((X+b).^2+Y.^2)./((X-b).^2+Y.^2);
PHI = A*(log(arg1)+log(arg2)); % Funktionsberechnung für ausgewählte
Gitterpunkte
subplot(121)
contour3(X,Y,PHI,100) % Grafik zeichnen in 3D (contour3)
xlabel('x, m')
ylabel('y, m')

subplot(122)
meshc(X,Y,PHI) % Grafik zeichnen in 3D (meshc)
xlabel('x, m')
ylabel('y, m')
grid on
```



Potentialfeld eines zweiadrigen Kabels





Kontrollstrukturen

Definition

Kontrollstrukturen beeinflussen gezielt den Ablaufs eines Programms.

Schleife

- **for**
- **while**

Verzweigung

- **if-ifelse-else**
- **switch-case-otherwise**

zusätzlich

- **break**
- **continue**
- **return**

for - Schleife

Zählschleife:

- Wiederholung der Anweisungen
- Integration der Zahlenwerte in die Anweisungen

for *i=wert*

Anweisung 1
……
Anweisung n

end

for *i=aw:n:ew*

Anweisung 1
……
Anweisung n

end

for - Schleife

- index = Matrix

```
>> A = [1 2;
        3 4];
>> for  i = A      % Indexzuweisung
       x = i       % Anweisung
   end
x =1               % Schleifendurchlauf 1
   3
x =2               % Schleifendurchlauf 2
   4
```

- index = Zeilenvektor

```
>> a = 1:3;
>> for  i = a      % Indexzuweisung
       x = i       % Anweisung
   end
x = 1              % Schleifendurchlauf 1
x = 2              % Schleifendurchlauf 2
x = 3              % Schleifendurchlauf 3
```

- index = mathematischer Ausdruck

```
>> x = 0;
>> for  i = 1:5    % Indexzuweisung
       x = x+i;    % Anweisung
   end
>> x
x =15
```

while - Schleife

logische Schleife:

- Wiederholung der Anweisungen, solange die logische Bedingung erfüllt ist
- Wichtig: für ein Abbruchkriterium zu sorgen

while *wert*

Anweisung 1
……
Anweisung n

end

while *ausdruck*

Anweisung 1
……
Anweisung n

end

while - Schleife

- Matrix als Bedingung

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

>> while  A                  % Endbedingung Matrix
     A(2,2) = A(2,2)-2        % Anweisung
   end

A = 1   2                    % Schleifendurchlauf 1
    3   2

A = 1   2                    % Schleifendurchlauf 2
    3   0
```

- logische Bedingung

```
>> a = 1;
>> b = 20;
>> while  (a<b)              % logischer Ausdruck
     b = b/2                 % Anweisung 1
     a = a+1                 % Anweisung 2
   end

b = 10   a = 2               % Schleifendurchlauf 1
b = 5    a = 3               % Schleifendurchlauf 2
b = 2.5  a = 4               % Schleifendurchlauf 3
```

if-elseif-else - Verzweigung

- Ausführung von Anweisungen, falls logische Bedingung nach **if** erfüllt ist
- Weitere Abfragen unter **elseif** möglich
- Ausführung von Anweisungen unter **else**, falls keine der Abfragen erfüllt sind

if *ausdruck 1*

Anweisungen

elseif *ausdruck 2*
Anweisungen

elseif *ausdruck 3*
Anweisungen

else
Anweisungen

end

if-elseif-else - Verzweigung

Prüfen des Vorzeichens jedes Matrix – Elementen mit Ausgabe der Ergebnisse:

```
m = 2;
n  = 0;
A = 10*rand(m)-5

for i = 1:(m*m)

    if A(i)<0 % logischer Ausdruck 1
        disp(['Die Zahl  ',num2str(A(i)),'  ist kleiner Null']) % Anweisung 1
        n = n+1; % Anweisung 2
    elseif A(i) == 0% logischer Ausdruck 2
        disp(['Die Zahl  ',num2str(A(i)),'  ist gleich Null']) % Anweisung 3
    else
        disp(['Die Zahl  ',num2str(A(i)),'  ist größer Null']) % Anweisung 4
    end

end
disp(['Matrix A enthält ',num2str(m*m-n),'  positive und  ',num2str(n),'  negative Elemente']);
```

if-elseif-else - Verzweigung

Antwort des Systems:

```
A =

    0.2250   -2.8132
    4.9370   -3.9420

Die Zahl  0.22495  ist größer Null
Die Zahl  4.937    ist größer Null
Die Zahl  -2.8132  ist kleiner Null
Die Zahl  -3.942   ist kleiner Null
Matrix A enthält 2  positive und  2  negative Elemente
>>
```

switch-case-otherwise

Abfrage des Wertes oder Zustands einer Variablen und Ausführung der Anweisungen je nach zutreffendem Fall.

switch *variable*

case *wert 1*
Anweisungen

case *{wert 2,wert 3}*
Anweisungen
…….
case *wert n*
Anweisungen

otherwise

Anweisungen
end

Mehrere Alternativwerte mit gleichen Anweisungen

switch-case-otherwise

```
switch x
   case 0
      disp('x ist gleich Null');
   case 1
      disp('x ist eine Eins');
   case 2
      disp('x ist eine Zwei');
   otherwise
      disp('keine Ahnung was x ist');
end
```

Systemantwort:

```
>> x = 2;
x ist eine Zwei
```

```
>> x = 10;
keine Ahnung was x ist
```

Hilfreiche Befehle

Befehl	Beschreibung	Beispiel

break	Verlassen einer Schleife. Das Programm führt dann die Anweisungen nach der Schleife weiter aus.	while x>0
		x = x/i;
		if x<=1
		break
		end
		end
		
		x = 100; i = 2;

continue	Der aktuelle Durchgang einer Schleife wird verlassen ohne Ausführung der nachfolgenden Anweisungen. Danach startet der nächste Durchgang der Schleife.	for i = x if i<0 disp('Keine Wurzel aus einer negativen Zahl'); **continue** end j = j+1; y(j) = sqrt(i) end x = [4 -7 16 9]; j = 0 ;
return	Beendet den Ablauf eines Programms	
error	Beendet das Programm und gibt die Nachricht aus	error('Es ist ein Fehler aufgetreten! Das Programm wird beendet!');

Effizienz

- Vektorisieren

```
tic
for k = 1:100001
    x(k) = (k-1)*k;
end
toc
```

Elapsed time is **0.008099** seconds.

Viel effizienter und übersichtlicher ist die Vektorisierung dieses Codes:

```
tic
n = 1:100001;
y = (n-1).*n;
toc
```

Elapsed time is **0.000807** seconds.

Funktionen

Definition

Erweiterung des vorhandenen Befehlumfangs. Mit Funktionen lassen sich:

- Parameter übergeben
- Algorithmen ausführen
- Funktionswerte berechnen
- Ergebnisse ausgeben

function [*out1,out2,…*] = funktionsname (*in1,in2,…*)
Befehle

function - Definition einer Funktion
in1,in2… - Übergabeparameter
out1,out2,… - Rückgabewerte
funktionsname - selbstdefinierter Name

Definition

- Funktionsname = Name des M-Files
- Innerhalb eines Function-Files definierte Größen sind stets lokal, d. h. sie werden nicht im Workspace abgelegt
- Variablen, die sowohl im Workspace als auch im m-File bekannt sein sollen, müssen im **Command-Fenster** und im **m-File** global deklariert werden: global x
- Funktionen können von anderen Funktionen oder Skripts aufgerufen werden
- Mit Kommentaren arbeiten
- Ein- und Ausgabeparameter:

Befehl	**Beschreibung**
nargin	Anzahl der Eingabeparameter
nargout	Anzahl der Ausgabeparameter

Funktionsaufruf

- Aufruf ohne Wertzuweisung

  funktionsname (*in1,in2,…*)

  – besitzt die Funktion mehrere Ausgabeparameter, wird der erste davon an die Variable *ans* übergeben

- Einfache Wertzuweisung

  y = funktionsname (*in1,in2,…*)

  – besitzt die Funktion einen Ausgabeparameter, wird er der Variablen *y* zugewiesen
  – besitzt die Funktion mehrere Ausgabeparameter, wird der erste davon der Variablen *y* zugewiesen

- Wertzuweisung an einen Vektor

  [y1,y2,….] = funktionsname (*in1,in2,…*)

  – Die Zahl der Vektorkomponenten [y1,y2,…] muss kleiner oder gleich der Anzahl der Ausgabeparameter sein.
  – Ausgabeparameter werden als Vektorkomponenten an den Vektor [y1,y2,…] übergeben

Variable Anzahl der Ausgabeparameter

- Mehrere Ausgabeparameter

function [*out1,out2,…*] = funktionsname (*in1,in2,…*)

- Ein Ausgabeparameter

function *out1* = funktionsname (*in1,in2,…*)

- Keine Ausgabeparameter

function funktionsname (*in1,in2,…*)

Aufgaben

1. Fakultät

Schreiben Sie eine Funktion *fakultaet*, die Fakultät einer Zahl berechnet und ausgibt.

Eingabeparameter: zahl

Ausgabeparameter: fakultaet

Prüfen Sie den Definitionsbereich der Eingabeparameter und geben Sie die Ergebnisse über eine kurze formatierte Ausgabe aus.

2. Lineare Funktion

Schreiben Sie eine Funktion *gerade*. Die Funktion soll aus zwei Punkte-Paaren (x1,y1) und (x2,y2) die Steigung m und Achsenabschnitt b bestimmen und ausgeben. Außerdem soll die lineare Funktion grafisch dargestellt werden.

Lineare Geradengleichung lautet y = m*x + b

1. Fakultät

```
% Die Funktion fakultaet soll Fakultät einer Zahl berechnen und ausgeben,
% den Definitionsbereich der Eingabeparameter überprüfen und
% die Ergebnisse formatiert ausgeben.

function [n] = fakultaet(param)
n = 1;
if param < 0 | round(param)~= param
    error('Eingabe außerhalb des zulässigen Definitionsbereichs');
else
    for i = 1:param
            n = n*i;
    end
end

disp([ num2str(param,'%d'),'! = ',num2str(n,'%d')])

end
```

Funktionsaufruf:

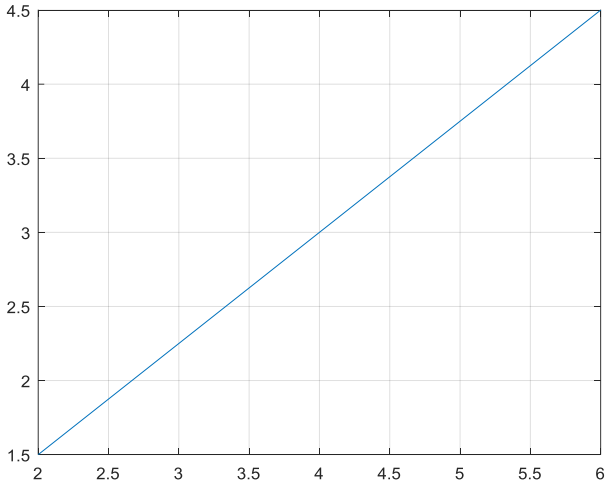
```
>> fakultaet(6);
6! = 720
```

2. Lineare Funktion

```
% Die Funktion gerade soll aus zwei Punkte-Paaren (x1,y1) und (x2,y2)
% die Steigung m und Achsenabschnitt b bestimmen und ausgeben
% lineare Geradengleichung lautet y = m*x + b

function [m,b] = gerade (x1,y1,x2,y2)

m = (y2-y1)/(x2-x1)
b = y1-m*x1
x = linspace(x1,x2,100);
y = linspace(y1,y2,100);
plot(x,y);
grid on
end
```

<u>Funktionsaufruf</u>:

```
>> gerade(2,1.5,6,4.5);

m = 0.7500
 b = 0
```



Variable Anzahl der Eingabeparameter

function [*out1,out2,…*] = funktionsname (**varargin**)

- Typ der Eingabeparameter: **Cell Array**
- Typumwandlung: **sell2mat**

varargin =

[] [] [] ……

Aufgabe:

- Schreiben Sie eine Funktion *mittelwert*, die den Mittelwert der variablen Anzahl der Eingabeparameter bestimmt und ausgibt.
- Vektorisieren Sie die Berechnung

Mittelwert

```
% Die Funktion mittelwert soll den Mittelwert der variablen Anzahl der Eingabeparameter
% bestimmen und ausgeben.
function [m] = mittelwert(varargin)

n = nargin;
m = 0;

% Berechnung über for Schleife
for i = 1:n
m = (m+cell2mat(varargin(i)));
end

% vektorisierte Berechnung
m = sum(cell2mat(varargin));
m = m/n;
disp(['Mittelwert = ', num2str(m,'%f')])
end
```

Funktionsaufruf:

```
>>mittelwert(1.7,2,3.7,4,1.3);

Mittelwert = 2.540000
```

Funktionen als Eingabeparameter

Beim Aufruf einer Funktion können nicht nur konstante Parameter, Variablen oder Matrizen übergeben werden, sondern auch weitere Funktionen:

- Function Handle

Fhandle = @funk

- Fhandle stellt den Zeiger auf eine Funktion dar
- Wird durch den Operator @ vor dem Funktionsnamen erzeugt
- Function Handle kann auf bereits definierte Funktionen (sin, exp etc.) oder selbstdefinierte Funktionen angewendet werden und
- Funktionswerte berechnen, auf die der Handle verweist:

```
>>handle = @sin
>>handle(pi/2) = 1
```

Funktionen als Eingabeparameter

- Syntax

- **vordefinierte Funk.**
  @sin

- **selbsterstellte Funk.**
  @fun
  function y = fun(x)

- **anonyme Funk.**
  @(x) (x.^2+x-1)

(Definition + erstellen eines Function Files !)

- Auswertung

- **feval**
Führt eine Funktion (handle) aus

feval(fhandle, $x_1 \ldots x_n$)
>>feval(@sin,x)

- **eval**
Führt eine Funktion aus, die als String eingegeben wurde

eval('Funktion')
>>eval('x.^2+x-1')

Aufgaben

1. Grafik

Schreiben Sie eine Funktion *grafik*, die eine Funktion (handle) grafisch abbildet.

Eingabeparameter: Funktion, Darstellungsbereich, Titel

2. Ableitung

Schreiben Sie eine Funktion *ableitung*, die die Ableitung unterschiedlicher Funktionen im vorgegebenen Bereich berechnet. Testen Sie Funktionsaufruf für vordefinierte, anonyme und selbstdefinierte Funktionen.

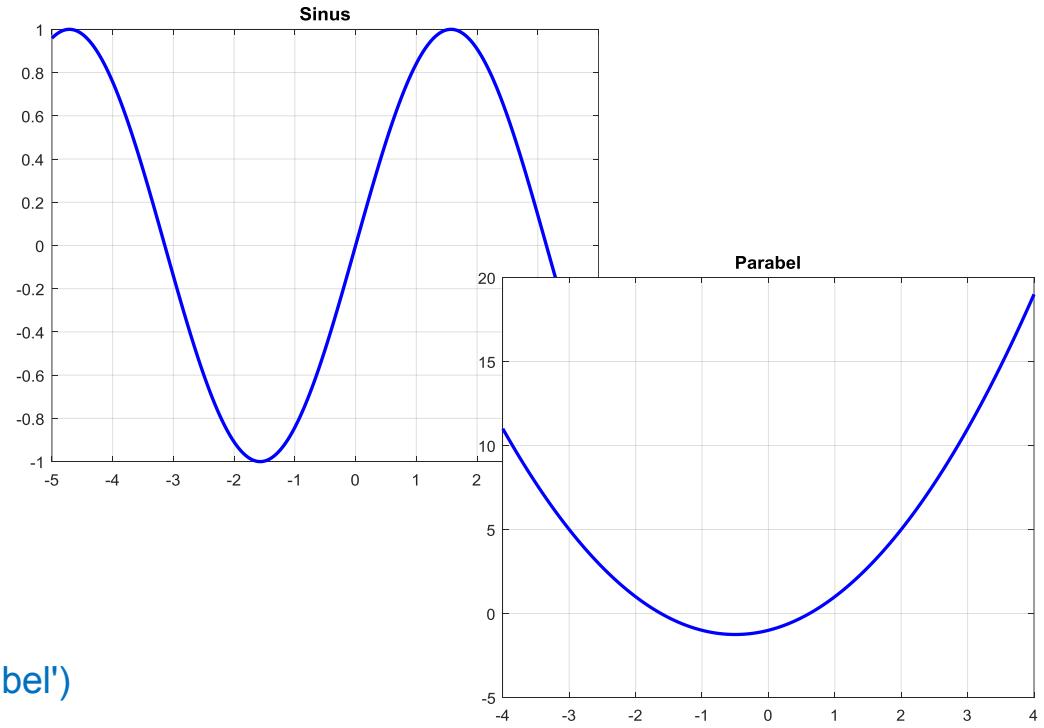
$$f' = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

1. Grafik

```
% Die Funktion grafik soll eine Funktion (handle) grafisch abbilden.
% Eingabeparameter: Funktion, Darstellungsbereich, Titel
function grafik( f,x1,x2,titel )

x = linspace(x1,x2,1000);
y = f(x);

plot(x,y,'b','LineWidth',2)
grid on
title(titel);

end
```

Funktionsaufruf:

```
>> grafik(@sin,-5,4,'Sinus')
>> grafik(@(x) (x.^2+x-1),-4,4,,Parabel')
```



2. Ableitung

```
% Die Funktion ableitung soll die Ableitung unterschiedlicher Funktionen im vorgegebenen Bereich
% berechnen. Eingabeparameter: x, deltax, Funktion

function df = ableitung( x,deltax,f)
df = (f(x+deltax)-f(x))/deltax;
end
```

$$f' = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Funktionsaufruf:

```
% Festlegung des Berechnungsintervalls x und Schrittweite deltax
>> x = 0:0.1:2;
>> deltax = 0.1;
```

- vordefinierte Funktionen

```
>> df = ableitung(x,deltax,@sin)
```

- anonyme Funktionen

```
>> f = @(x) (x.^2+x-1)
>> df = ableitung(x,deltax,f)
>> df = ableitung(x,deltax, @(x) (x.^2+x-1))
```

Workspace

Name	Value	Size	Max	Min
deltax	0.1000	1x1	0.1000	0.1000
df	*1x21 double*	1x21	0.9983	-0.4609
x	*1x21 double*	1x21	2	0

2. Ableitung

- selbstdefinierte Funktionen

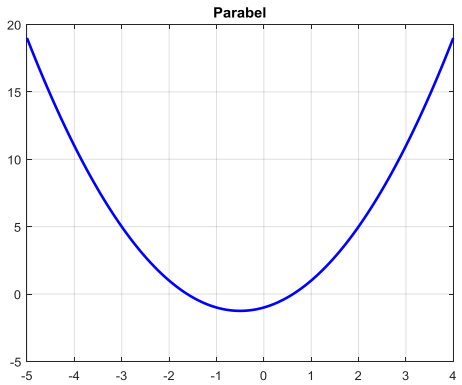
```
% zuerst wird eine Funktion f erstellt, die das Berechnungsintervall x einliest und die Funktionswerte y der selbsterstellten
% Funktion berechnet und zurück gibt
function y = f(x)
y = x.^2+x-1;
end

% Berechnung der Ableitung
>> df = ableitung(x,deltax,@f)
```

Nullstellen einer Funktion

- grafisch ermitteln

$$f(x) = x^2 + x - 1$$

$$\begin{cases} x_1 \approx -1{,}7 \\ x_2 \approx 0{,}7 \end{cases}$$



- Nullstellenberechnung mit **fzero**

- fzero(f,x0) Berechnung **einer** Nullstelle einer Funktion f in der Nähe eines Wertes x0

- fzero(f,[x1 x2]) Berechnung **einer** Nullstelle einer Funktion f im Intervall [x1 x2]

Nullstellen einer Funktion

$$f(x) = x^2 + x - 1$$

- Nullstellensuche in der Nähe von $x_1 \approx -1{,}7$

```
>> y = @(x) (x.^2+x-1);
>> fzero(y,-1.7)
ans = -1.6180
>> feval(y,ans)
ans = -2.2204e-16
```

- Nullstellensuche im Intervall [0 1]

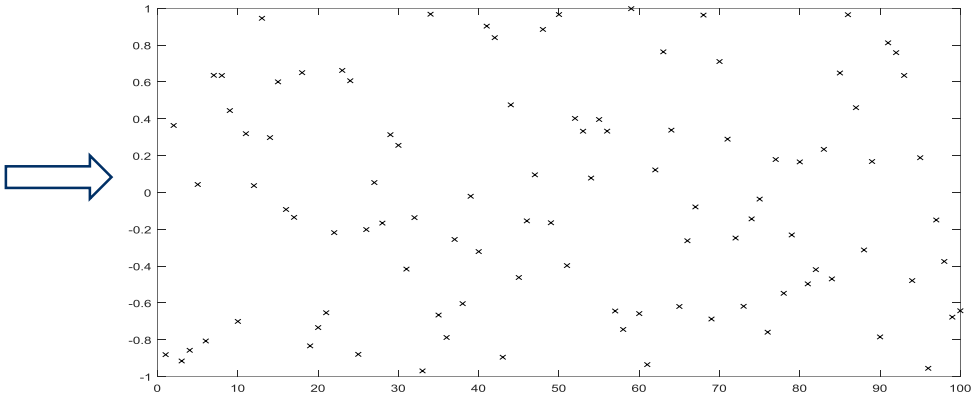
```
>> y = @(x) (x.^2+x-1);
>> fzero(y,[0 1])
ans = 0.6180
>> feval(y,ans)
ans = 0
```

Polynome und Interpolation

Definition

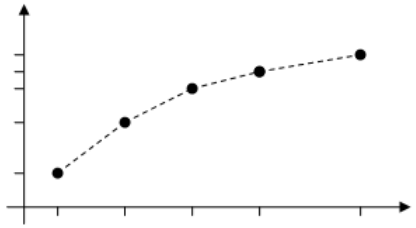
Reihe aufgenommener Messwerte

-8.3200532e-01
1.5685989e+00
1.2797336e+00
-2.0801592e+00
-6.8348830e-01
-4.8451287e+00
4.8406372e+00
-3.3283159e+00
-3.9378366e+00
-1.2759026e+00
-3.0188160e+00



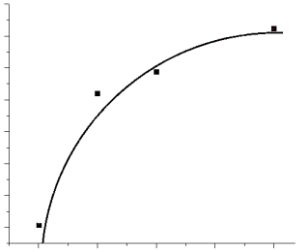
<u>Messwerte analytisch beschreiben durch:</u>

- Interpolation



- gegebene Messwerte werden exakt getroffen
- der gesuchte Wert liegt zwischen den Messwerten

- Approximation



- gegebene Messwerte werden nicht exakt getroffen
- möglichst genaue Näherung

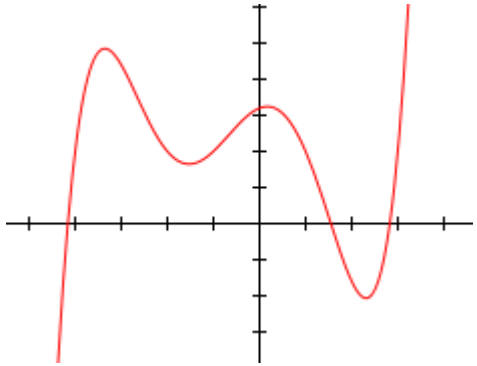
Definition

Polynom n-ten Grades

$$p(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \cdots + a_0 = \sum_{k=0}^{n} a_k \cdot x^k$$

Wird in Matlab durch den Zeilenvektor dargestellt:

$$\boldsymbol{p} = [\boldsymbol{a_n}, \boldsymbol{a_{n-1}}, \ldots \boldsymbol{a_2}, \boldsymbol{a_1}, \boldsymbol{a_0}]$$



- $(n+1)$ - Anzahl der Elemente
- Koeffizienten in absteigender Reihenfolge

Beispiel:

$p(x) = 2x^3 + x + 4$ →
p = [2,0,1,4]

Polynomoperationen

p1 = [1 7 3 -5]

$$p_1(x) = x^3 + 7x^2 + 3x - 5$$
$$p_1'(x) = 3x^2 + 14x + 3$$
$$\int p_1(x) = \frac{1}{4}x^4 + \frac{7}{3}x^3 + \frac{3}{2}x^2 - 5x$$

Befehl	Beschreibung	Beispiel
roots(p)	n Nullstellen aus Polynom-koeffizienten berechnen	>> r = **roots(p1)** r = -6.4103 -1.2259 0.6362
poly(p)	(n+1) Polynomkoeffizienten aus n Nullstellen berechnen	>> p = **poly**(r) p = 1.0000 7.0000 3.0000 -5.0000
polyval(p,x)	Polynomwerte an der Stelle x berechnen	>> x = [-1 0 1] >> yp = **polyval**(p1,x) yp = -2 -5 6
polyder(p)	Koeffizienten der Ableitung des Polynoms p berechnen	>> d = **polyder**(p1) d = 3 14 3
polyint(p)	Koeffizienten des Integrals des Polynoms p berechnen	>> i = **polyint**(p1) i = 0.2500 2.3333 1.5000 -5.0000 0

Grundrechenarten

p1 = [1 7 3 -5] p2 = [1 2 3]

- Addition

$$p_1(x) = x^3 + 7x^2 + 3x - 5$$
$$p_2(x) = x^2 + 2x + 3$$
$$\boldsymbol{p_1 + p_2} = x^3 + 8x^2 + 5x - 2$$

>> **p1+[0,p2]**
ans = 1 8 5 -2

>> p1+p2
error: Matrix dimensions must agree.

- Multiplikation

$$\boldsymbol{p_1 \cdot p_2} = x^5 + 9x^4 + 20x^3 + 22x^2 - x - 15$$

>> **conv(p1,p2)**
ans = 1 9 20 22 -1 -15

- Division

$$\frac{p_1(x)}{p_2(x)} = (x+5) + \frac{-10x - 20}{x^2 + 2x + 3}$$

r(x) — q(x)

>> **[q,r] = deconv(p1,p2)**

q = 1 5

r = 0 0 -10 -20

Partialbruchzerlegung

$$\frac{p_1(x)}{p_2(x)} = (x+5) + \frac{-10x-20}{x^2+2x+3}$$

$$\frac{p_1(x)}{p_2(x)} = q(x) + \frac{a_1}{(x-x_1)} + \frac{a_2}{(x-x_2)}$$

a_1, a_2 - Zählerkoeffizienten

x_1, x_2 - Nullstellen des Nennerpolynoms

[A,N,q] = residue(p1,p2) – komplexe Partialbruchzerlegung

$A = [a_1; a_2]$ - Zählerkoeffizienten

$N = [x_1; x_2\,]$ - Nullstellen des Nennerpolynoms

q – ganzrationaler Anteil q(x) als Zeilenvektor

Definitionen

$$(x_1, y_1), (x_2, y_2), \ldots \ldots (x_n, y_n)$$

gesucht wird ein Polynom:

- minimalen Grades (maximal (n-1))
- verläuft durch alle Messpunkte
- funktionaler Zusammenhang für die Zwischenwerte

polyfit(x,y,n) – anpassen eines Polynoms an die Messpunkte

$[x_1, x_2 \ldots \ldots x_n]$
$[y_1, y_2 \ldots \ldots y_n]$ } - Messwerte

n - Polynomgrad

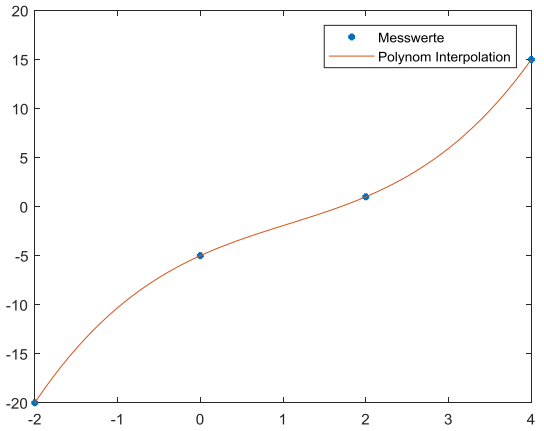
Gegeben sind folgende Messwertepaare:

X	-2	0	2	4
Y	-20	-5	1	15

Gesucht wird ein Polynom dritten Grades, das durch alle gegebene Punkte geht und einen funktionalen Zusammenhang der Messreihe darstellt.

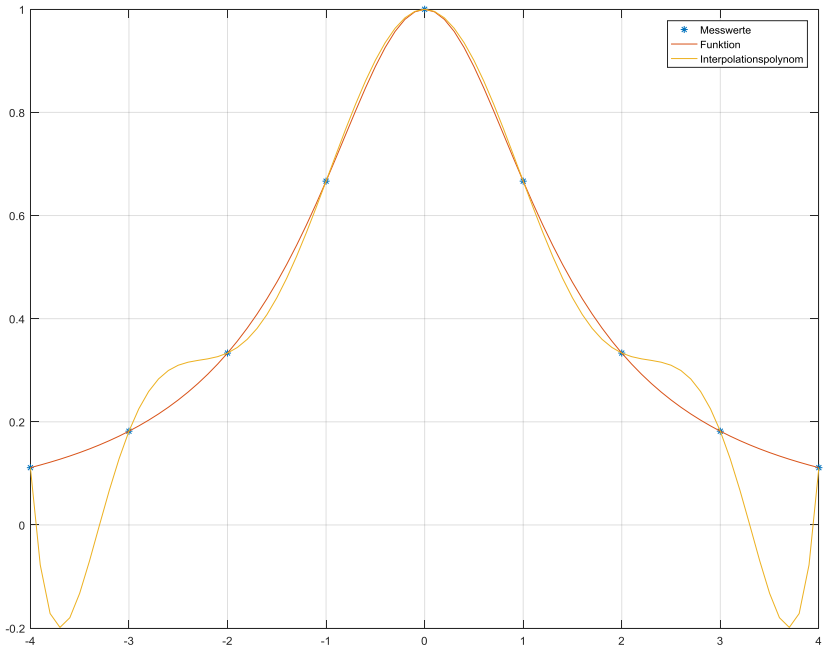
```
>> x = [-2,0,2,4];
>> y = [-20,-5,1,15];
% Berechnung der Koeffizienten des Interpolationspolynoms dritten Grades
>> p = polyfit(x,y,3);
p =   0.3542   -1.1250    3.8333   -5.0000
% Berechnung der Polynomwerte für den Bereich [-2;4]
>> pw_x = polyval(p,[-2:.1:4]);
>> plot(x,y,'*','LineWidth',2,'MarkerSize',4)
>> hold on
>> plot([-2:.1:4],pw_x)
>> legend('Messwerte','Polynom Interpolation')
>> hold off
```

$$p = 0.35x^3 - 1.13x^2 + 3.83x - 5$$



Gegeben ist eine Funktion $y = \frac{2}{(x^2+2)}$, die für $x = -4:4$ durch ein Polynom minimalen Grades interpoliert werden kann:

```
% Interpolationspunkte
x = -4:4;
y = 2./(x.^2+2);
% Berechnung der ursprünglichen Funktion
xf = -4:0.1:4;
yf = 2./(xf.^2+2);

% Berechnung der Koeffizienten des Interpolationspolynoms
p = polyfit(x,y,length(x)-1);
% Berechnung der Polynomwerte
yp = polyval(p,xf);


plot(x,y,'*')
grid on
hold on
plot(xf,yf);
plot(xf,yp);
legend('Messwerte','Funktion','Interpolationspolynom');
```



Interpolationsarten

- Lagrange Interpolation

Polynominterpolation mit **äquidistanten** Punkten

```
x = linspace(a,b,n)
y = f(x)
p = polyfit(x,y,(n-1))
```

- Chebyshev Interpolation

Verteilung der Berechnungspunkte zwischen (a,b):

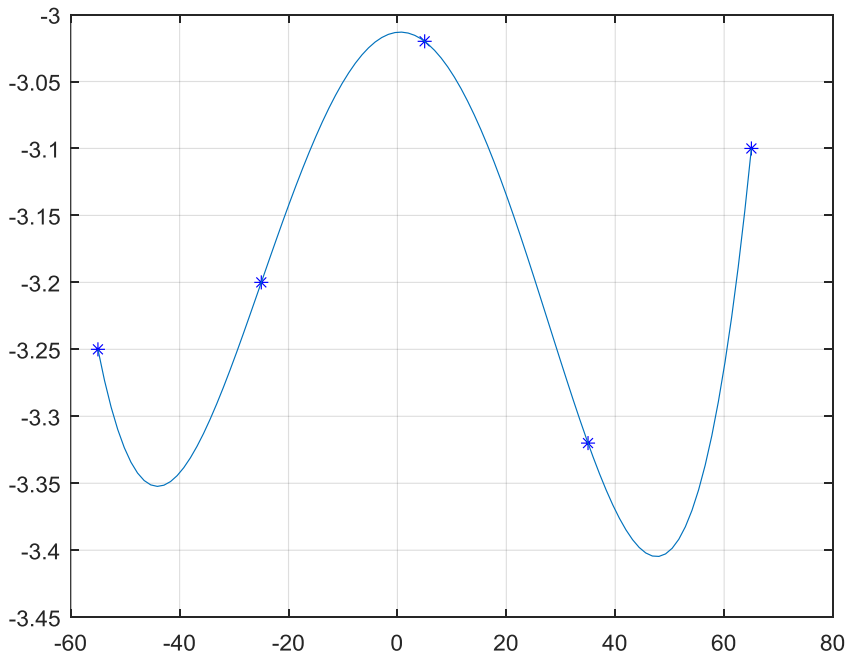
$$x_k = \frac{a+b}{2} - \frac{b-a}{2} \cdot \cos\left(\frac{2k+1}{n+1}\frac{\pi}{2}\right) \quad k = 0, \dots, n$$

Anzahl der Interpolationspunkte

```
y = f(x)
p = polyfit(x,y,(n-1))
```

Lagrange Interpolation

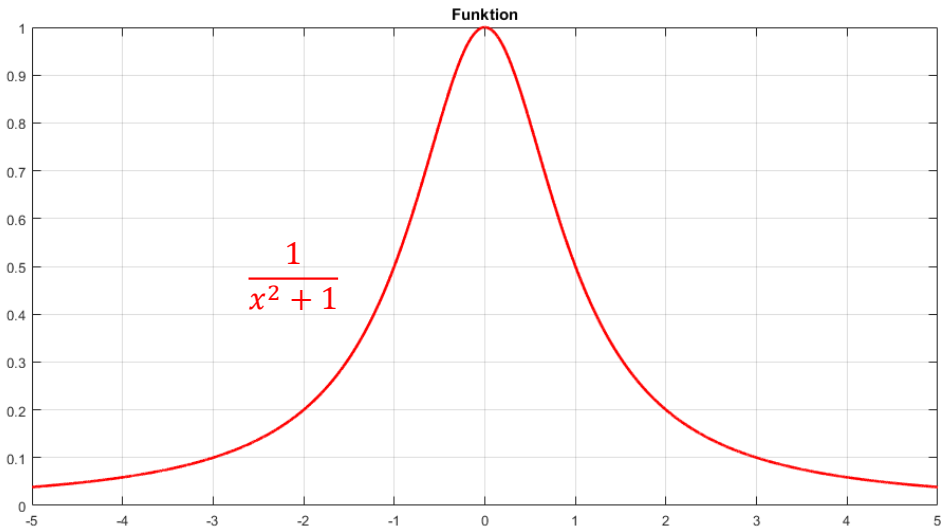
```
% äquidistante Interpolationspunkte
x = [-55 -25 5 35 65];
y = [-3.25 -3.2 -3.02 -3.32 -3.1];

% Berechnung der Koeffizienten des Interpolationspolynoms
polynom = polyfit(x,y,4);
xl = linspace(x(1),x(end),100);

% Berechnung der Polynomwerte
pl = polyval(polynom,xl);

plot(x,y,'b*')
hold on
plot(xl,pl)
grid on
```



Chebyshev Interpolation

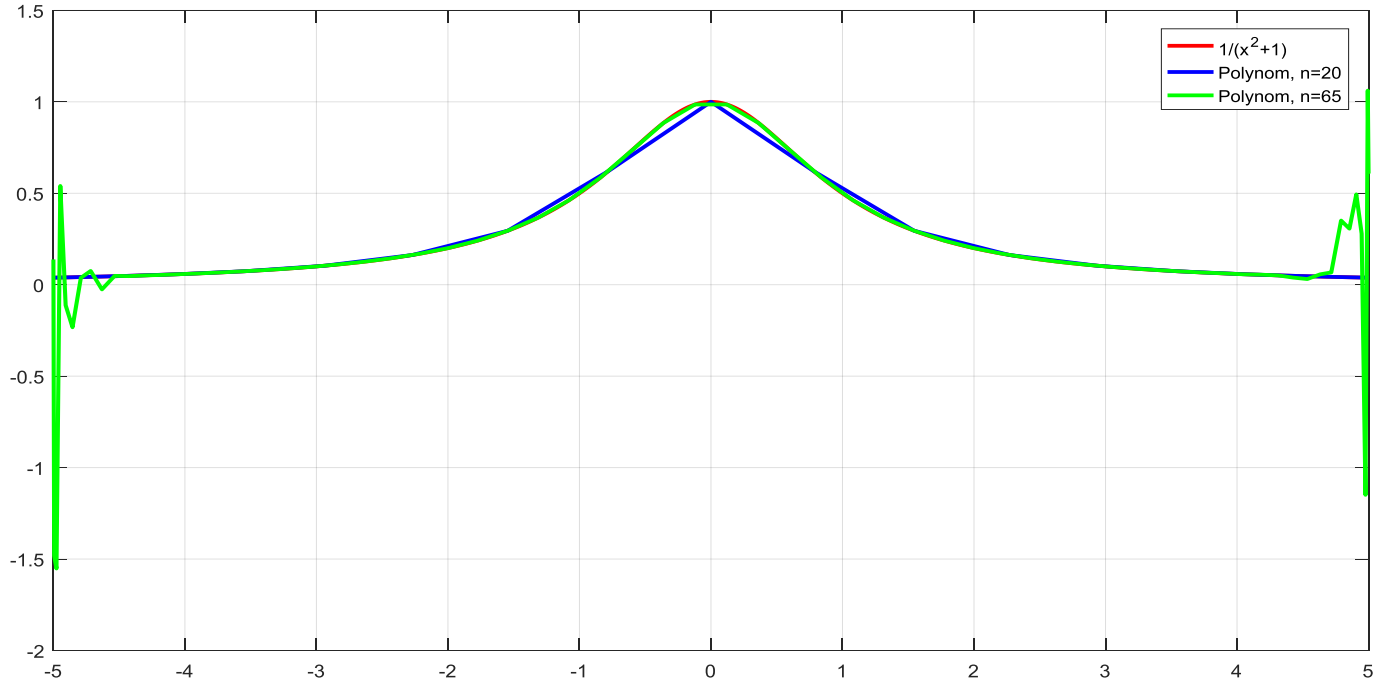
```
% Interpolationsgrenzen
a = -5;
b = 5;

% die zu interpolierende Funktion
x = linspace(-5,5,500);
y = 1./(x.^2+1);

% Polynomgrad 20
n = 21;
% Interpolationspunkte
xch = (a+b)*0.5-(b-a)*0.5*cos(pi*[0:n]/n);
ych = 1./(xch.^2+1);
% Berechnung des Interpolationspolynoms
polynom = polyfit(xch,ych,(n-1));
pch = polyval(polynom,xch);
```

$$x_k = \frac{a+b}{2} - \frac{b-a}{2} \cdot \cos\left(\frac{2k+1}{n+1}\frac{\pi}{2}\right) \quad k = 0, \dots, n$$

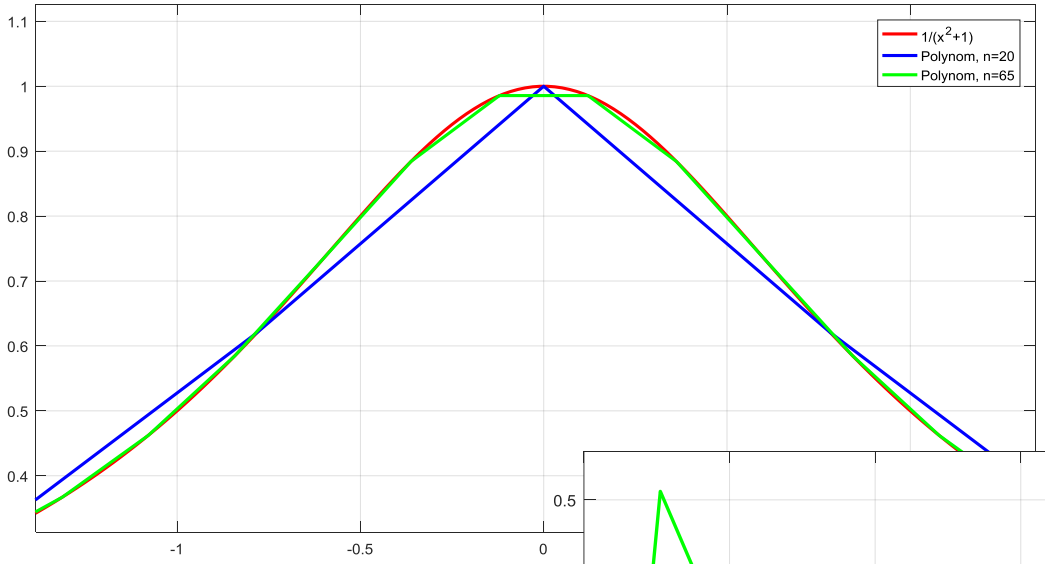


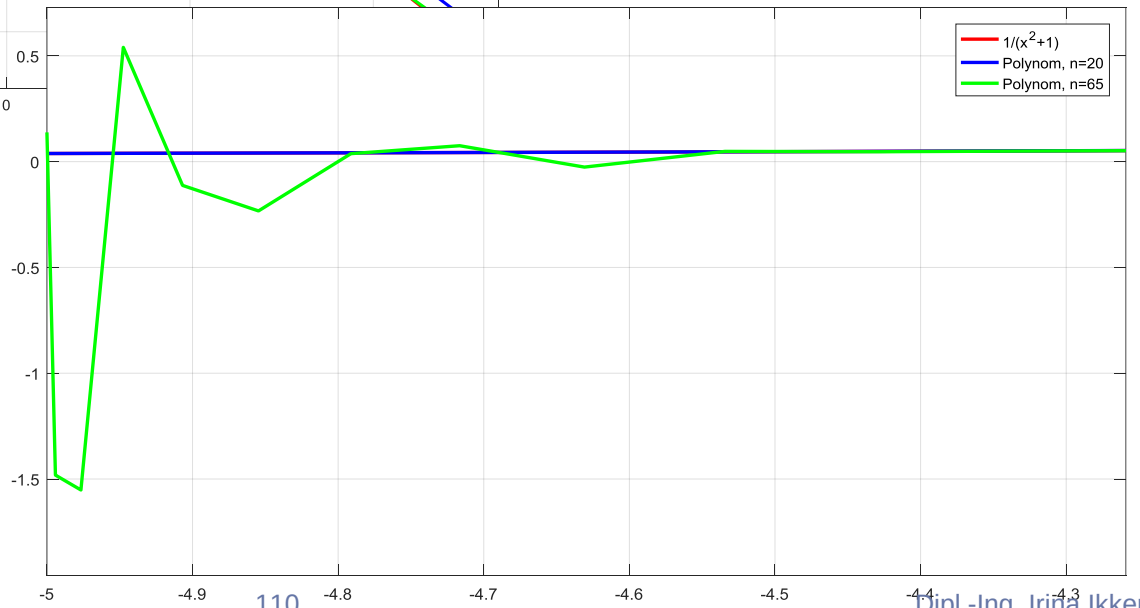
Chebyshev Interpolation

```
% Polynomgrad 65
n = 66;
% Interpolationspunkte
xch2 = (a+b)*0.5-(b-a)*0.5*cos(pi*[0:n]/n);
ych2 = 1./(xch2.^2+1);
% Berechnung des Interpolationspolynoms
polynom2 = polyfit(xch2,ych2,(n-1));
pch2 = polyval(polynom2,xch2);
```



Chebyshev Interpolation





Differential- und Integralrechnung

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

numerische Differentiation	y=diff(x)	Berechnung der Differenz zwischen aufeinanderfolgenden Elementen des Vektors x
	diff(y)./diff(x)	Berechnung der Ableitung dy/dx
symbolische Differentiation syms x y = f(x)	diff(y)	Berechnung der Ableitung nach symb. Variablen
	diff(y,x)	Berechnung der Ableitung nach symb. Variablen x
	diff(y,n)	n-fache Ableitung nach symb. Variablen
	diff(y,x,n)	n-fache Ableitung nach symb. Variablen x

Numerische Differentiation

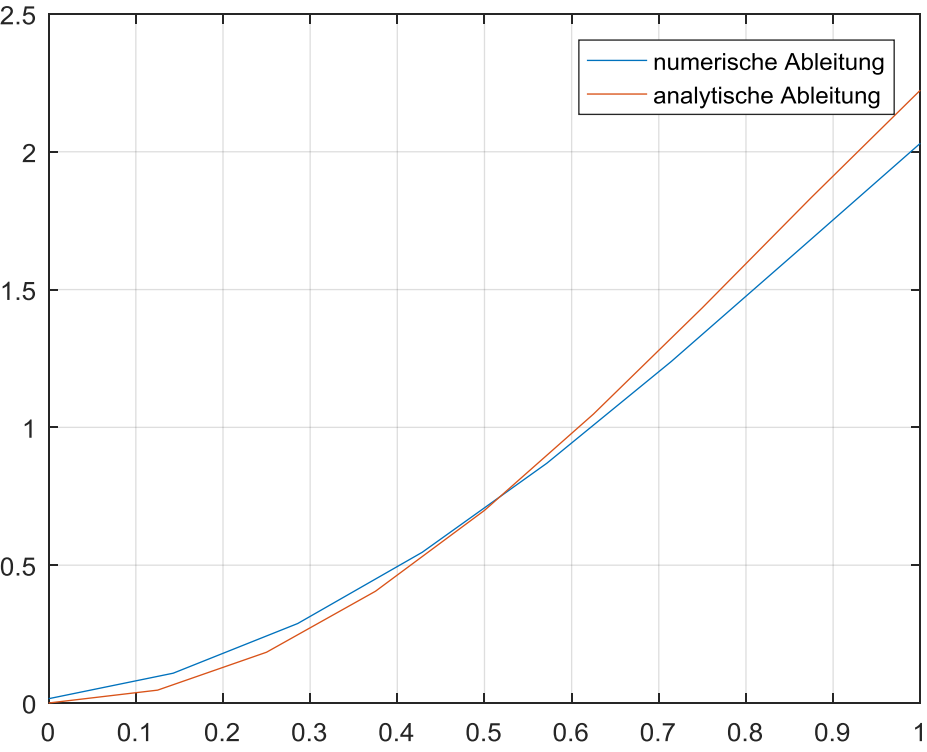
x	0	0,125	0,25	0,375	0,5	0,625	0,75	0,875	1
y,*10^{-3}	0	1,95	15,46	51,51	119,86	228,55	383,42	587,65	841,47

```
x = [0,0.125,0.25,0.375,0.5,0.625,0.75,0.875,1];
y = [0,1.95,15.46,51.51,119.86,228.55,383.42,587.65,841.47].*10^(-3);

abl = diff(y)./diff(x)
```

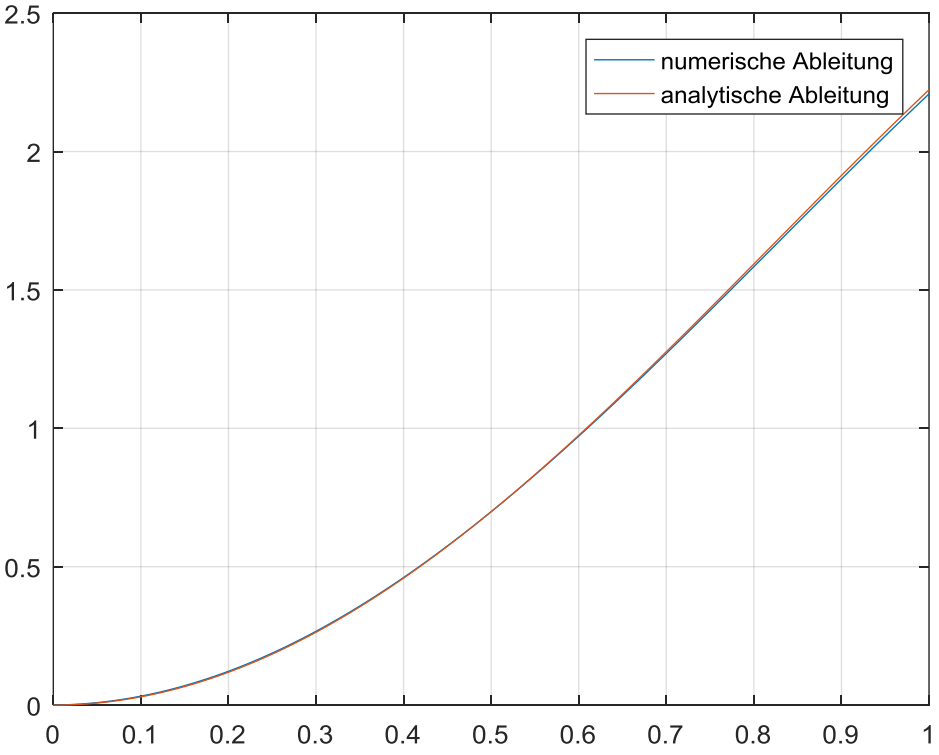
0.016 0.108 0.288 0.547 0.87 1.239 1.634 2.031

Bei der Differenzbildung ändert sich die Dimension des Rückgabevektors!

Numerische Differentiation



N = 10



N = 100

Symbolische Differentiation

- Hohe Genauigkeit, keine Näherung durch numerische Berechnung
- Einfach zu berechnen und auszuwerten
- Schnell zu realisieren

$$y = \ln(x) \cdot \cos(\omega)$$

```
>> syms x w
>> y = log(x)*cos(w)

% Berechnung der Ableitung nach symb. Variablen
>> d = diff(y)
 d = cos(w)/x

% Berechnung der Ableitung nach symb. Variablen ω
>> d = diff(y,w)
 d = -log(x)*sin(w)
```

Symbolische Differentiation

```
% Berechnung der zweifachen Ableitung nach symb. Variablen
>> d = diff(y,2)
 d = -cos(w)/x^2

% Berechnung der zweifachen Ableitung nach symb. Variablen w
>> d = diff(y,w,2)
 d = -log(x)*cos(w)
```

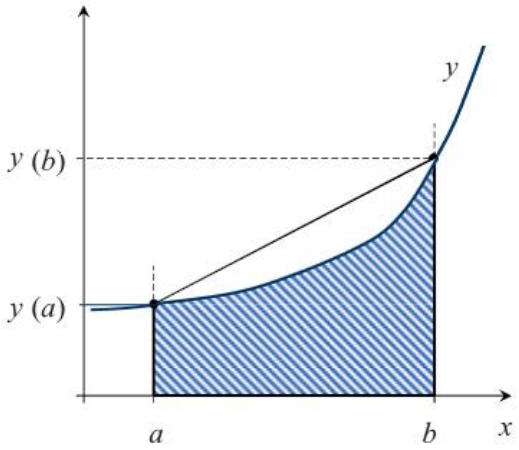
$$I = \int_a^b y(x)dx$$

numerische Integration	trapz(x,y)	Berechnung des Integrals von y über x nach der Trapezregel
	quad(y,a,b)	Berechnung des Integrals von y mit Simpsons - Verfahren
symbolische Integration syms x y = f(x)	int(y)	Berechnung des Integrals von y nach symb. Variablen
	int(y,x)	Berechnung des Integrals von y nach symb. Variablen x
	int(y,a,b)	Berechnung des Integrals von y im Bereich zwischen a und b
	int(y,x,a,b)	Berechnung des Integrals von y nach symb. Variablen x im Bereich zwischen a und b

Numerische Integration
Trapezregel

$$I = \int_a^b y(x)dx = (b-a) \cdot \frac{y(a)+y(b)}{2}$$

$trapz(x,y)$



- Annäherung der Fläche durch ein Trapez
- Integration von **Vektoren** als Argument
- Die **Integrationsgrenzen** sind automatisch durch die Endpunkte der Laufvariablen x gegeben:

$$a = x_{min} \quad b = x_{max}$$

Beispiel 1:

```
>> x = 0:.01:1;
>> I = trapz(x,exp(x))
I = 1.7197
```

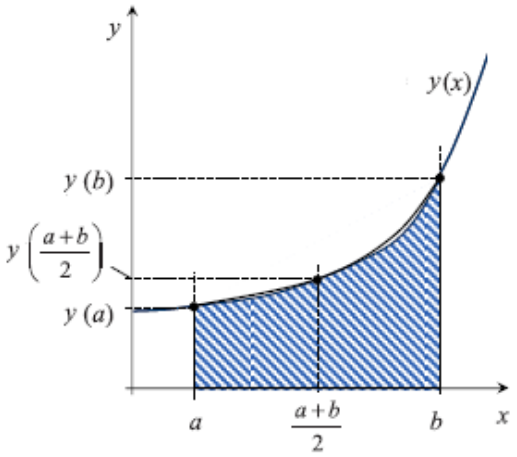
Beispiel 2:

```
>> x = [0,0.125,0.25,0.375,0.5,0.625,0.75,0.875,1];
>> y = [0,1.95,15.46,51.51,119.86,228.55,383.42,587.65,841.47].*10^(-3);
>> I = trapz(x,y)
I = 0.2261
```

Numerische Integration

Simpsons-Verfahren

$$\int_a^b y(x)dx = \frac{(b-a)}{6} \cdot (y(a) + 4 \cdot y((a+b)/2) + y(b))$$



quad(y, a, b)

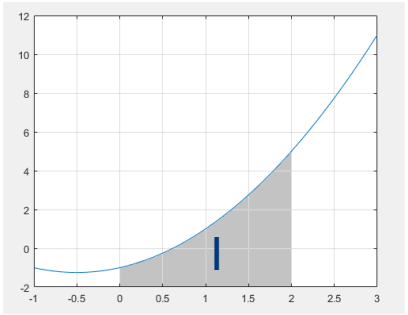
- Annäherung der Fläche durch eine Parabel
- Integration von y als **Function Handle**
- Die **Integrationsgrenzen** werden an die Funktion übergeben
- Sehr genau

Beispiel 1:

>> **I = quad(@exp,0,1)**
I = 1.7183

Beispiel 2:

>> **I = quad(@(x)(x.^2+x-1),0,2)**
I = 2.6667



Symbolische Integration

- Hohe Genauigkeit, keine Näherung durch numerische Berechnung
- Einfach zu berechnen und auszuwerten
- Schnell zu realisieren

$$y = \ln(x) \cdot \cos(\omega)$$

```
>> syms x w
>> y = log(x)*cos(w)

% Berechnung des Integrals von y nach symb. Variablen
>> I = int(y)
 I = -cos(w)*(x-x*log(x))

% Berechnung des Integrals von y nach symb. Variablen ω
>> I = int(y,w)
 I = log(x)*sin(w)
```

Symbolische Integration

% Berechnung des Integrals von y im Bereich zwischen 0 und 2π

>> **I = int(y,0,2*pi)**

I = 2*pi*cos(w)*(log(2*pi) - 1)

% Berechnung des Integrals von y nach symb. Variablen w im Bereich zwischen 0 und $\frac{\pi}{2}$

>> **I = int(y,w,0,pi/2)**

I = log(x)

% Berechnung des Integrals von y nach symb. Variablen w im Bereich zwischen 0 und 2π

>> **I = int(y,w,0,pi*2)**

I = 0

Auswertung symbolischer Berechnungen

- Symbolische Variablen durch Zahlen ersetzen

subs(y,x,zahl) oder **subs(y,{x1, x2,..},{zahl1,zahl2,..})**

y – symbolischer Ausdruck

x – symbolische Variablen

```
>> syms x w                    % Definition symbolischer Variablen
>> y = log(x)*cos(w)           % Symbolischer Ausdruck
>> I = int(y,0,2*pi)           % Berechnung des Integrals
I = 2*pi*cos(w)*(log(2*pi) - 1)
>> I = subs(I,w,pi/3)          % Symbolische Variable ω wird durch die Zahl π/3 ersetzt
I = pi*(log(2*pi) - 1)
```

- Umwandlung des symbolischen Ergebnisses in die numerische Zahl

```
>> double(I)          >> int8(I)
ans = 2.6323          ans = 3
```

Symbolische Berechnung von Gleichungen, Gleichungssystemen und DGLs

Gleichungen

- Symbolische Gleichung y nach einer Variablen lösen

solve(y)

$-2x^2 - 4x + 6 = 0$

```
>> syms x
>> solve(-2*x^2 - 4*x + 6)
ans = -3   1
```

- Symbolische Gleichung mehrerer Variablen lösen

solve(y,x)

$z = x^2 + y^2$

```
>> syms x y
>> solve(z,y)
ans = -x*1i
        x*1i
```

Gleichungen

- Schnittpunkte zweier Gleichungen berechnen

$$x + 2 = x^2 + 2x - 1$$

```
>> syms x
>> solve( x+2 == x^2+2*x-1)

ans =
 - 13^(1/2)/2 - 1/2
   13^(1/2)/2 - 1/2

>> double(solve( x+2 == x^2+2*x-1))

ans =
  -2.3028
   1.3028
```

Gleichungssysteme

Berechnung der Lösung eines Gleichungssystems aus symbolischen Gleichungen:

solve(g1,g2,g3,‘x,y,z‘)

$$\begin{cases} x + y + z = 0 \\ 4x + 5y + z = 3 \\ -2x + y - 3z = 5 \end{cases}$$

```
>> syms x y z
>> g1 = 'x + y + z = 0';
>> g2 = '4*x + 5*y + z = 3';
>> g3 = '-2*x + y - 3*z = 5';
>> [x y z] = solve(g1,g2,g3)
x = -1
y = 3/2
z = -1/2
```

Differentialgleichungen $y' = 2x + y$

- Berechnung der allgemeinen Lösung einer linearen Differentialgleichung:

y = dsolve('Dy = 2*x+y','x')

  - DGL und die unabhängige Variable x werden als String übergeben
  - Dy bezeichnet das Differential
  - Das Ergebnis ist symbolisch mit C als Integrationskonstante

```
>> syms x y
>> y = dsolve('Dy=2*x+y','x')
y = C2*exp(x) – 2*x – 2;
```

- Berechnung der Lösung mit Anfangsbedingung

y = dsolve('Dy = 2*x+y','y(0)=1','x')

```
>> y = dsolve('Dy=2*x+y','y(0)=1','x')
y = 3*exp(x) – 2*x – 2;
```

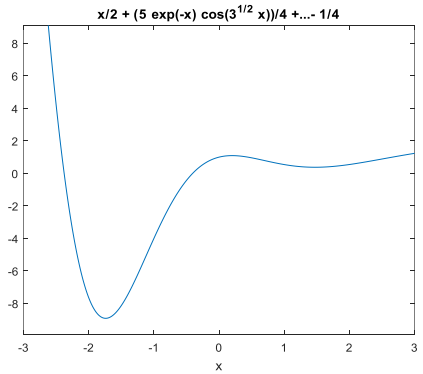
Differentialgleichungen

$$y'' + 2y' + 5y = 2x + y; \quad y(0) = 1, y'(0) = 1$$

- Berechnung der Lösung einer linearen Differentialgleichung höherer Ordnung:

y = dsolve('D2y + 2*Dy + 5*y = 2*x+y','y(0)=1','Dy(0)=1','x‘)

```
>> syms x y
>> y = dsolve('D2y + 2*Dy + 5*y = 2*x+y','y(0)=1','Dy(0)=1','x')
y = x/2 + (5*exp(-x)*cos(3^(1/2)*x))/4 + (7*3^(1/2)*exp(-x)*sin(3^(1/2)*x))/12 - 1/4
```



ezplot(y,[-3,3])