Matlab®-Workshop

- Kurzskript-

- 19. August 2004 -

Dieses Werk erhebt keinen Anspruch auf Vollständigkeit.
Angaben ohne Gewähr

0) Inhaltsverzeichnis

1) Einführung 3
1.1) Die Bedienoberfläche: 3
1.2) Variablen: 3
1.2.1) Variablen erzeugen: 4
1.2.2) Variablen verknüpfen: 4
1.2.3) Spezielle Variablen und Konstanten: Vorsicht: nicht überschreiben 5
1.3) Allgemeine Funktionen: 5
1.4) Matrixmanipulation: 5
1.4.1) Elemente anzeigen: 5
1.4.2) Elemente ersetzen : 5
1.4.3) Elemente hinzufügen: 5
1.4.4) Elemente löschen: 5
1.5) Funktionen: 6
1.6) komplexe Zahlen: 6
1.7) Zeichenketten/ Strings: 6
1.8) Cell Arrays: 7
1.9) m-Files: 8
1.10) Kontrollstrukturen: 9
1.11.1) for-Schleife: 9
1.11.2) if-then-else: 9
1.11.3) while-Schleife: 9
1.11.4) switch-case: 9
1.11.5) try-catch: 9
1.11) Speichern und Laden von Daten 10
1.12) Schreiben eines m-Files: 10
1.13) vordefinierte Fenster: 11
1.14) LTI-Modelle: 12
1.14.1) Zeitbereich: 12
1.14.2) Frequenzbereich: 13
1.15) Tipps und weiterführende Informationen: 13
1.16) Demos: 14
2) Visualisierung von Daten: 15
2.1) 2-dim Grafiken: 15
2.1.1) Funktion plot: zeichnet Datenpunkte, verbindet sie mit Geraden 15
2.1.2) Funktion bar: zeichnet Balkendiagramm 16
2.1.3) Funktion polar: zeichnet Polardiagramm 17
2.1.4) weitere 2D-Zeichenfunktionen: 17
2.2) 3-dim Grafiken: 18
2.2.1) Funktion plot3: zeichnet Datenpunkte, verbindet sie mit Geraden 18
2.2.2) Funktion mesh/ surf: zeichnet Flächen 19
2.2.3) weitere 3D-Zeichenfunktionen: 19
3) Realisierung von GUI-Oberflächen: 20
3.1) grafische Objekte: 20
3.2) Erzeugen grafischer Objekte: 20
3.3) Eigenschaften grafischer Objekte: 21
3.4) Erstellung einer GUI-Oberfläche: 21
3.5) Erstellung einer Animation: 24
4) Anhang: 27
A] Eigenschaften verschiedener grafischer Objekte: >>set(handle) 27
B] Übersicht einiger wichtiger Funktionen 29

1) Einführung

1.1) Die Bedienoberfläche:

Die folgenden Fenster können bei Matlab 6.5 in der Menüleiste unter 'View' bzw. 'Ansicht' aktiviert/ deaktiviert werden

- **Matlab Command Window:**
  - → **Eingabe von Befehlen, Variablen**
  - → **Ausgabe von Daten, Status- und Fehlermeldungen**
- **Launch Pad:**
  - → **Zeigt die geladenen Toolboxen an**
  - → **Demos, Help-Dateien und Programme (Tools) können direkt aus dem Fenster mittels Doppelklick gestartet werden**
- **Workspace (Arbeitsspeicher):**
  - → **zeigt alle im Arbeitsspeicher vorhandenen Variablen mit Namen, Typ, Arraygröße, Speichergröße und seine Klasse an, siehe Beispiele unter 1.2) bis 1.10)**
  - → **Variablen können durch Doppelklick direkt verändert werden**
- **Current Directory (Arbeitsverzeichnis):**
  - → **Wählt das aktuelle Arbeitsverzeichnis aus**
  - → **zeigt alle enthaltenen Dateien samt Modifikationsdatum und bei m-Files ihre Kurzerklärung**
  - → **Dateien können durch Doppelklick direkt geöffnet werden**
- **Command History:**
  - → **Anzeige aller im Command Window eingegebenen Daten mit**
  - → **Datum der jeweiligen Matlab Sitzung als Überschrift**
  - → **mittels Doppelklick auf eine dokumentierte Eingabe wird diese in das Command Window übernommen und ausgeführt**

1.2) Variablen:

- **Typen:**
  - **simple arrays: [ ], besitzen innere Matrixstruktur , können nur Zahlen oder Characters beinhalten.**
    BSP: a = [5 1]; name = ['Joe'];
  - **cell arrays: {}, besitzen äußere Matrixstruktur, können Zahlen und Characters und Strings und Matrizen beinhalten.**
    BSP: person={ 24 'Sneider' ones(1,1)};
  - **structure arrays: punkt-Notation, BSP: kunde.name.vorname = 'Joe'**
  - **LTI-Model: (Linear-Timeinvarant Model), Objekt der Control System Toolbox zur Modellierung von linearen Systemen**

  außerdem: globale und lokale Variablen

  Im Folgenden werden zunächst simple Arrays verwendet

- **Organisation:**

  Simple arrays werden als n x m x p x ... – Matrizen behandelt (n Zeilen, m Spalten, p Ebenen, usw.)

  - **Skalar: 1x1 Matrix, BSP: >>x = 1; oder >>x = [1];**

- **Spalten-Vektor: 3x1 Matrix, >>y = [1;2;3]; entspricht** $y = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$
- **Matrix: z.B. 2x2: >>z = [1 0 ;0 1]; entspricht** $z = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

1.2.1) Variablen erzeugen:

- **durch direkte Zuweisung (=): BSPs**

  >>omega = 50; **>>Gehalt = 2517.234;** **>>A = [1,1];**

  ⇒ <u>Beachte:</u> Punkt für Gleitkommazahlen, Komma als Matrixtrennzeichen

  >>a = [1 2 3]; **>>b = [1;2;3];** **>>E = [1,0,0; 0,1,0; 0,0,1];**
  >>c = 1:0.05:10; **>>d = 20:-2:0;** **>>x = 6+3i;**

- **durch Funktionen: BSPs**

  >>t = linspace(0,0.02,100);
  >>f = zeros(4,4);
  hier stellt erste 4 die Anzahl der Zeilen dar, zweite 4 die Spaltenanzahl
  >>g = ones(4,4);
  >>h = eye(4,4);
  >>k = rand(4,4);

- **durch Anwendung arithmetischer Operationen: BSPs**

  >>erg = omega*Gehalt;
  >>e1 =a*b; (ergibt skalar);
  >>e2 = b*a; (ergibt 3x3 Matrix);
  >>e3 = a.*b'; (ergibt Zeilenvektor);

1.2.2) Variablen verknüpfen:

Addition:	**+**	**Zuweisung:**	**=**
Subtraktion:	**-**	**Vergleich auf Gleichheit:**	**==**
Division:	**/**	**Vergleich auf Ungleichheit:**	**~=**
Multiplikation:	*****		
Kleiner:	**<**	**Und-Verknüpfung:**	**&**
kleiner gleich:	**<=**	**Oder-Verknüpfung:**	**|**
größer:	**>**	**Nicht-Verknüpfung:**	**~**
größer gleich:	**>=**		
Exponenzieren:	**^**	**alle oder bis:**	**:**
Transponieren	**'**	**Operation elementweise**	**.**
Auskommentieren	**%**		

1.2.3) Spezielle Variablen und Konstanten: Vorsicht: nicht überschreiben

- **ans (answer) :**
  Standardausgabevariable, wenn nichts anderes spezifiziert ist. z.B. >>a+b bedeutet ans = a+b
- **eps:**
  kleinster Abstand zwischen zwei Zahlen. eps = 2.2204e-016
- **pi = 3.1416**
- **i = $\sqrt{-1}$, j= $\sqrt{-1}$; imaginäre Einheit**
- **inf (infinity): entspricht ∞, bzw. 1/0**
- **nan (not a number): entspricht 0/0 oder ∞/∞**

1.3) Allgemeine Funktionen:

- **help function: gibt kurze Erklärung zu Funktionen, Beispiele und Verweise auf ähnliche Funktionen**
- **lookfor expression: sucht nach expression in allen m-Files**
- **clear: löscht Variablen**

1.4) Matrixmanipulation:

1.4.1) Elemente anzeigen:

durch Ansprechen ihrer Position innerhalb der Matrix oder Anzeigen des Gesamtinhalts durch Eingabe des Variablennamens (+Enter)

- **>>x oder >>x(1) → x = 3.0000 + 6.0000i**
- **>>a(3) oder >>b(3) → 3**
- **>>E(2,3) → 0**
- **>>t(1:10) → 0 0.0002 0.0004 0.0006 .... 0.0018**
- **>>h(2,:) → 0 1 0 0**

1.4.2) Elemente ersetzen :

Beachte, dass eine Matrix immer eine rechteckige Struktur haben muss

- **>>g(2,3) = 4;**
- **>>b(2) = 2^2;**
- **>>f(3,2:4) = [2 2 2];**

1.4.3) Elemente hinzufügen:

Matlab erweitert Matrizen automatisch und füllt sie mit Nullen auf

- **>>a(6) = 6; → 1 2 3 0 0 6;**
- **>>g(4,6) = 1;**
- **>>b(:,2) = [4;5;6];**

1.4.4) Elemente löschen:

- **>>a = [a(1:2) a(6)];**

- **>>d(3:6)=[ ];**
- **>>k(:,2:3)=[ ];**

1.5) Funktionen:

- **allgemeine Form: y = f(x)**
  → y ist Ausgabeparameter, x ist Eingabeparameter,
  → f spezifiziert die Funktion
  → Aufruf durch "Call by Value"
  → anwendbar auf Skalare und Matrizen
  → i. A. beliebig viele Ein- und Ausgabeparameter möglich, aber ihre Reihenfolge ist relevant

- **BSPs: >> y1 =sqrt(256); >> y2 = exp(k);**
  >> [n,m] = size(k);
  >> s = logspace(1,100,50);

  → Optionen der Parameter nachschlagen mit >>help function

- **Übersicht einiger Funktionen im Anhang B**

1.6) komplexe Zahlen:

- **kein Unterschied zu normalen Zahlen, fast alle Funktionen verarbeiten auch komplexe Zahlen. Vorsicht nur bei Zeichenfunktionen (plot etc.)**
- **Matlab stellt kompl. Zahlen kartesisch dar:**
  $z = x + iy$, $x = Re\{z\}$, $y = Im\{z\}$
- **Imaginäre Einheit ist i oder j**
- **Rechenregelen beachten: $c6^T$ entspricht transponieren + kompl. konj.**
- **wichtigste Funktionen: → real(), imag()**
  → abs(), angle()
- **BSPs: >> c1 = 1-2i; >>c2 = 3*c1; >>c3 = [(c1 + c2)/c2; c1];**
  >> c4 = exp(j*pi/4); >>c5 = sqrt(-16);
  >> c6 = sin(c3); >> c6'

1.7) Zeichenketten/ Strings:

- **werden durch Apostroph (') eingeleitet und abgeschlossen**
- **Beispiele: >>name = 'Schneider' → 1x9 Matrix aus Characters**
  Beachte unterschied zu Matrizen mit Zahlen:
  >>string_matrix1 = ['Hi','Du'] → 1x4 Matrix aus Characters
  >>string_matrix1(4)
  >>string_matrix1 = ['Hi';'Du'] → 2x2 Matrix aus Characters
  >>string_matrix1(2,2)

- **wichtige Funktionen mit Strings:**
  - **Konvertierung : str2num, num2str, str2double, double2str; BSP**
    >>eingabe = '1';
    >>ausgabe = eingabe*2; → kommt irgendwas raus
    >>eingabe = str2num(eingabe);
    >>ausgabe = eingabe*2; → ergibt 2

- String in Zeichenkette suchen: >>findstr(name,'e')
  gibt Indizes zurück, bei denen der String 'e' anfängt (hier 4)
- Strings zusammenfügen mit string concatenate:
  >>strcat('string1','string2')
  gibt zusammengefügten String zurück
- Strings vergleichen mit string compare:
  >>strcmp('string1','string2')
  gibt 1 für erfüllt oder 0 für nicht erfüllt zurück

1.8) Cell Arrays:

- besitzen äußerlich gleiche Struktur wie simple Arrays; können jedoch verschiedene Arten von Daten enthalten. Man betrachtet die Elemente als 'Container'
- Beispiel:

$$A = \begin{Bmatrix} 15 & [1,2,3,4,5] \\ 'Schneider\ ' & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{Bmatrix}$$

- Erzeugen von Cells:

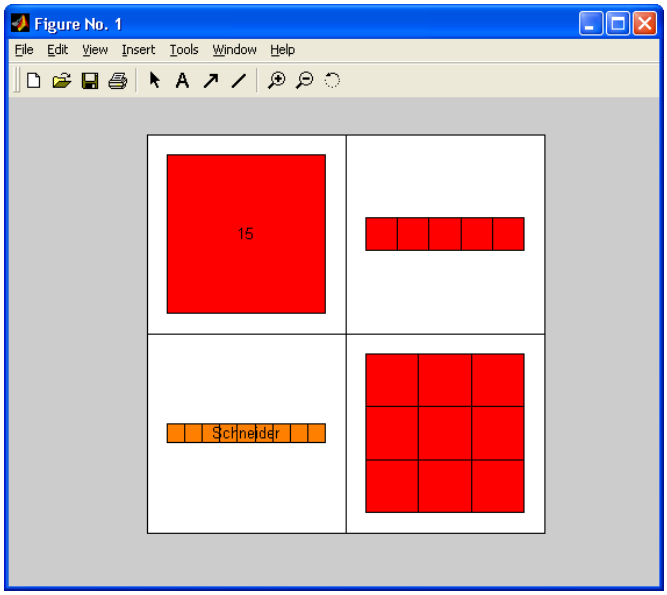
```
>>A(1,1) = {15};
>>A(1,2) = {[1,2,3,4,5]};
>>A(2,1) = {'Schneider'};
>>A(2,2) = {[1 0 0;0 1 0;0 0 1]};
```

auch durch A{m,n} = ... möglich

- Inhalte von Cells anzeigen:
  eine einfache Eingabe von A reicht hier nicht. Es wird nur die innere Struktur angezeigt, aber kein Inhalt der Container:

```
    [      15]      [1x5 double]
'Schneider'     [3x3 double]
```

  - Anzeigen des Inhalts eines Containers: >>A{2,2}
  - Anzeigen eines ganzen Cells: >>celldisp(A)
  - grafische Darstellung der inneren Struktur eines Cells: >>cellplot(A)



1.9) m-Files:

- bilden die Funktionsbibliothek von Matlab
- jede Funktion ist in einem m-File implementiert
- ASCII-Dateien mit Endung .m
- zum Schreiben/ Editieren von m-Files wird der Matlab Editor benutzt >>edit
- Beispiel: cross.m → Berechnung des Kreuzprodukts : >>edit cross.m

```
function c = cross(a,b,dim)
%CROSS  Vector cross product.
%   C = CROSS(A,B) returns the cross product of the vectors
%   A and B.  That is, C = A x B.  A and B must be 3 element
%   vectors.
%
%   C = CROSS(A,B) returns the cross product of A and B along the
%   first dimension of length 3.
%
%   C = CROSS(A,B,DIM), where A and B are N-D arrays, returns the cross
%   product of vectors in the dimension DIM of A and B. A and B must
%   have the same size, and both SIZE(A,DIM) and SIZE(B,DIM) must be 3.
%
%   See also DOT.

%   Clay M. Thompson
%   updated 12-21-94, Denise Chen
%   Copyright 1984-2001 The MathWorks, Inc.
%   $Revision: 5.17 $  $Date: 2001/04/15 12:01:46 $

% Special case: A and B are vectors
rowvec = 0;
if ndims(a)==2 & ndims(b)==2 & nargin<3
   if size(a,1)==1, a = a(:); rowvec = 1; end
   if size(b,1)==1, b = b(:); rowvec = 1; end
end;

% Check dimensions
if ~isequal(size(a),size(b)),
  error('A and B must be same size.');
end

if nargin == 2
  dim = min(find(size(a)==3));
  if isempty(dim),
    error('A and B must have at least one dimension of length 3.');
  end
end;

% Check dimensions
if (size(a,dim)~=3) | (size(b,dim)~=3),
 error('A and B must be of length 3 in the dimension in which the cross product is taken.')
end

% Permute so that DIM becomes the row dimension
perm = [dim:max(length(size(a)),dim) 1:dim-1];
a = permute(a,perm);
b = permute(b,perm);

% Calculate cross product
c = [a(2,:).*b(3,:)-a(3,:).*b(2,:)
    a(3,:).*b(1,:)-a(1,:).*b(3,:)
    a(1,:).*b(2,:)-a(2,:).*b(1,:)];
c = reshape(c,size(a));

% Post-process.
c = ipermute(c,perm);
if rowvec, c = c.'; end
```

1.10) Kontrollstrukturen:

1.11.1) for-Schleife:

Syntax

```
for n = anfang:schrittweite:ende
  Anweisungen;
end
```

Beispiel:

```
for n = 1:5
  for m = 5:-1:1
    A(n,m) = n^2 + m^2;
  end
end
```

1.11.2) if-then-else:

Syntax

```
if Bedingung
  Anweisungen;
elseif
  Anweisungen;
else
  Anweisungen;
end
```

Beispiel:

```
for n = 1:10
  if (n>6 & n<8)
     n
  elseif (n>=8)
     disp('Wert überschritten')
  else
     disp('Wert noch nicht erreicht')
  end
end
```

1.11.3) while-Schleife:

Syntax

```
while Bedingung
  Anweisungen
end
```

Beispiel:

```
num = 0; % Zählvariable
eps = 1;  % neue Definition
while (eps+1) > 1
  eps = eps/2;
  num = num+1;
end
num
eps
```

1.11.4) switch-case:

Syntax

```
switch Ausdruck
  case Ausdruck1
    Anweisungen; break;
  case Ausdruck2
    Anweisungen; break;
     ...
  otherwise
    Anweisungen; break;
end
```

Beispiel:

```
switch Tag
  case 'Montag'
    disp('erster Tag in der Woche');
    break;
    ...
  case 'Freitag'
    disp('letzter Tag in der Woche');
    break;
  otherwise
    disp('Wochenende');
end
```

1.11.5) try-catch:

Syntax

```
try
   Anweisungen;
catch
   Anweisungen;
end
```

Beispiel:

```
try
   y = cus(pi); % cus gibt's nicht
catch
   disp('Fehler aufgetreten')
end
lasterr
```

1.11) Speichern und Laden von Daten

- **speichern: >>save('Name.mat','variable1','variable2',...);**

  BSP: >>save('test.mat','A','kunde','Gehalt');

  - auch >>save 'Name.mat' 'variable1' möglich, wie oben aber einheitlicher!
  - werden Variablen nicht spezifiziert, werden alle Variablen gespeichert
  - mat-dateien haben Binärformat
  - speichern als ASCII z.B. durch >>save 'Name.txt' 'variable1' -ASCII

- **laden: >>load('Name.mat','variable1', ...);**

  BSP: >>load('test.mat');

  - wird .mat nicht angegeben, wird die Datei automatisch als mat-Datei behandelt
  - auch >>load 'Name.mat' möglich (oder ohne ''), wie oben aber einheitlicher!
  - werden Variablen nicht spezifiziert, werden alle Variablen geladen

- **siehe auch Funktionsfamilie fopen, fwrite, fread, fclose, fprintf;**

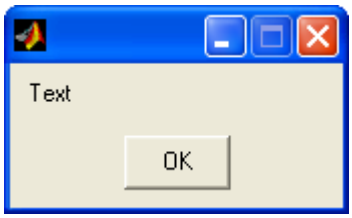
1.12) Schreiben eines m-Files:

- **Implementieren Sie die Funktion** $si(x) = \frac{\sin(x)}{x}$
- **Es sollen Matrizen übergeben und das Ergebnis zurückgegeben werden**
- **Beachten Sie die Definitionslücke**

- **Betrachten Sie die Funktion sinc.m**

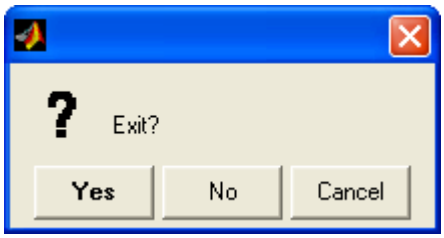
1.13) vordefinierte Fenster:

- msgbox('Text'); → erzeugt ein Fenster mit dem spezifizierten Text



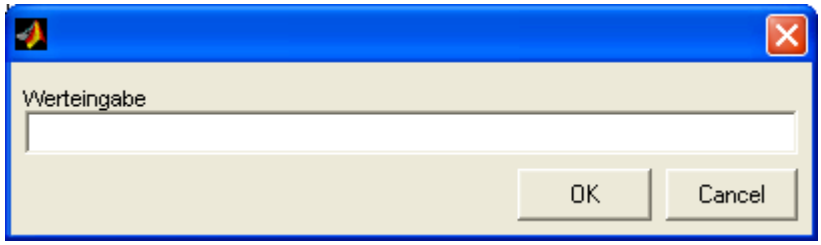
- errordlg('Text'); → wie oben mit Hand-Bild und OK-Knopf

- warndlg('Text'); → wie oben mit Warndreieck und OK-Knopf
- helpdlg('Text'); → wie oben mit Kopf+Sprechblase und OK-Knopf
- out = questdlg('Exit?'); →Abfragefenster mit Knöpfen Yes/No/Cancel;



Knopfbeschriftung wird als String in out zurückgegeben

- in = inputdlg('Werteingabe'); → Fenster mit Textfeld. Eingegebenes wird als String in den Cell-Array in geschrieben. Bei Zahlen muss umgewandelt werden: in_zahl = str2num(in{1});



Wird ein Fenster geöffnet, wartet Matlab auf eine Eingabe. Andere Fenster sind inaktiv!

1.14) LTI-Modelle:

LTI = Linear-Time-Invariant Model
Matlab stellt eine Klasse von Objekten zur Verfügung mit denen lineare zeitinvariante Systeme definiert und analysiert werden können. Für die Beschreibung eines Systems gibt es verschiedenen Modelle. Davon sollen im Folgenden zwei behandelt werden:

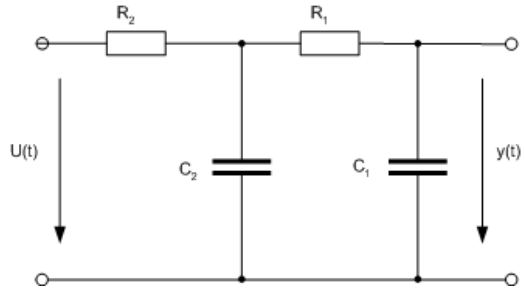
1.14.1) Zeitbereich:

Das Zustandsraummodell (ZRM) (State-Space Form):

$$\dot{\underline{x}} = \underline{\underline{A}}\underline{x}(t) + \underline{B}u(t) \text{ und } y = \underline{C}\underline{x}(t) + Du(t) \text{ mit}$$

$\underline{x}$(t) : Zustandsvektor
$\underline{\underline{A}}$: Systemmatrix
$\underline{B}$: Steuervektor
$\underline{C}$: Beobachtungsvektor
D : Durchgriff
u(t): Eingangsgröße
y(t): Ausgangsgröße

Beispiel zum ZRM:



Zustandsgrößen: $x(t) = (u_{c1}(t)\ u_{c2}(t))^T$
Anfangsbed.: $u_{c1}(0) = u_{c2}(0) = 0$, d.h.

$$x0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$y = (1 \quad 0)\begin{pmatrix} u_{c1} \\ u_{c2} \end{pmatrix} + 0 \cdot u$$

$$\begin{pmatrix} \dot{u}_{c1} \\ \dot{u}_{c2} \end{pmatrix} = \begin{pmatrix} -\frac{1}{R_1 C_1} & \frac{1}{R_1 C_1} \\ -\frac{1}{R_1 C_2} & -\frac{R_1 + R_2}{R_1 R_2 C_2} \end{pmatrix}\begin{pmatrix} u_{c1} \\ u_{c2} \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{R_2 C_2} \end{pmatrix} u$$

Sei $R_1 = R_2 = 1k\Omega$ und $C_1 = 10\mu F$, $C_2 = 1nF$, dann folgt

$$\underline{\underline{A}} = \begin{pmatrix} -10^2 & 10^2 \\ 10^6 & -2 \cdot 10^6 \end{pmatrix},\ \underline{B} = \begin{pmatrix} 0 \\ 10^6 \end{pmatrix},\ \underline{C} = (1 \quad 0),\ D = 0$$

Erzeugen eines LTI-Objekts system: **>> system = ss(A, B, C, D);**
Anzeigen der Systemparameter: **>>[A, B, C, D] = ssdata(system)**

Auf LTI-Objekte können nur bestimmte Funktionen angewandt werden.
Darstellen der Impulsantwort (Gewichtsfunktion): **>>impulse(system);**
Darstellen der Sprungantwort (Übergangsfunktion): **>>step(system);**
Darstellen der Eigenbewegung: **>>initial(system,x0);**
--> Eine Eigenbewegung ist hier nicht vorhanden, weil als Anfangsbedingung $u_{c1}(0) = u_{c2}(0) = 0$ gewählt wurde, also bei t=0 keine Energie im System steckt.

1.14.2) Frequenzbereich:

Die Übertragungsfunktion (Transfer-Function) mit s = σ + jω:

$$H(s) = \frac{Z(s)}{N(s)} = \frac{a_1 s^m + a_2 s^{m-1} + ... + a_m s + a_{m+1}}{b_1 s^n + b_2 s^{n-1} + ... + b_n s + b_{n+1}}$$

Beispiel siehe oben: Das System beschreibt die Reihenschaltung zweier Tiefpässe der Grenzfrequenzen $\omega_1 = \frac{1}{R_1 C_1}$ **und** $\omega_2 = \frac{1}{R_2 C_2}$**, wobei jedoch der erste TP durch den Zweiten belastet ist.**
Die Übertragungsfunktion lautet (σ = 0):

$$H(j\omega) = \frac{1}{1 + j\omega R_1 C_1 + j\omega R_2 C_1 - \omega^2 R_1 C_1 R_2 C_2} \Rightarrow H(s) = \frac{10^8}{s^2 + s(10^6 + 10^6) + 10^8}$$

Zur Vereinfachung gilt $s = j\omega$ **. Erzeugen einer Übertragungsfunktion durch Definition von Zähler und Nenner und Aufruf der Funktion 'transfer-function'**
Dabei sollte Ausdruck so normiert sein, dass $b_1 = 1$ ist.

```
>>z = [0,0,1e008];  % z = [a1 a2 a3  ...]
>>n = [1, 2e006, 1e008]; % n = [b1 b2 b3  ...]
>>system2 = tf(z,n)
```

Anzeigen von Zähler und Nenner:	**>>[z n] = tfdata(system2);**
	>>z{:},n{:}
Anzeigen des Amplituden- und Phasengangs:	**>>bode(system2)**
"" mit Verstärkung und Phasenreserve:	**>>margin(system2)**
Anzeigen der Ortskurve:	**>>nyquist(system2)**

LTI-Modelle, die in der einen Darstellung modelliert wurden, können in andere Darstellungen überführt werden:

```
>>sys1 = ss(system2)
>>sys2 = tf(system1)
```

Beachte: Aufgrund von numerischen Genauigkeiten und unterschiedlichen Möglichkeiten ZRMs aufzustellen, ist Identität von Systemen nicht direkt sichtbar: sys1 gleicht nicht system1 aber man erkennt sys2 = system2.

1.15) Tipps und weiterführende Informationen:

- **Mit den Tasten ↑ und ↓ können die zuvor eingetippten Befehle wieder angesprochen werden**
- **Terminieren einer Endlosrechnung mit Strg+c möglich**
- **Nutzen sie die Vorteile der Matrixnotation:**

  BSP: $y = \sum_{n=1}^{m} x_n^2$ **ist mit einer for-Schleife unsinnig. Besser** $y = x \cdot x^T$

```
x = 5;                  besser: x = max(5,y);
if  y > 5
  x = y;
```

end

- **am einfachsten lernt man mit Beispielen: → m-Files und → demos anschauen!!!**
- **Literatur:**

  - **Adrian Biran: Matlab 5 für Ingenieure, Addison-Wesley, 1999, ISBN 3-8273-1416-X**
  - **Frieder, Grupp: Matlab 6 für Ingenieure: Oldenbourg, 2002, ISBN 3-486-25957-1**
  - **Christoph Ueberhuber: Matlab 6 – eine Einführung, Springer, 2002, ISBN 3-211-83487-7**
  - **Patrick Marchand: Graphics and GUIs with Matlab, CRC-Press, 2003, ISBN 1-58488-320-0**
  - **Kammeyer, Kroschel: Digitale Signalverarbeitung mit Matlab-Übungen, Teubner, 2002, ISBN 3-519-46122-6**
  - **Jan Lunze: Regelungstechnik I und II, Springer, 2002, ISBN 3-540-43116-0 und 3-540-42178-5**

1.16) Demos:

eine kurze Auswahl von sehenswerten Programmen, Animationen und Slideshows

>> demo

+ Matlab: Matlab erklärt sich selbst

+ Desktop Environment:

- Desktop Overview: Animation, die die Bedienoberfläche von Matlab erklärt; verwendet Browser

+ Matrices

- Basic Matrix Operations: Slideshow, die die Grundlagen der Matrixerstellung erklärt

+ Gallery: zeigt Grafiken

- Knot: zeigt einen Knoten

- Quiver: zeigt Höhenunterschiede mittels Pfeilen

- Modes: Eigenschwingungen einer eingespannten Membran

+ more Examples:

- World Traveller: Programm, zeichnet Flugrouten

- Bending Truss: simuliert Eigenschwingungen einer Brücke

- Spinning Cruller Movie: Animation, drehendes Möbius-Band

+ Toolboxes

+ Control System: (Regelungstechnik)

- Heat Exchanger Control: Programm, Modell einer Heizung

+ Signal Processing: (Signalverarbeitung)

- Discrete Fourier Transformation: Interaktive Darstellung des Zusammenhangs von Zeit- und Frequenzfunktion

2) Visualisierung von Daten:

2.1) 2-dim Grafiken:

2.1.1) Funktion plot: zeichnet Datenpunkte, verbindet sie mit Geraden

Beispiel: Darstellung einer 50 Hz Schwingung mit 2 kHz Abtastfrequenz
Schreiben Sie die folgenden Befehle in ein m-File mit Namen plot_test.m

```
f = 50;
t = linspace(0,1,2000);

y1 = 3*sin(2*pi*f*t);
y2 = 2*sin(2*pi*f*t) - cos(2*pi*f*t);

figure
plot(y1);

figure
plot(t,y1);

figure
plot(t,y1,'c+:');

figure
plot(t,y1,'go:',t,y2,'r^-.');

figure
subplot(2,2,1);
plot(y1)
subplot(2,2,2);
plot(t,y1);
subplot(2,2,3);
plot(t,y1,'c+:');
subplot(2,2,4);
plot(t,y1,'go:',t,y2,'r^-.');
```

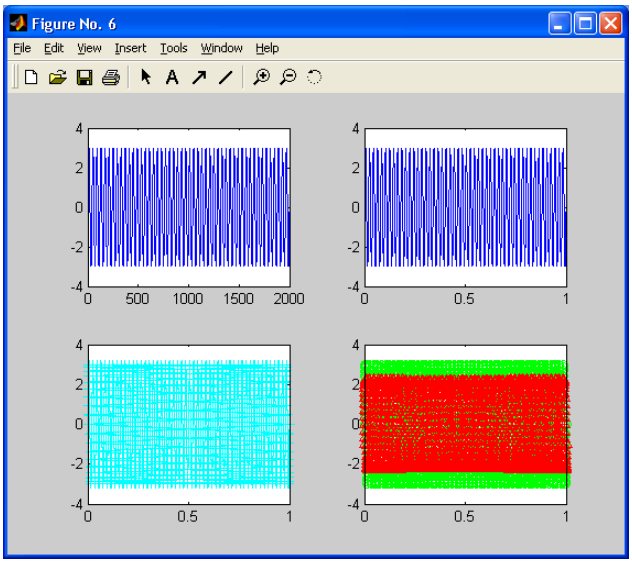
- verfügbare Farben und Marker:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star		
y	yellow	s	square		
k	black	d	diamond		
		v	triangle		
		^	triangle		
		<	triangle		
		>	triangle		
		p	pentagram		
		h	hexagram		

- subplot:

- Aufruf mit Angabe der Matrixstruktur (Aufteielung) des Figures
- Durchnummerierung der Plots der Reihe nach
- nicht nur in Zusammenhang mit plot verwendbar

mit subplot erzeugtes Figure:



- hilfreiche Beschriftungen:
  - grid; %Raster
  - title('Name');
  - xlabel('x');
  - ylabel('y');
  - legend('y1','y2'); % Legende

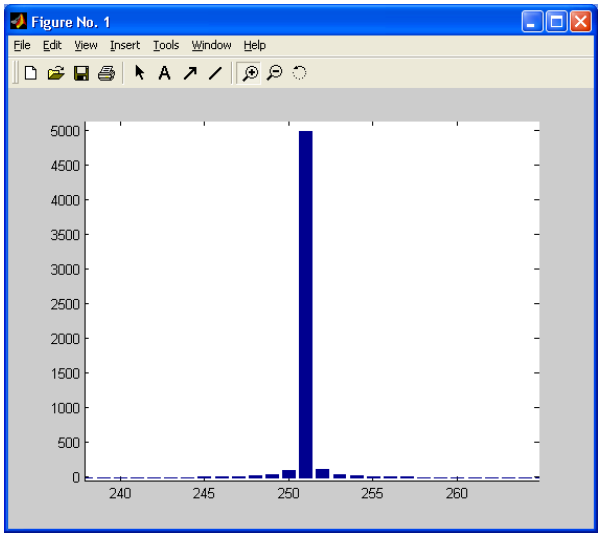
2.1.2) Funktion bar: zeichnet Balkendiagramm

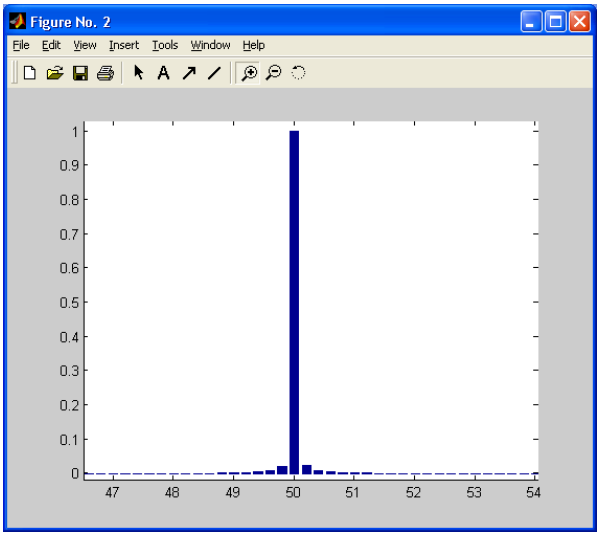
Beispiel: Fouriertransformation einer 50 Hz Schwingung mit 2 kHz Abtastfrequenz und 5s Dauer

- **zunächst: Theorie; was erwarten wir ? (siehe Erläuterungen)**
- **Y = fft(y): realisiert die Diskrete Fouriertransformation. Sie gibt einen (hier) Vektor zurück, dessen Einträge (auch bins = Container genannt) Informationen über die Amplitude der in dem Datensatz y enthaltenen Frequenzen gibt. Sie gibt also ein Frequenzspektrum Y zurück. (Es ist bekanntlich spiegelsymmetrisch zur y-Achse)**
  - **→ Y(1) beschreibt den Gleichanteil (0-tes bin)**
  - **→ Beachte: Matlab beginnt mir der Indizierung bei 1**
  - **→ FFT gibt nur positive Frequenzen**
  - **→ n-ter Index entspricht der Frequenz** $f = \frac{n-1}{D}$
  - **→ n-tes bin entspricht der Frequenz** $f = \frac{n}{D}$

Schreiben Sie die folgenden Befehle in ein m-File mit Namen bar_test.m

```
f = 50;
D = 5;
f_abt = 2000;
t = linspace(0,D,D*f_abt);

y = cos(2*pi*f*t);
Y = fft(y);

figure
bar(abs(Y));

Y = Y/max(Y); % Normieren, Aussage über Frequenzanteile in Prozent
Y = Y(1:D*f_abt/2); % überflüssiges Spektrum wegschmeißen
n = (0:length(Y)-1)/D; % Frequenzen auf x-Achse anzeigen; bin in Frequenzen umrechnen

figure
bar(n,abs(Y));
```





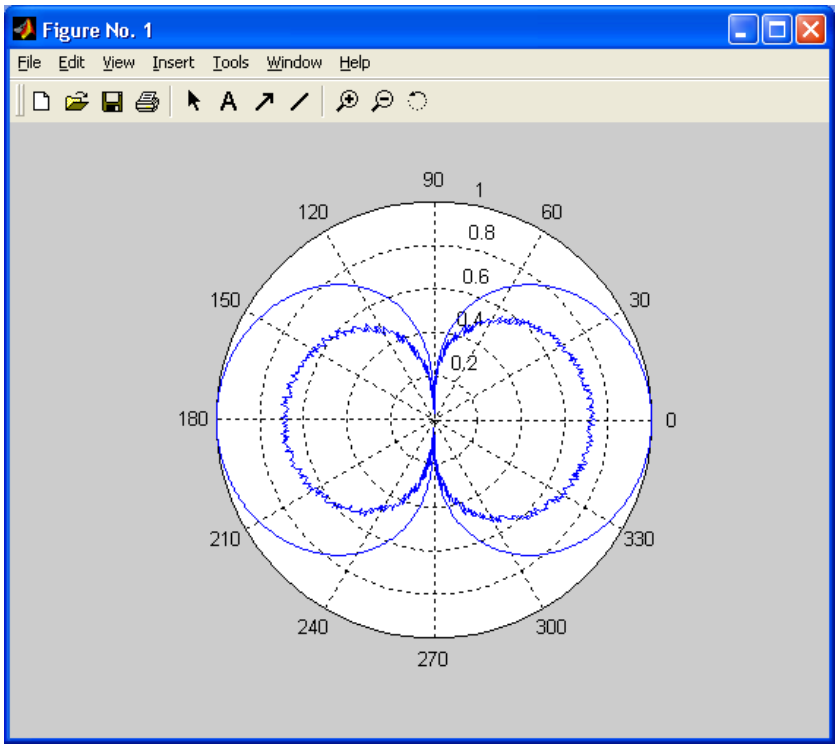
→ Beachte, das der gleiche Balkenverlauf auch bei D*f_abt – D*f vorhanden ist.

- Ein zeitlich endliches Signal verursacht eine Diskretisierung des Frequenzspektrums (für D → ∞ wird Spektrum kontinuierlich)
- Ein zeit-diskretes Signal verursacht eine periodische Fortsetzung des Frequenzspektrums mit Periode D*f_abt

2.1.3) Funktion polar: zeichnet Polardiagramm

Beispiel: Richtcharakteristik von Antennen oder Mikrofonen
Schreiben Sie die folgenden Befehle in ein m-File mit Namen polar_test.m

```
theta = 0:0.01:2*pi;
rauschen = 0.1*rand(1,length(theta));

r1 = sqrt(abs(cos(theta)));
r2 = sqrt(0.5*abs(cos(theta)+ rauschen));

figure
polar(theta,r1);
hold on
polar(theta,r2);
hold off
```



2.1.4) weitere 2D-Zeichenfunktionen:

- stair(x,y) : Interpoliert mit Sprüngen zwischen Abtastwerten
- errorbar(x,y,e): zeichnet x über y samt Fehlerbalken für jeden Wert
- stem(x,y): zeichnet "peaks" mit Kreis am Ende
- line(x,y): wie plot
- compass(x,y): zeichnet Polardiagramm mit Pfeilen vom Ursprung

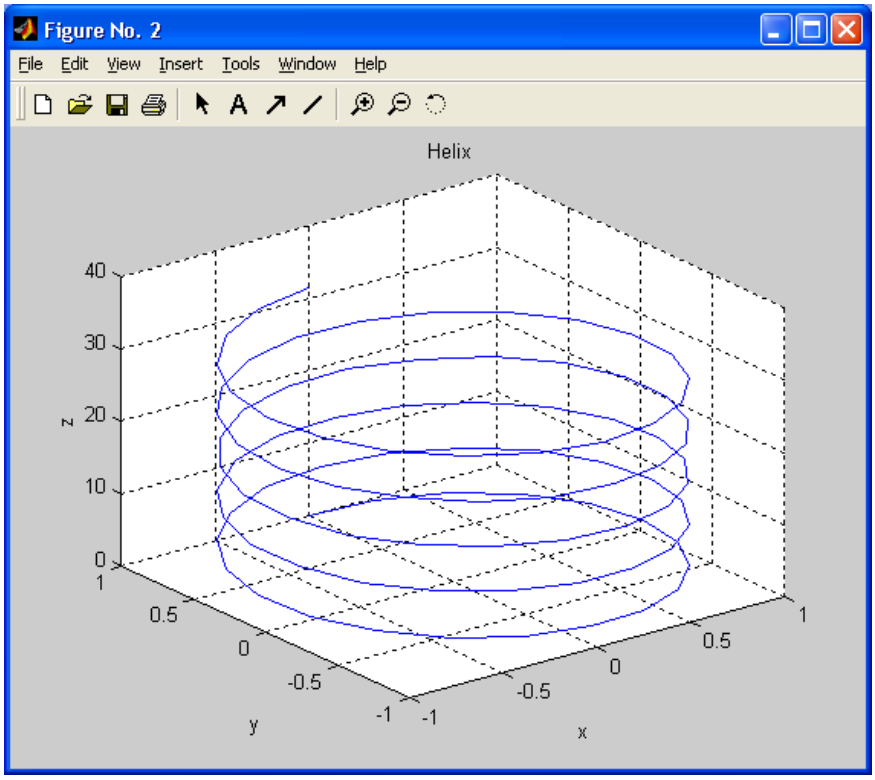
2.2) 3-dim Grafiken:

2.2.1) Funktion plot3: zeichnet Datenpunkte, verbindet sie mit Geraden

Beispiel: Darstellung einer Schraubenbewegung
Schreiben Sie die folgenden Befehle in ein m-File mit Namen helix.m

- **plot3 ist die Erweiterung der Funktion plot auf 3 Zeichendimensionen**
- **sämtliche von plot bekannten Optionen sind verfügbar**

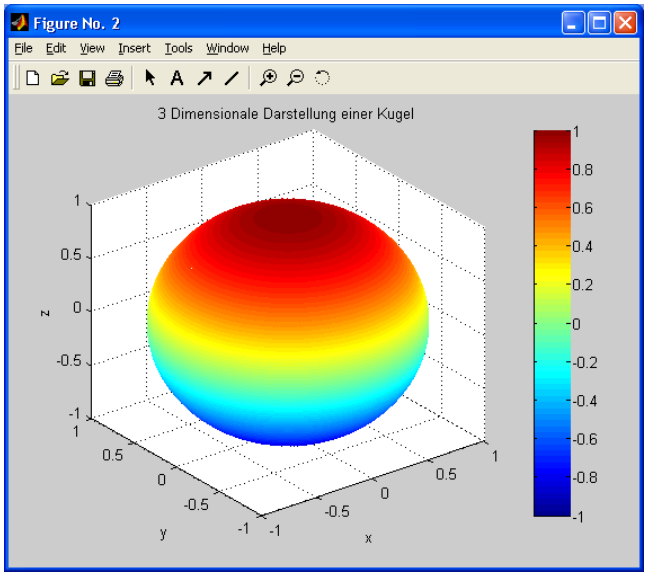
```
t = linspace(0,10*pi,1000);
x = sin(t);
y = cos(t);
z = t;
grid;
figure
plot3(x,y,z);
title('Helix');
xlabel('x');
ylabel('y');
zlabel('z');
```



2.2.2) Funktion mesh/ surf: zeichnet Flächen

Beispiel: Darstellung einer Kugel
Schreiben Sie die folgenden Befehle in ein m-File mit Namen kugel.m

```
phi = linspace(0,2*pi,500);
theta = linspace(0,pi,500);
r = 1;
[PHI,THETA]=meshgrid(phi,theta);
X = r*sin(THETA).*cos(PHI);
Y = r*sin(THETA).*sin(PHI);
Z = r*cos(THETA);

figure
mesh(X,Y,Z)
xlabel('x');
ylabel('y');
zlabel('z');
title('3 Dimensionale Darstellung einer Kugel')

colorbar
```



- [X,Y] = meshgrid(x,y): erzeugt ein Raster
- alternativ zu mesh kann auch die Funktion surf(X,Y,Z) benutzt werden, die eine richtige Fläche zeichnet, d.h. es wird zwischen dem Raster interpoliert. Die Zeichnung ist ein anderer Objekttyp, auf den eine Oberflächeninterpolation angewendet werden kann. Ersetzen sie mesh durch: surf(X,Y,Z); shading interp;
- Benutzen Sie die Rotate- Funktion in der Toolbarleiste des Figures zum Drehen der Zeichnung
- colorbar zeigt die Zuordnung der Farbwahl zu den Höhen

2.2.3) weitere 3D-Zeichenfunktionen:

- slice: darstellen von Skalarfeldern, z.B. Temperatur im Raum
- contour(x,y,z,n): erzeugt n Höhenlinien (s. auch contour3(x,y,z,n))
- waterfall(X,Y,Z): Achtung: erfordert viel Rechenleistung

3) Realisierung von GUI-Oberflächen:

Es gibt zwei Möglichkeiten GUIs zu erzeugen:
- **mit einem Tool: >>guide**
- **oder durch handarbeit**

In Matlab 5 waren die Tools hilfreich, um schnell Oberflächen zu erzeugen. Der Quellcode war zwar voll von Überflüssigem, aber ähnelte noch sehr der handschriftlichen Programmierung. Will man jedoch, dass die einzelnen Elemente auch ein Bißchen miteinander interagieren, hat man schnell den generierten Quellcode fast ganz umgeändert.

In Matlab 6 ist das Tool komplett anders und der Quellcode der für das Layout der Oberfläche erzeugt wird, ist dem Programmierer nicht direkt zugänglich. Jedenfalls habe ich es noch nicht geschafft, damit vernünftig zu arbeiten.

Die beste Methode, den Umgang und vor Allem die Möglichkeiten und Fähigkeiten der einzelnen Objekte zu verstehen, ist letztendlich selbst Programmieren

- **Jedes grafische Objekt ist durch Identifikationsnummer, in Matlab "handle" genannt, ansprech- und manipulierbar.**
- **Figures haben eine ganzzahlige Handlingnummer**
- **Sie wird beim Aufruf der (Erzeugungs-)Funktion zurückgegeben, wenn ein Ausgabeparameter erwünscht wird. z.B. >>fig_number = figure**
- **Jedes GUI-Objekt besitzt Eigenschaften. Siehe Anhang B**

3.1) grafische Objekte:

- **figure → erzeugt Fenster**
- **plot, line, bar, etc. → erzeugen Graphen**
- **text, title, xlabel, etc. → erzeugen Text**
- **axes → erzeugt Koordinatensystem /Achsen**
- **uicontrol → erzeugt Interaktionselemente:**
  - **Textfeld**
  - **Eingabefeld**
  - **Knopf (voreingestellt)**
  - **Kontrollkästchen**
  - **Radiobutton**
  - **Rahmen**
- **uimenu(parent) → erzeugt Menüpunkt in Menüzeile**
- **uicontextmenu(parent) → erzeugt ein Menü, dass mit rechter Maustaste anwählbar ist**

Die Eigenschaften der wichtigsten Objekte sind in Anhang B aufgeführt

3.2) Erzeugen grafischer Objekte:

...erfolg durch Aufruf der entsprechenden erzeugenden Funktion, für eine spätere Manipulation mit Angabe der Variable für die Handlingnummer

BSP: >> figNumber = figure;

>> menuNumber = uimenu(figure);
>> ui = uicontrol;

3.3) Eigenschaften grafischer Objekte:

lesen: >> get(handle, 'Property')
>> get(handle) → gibt alle Eigenschaften zurück

schreiben: >> set(handle, 'Property')
>> set(handle) → gibt alle Eigenschaften mit ihren möglichen Werten zurück

BSP:
```
>>set(figNumber,...
        'Name','Sprungfunktion',...
        'Menubar','none',...
        'Numbertitle','off',...
        'Position',[400 200  600 400]);
```

Tipp: Positionsangaben in Einheit Points oder Pixel machen, da kann man sich etwas drunter vorstellen (Points voreingestellt)
Groß- und Kleinschreibung der Properties vernachlässigbar
Zwischen Komma und Punkten kann auch ein Leerzeichen stehen

```
>>set(menuNumber,...
        'Label','&Datei',...
        'Position',[1]);
```

Tipp: mit '&' wird das D unterlegt → Menüpunkt kann auch mit Alt+D angewählt werden

3.4) Erstellung einer GUI-Oberfläche:

Es wird ein Beispielprogramm zur Darstellung einer Funktion mit mehreren Parametern entwickelt: BSP: Sprungfunktion $u(t-t_0) = \lim_{n\to\infty} \frac{2}{\pi} \arctan(e^{\frac{n(t-t_0)}{T}})$

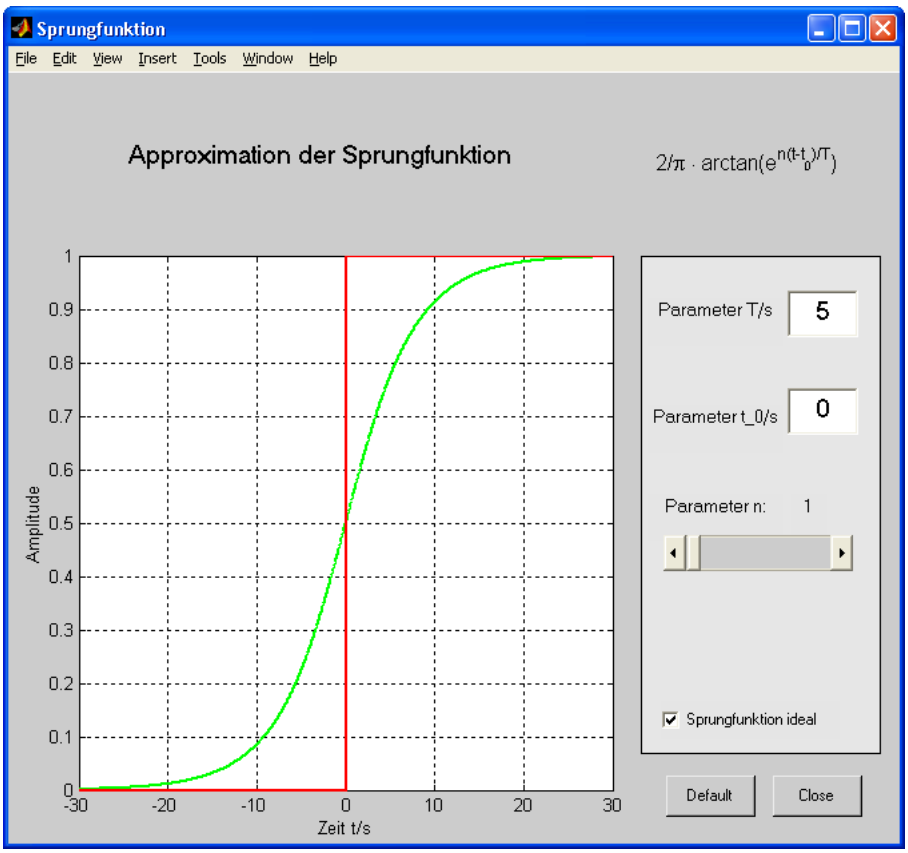
```
function sprungfunktion(action)

global DATA

background_color1 =[0.8 0.8 0.8];
background_color2 =[0.9 0.9 0.9];

schwarz = [0 0 0];


if nargin < 1
   action = 'init';
end

if strcmp(action,'init')

   fig = figure('Color',background_color1, ...
      'Position',[520 533 668 577],...
      'Units','normalized',...
      'Name','Sprungfunktion',...
      'Numbertitle','off',...
      'MenuBar','figure');

   ax = axes(...
      'Units','points',...
      'Position',[40 30 300 300],...
      'Nextplot','add',...
      'Visible','on',...
```

```
txtfield2 = uicontrol('Parent',fig, ...
   'Units','points',...
   'BackgroundColor',[1 1 1], ...
   'Position',[438 229.5 40 26], ...
   'FontSize',14, ...
   'Units','normalized', ...
   'Callback','sprungfunktion("refresh")',...
   'Style','edit', ...
   'String','0');

chbx = uicontrol('Parent',fig, ...
   'Units','points',...
   'BackgroundColor',background_color2, ...
   'Position',[367.5 59.25 100 20], ...
   'Units','normalized', ...
   'Callback','sprungfunktion("refresh")',...
   'Style','checkbox', ...
   'String','Sprungfunktion ideal',...
   'Value',[1]);

text = uicontrol('Parent',fig, ...
   'Units','points',...
   'FontSize',12, ...
   'BackgroundColor',background_color2, ...
   'Position',[360 220 75 26], ...
   'Units','normalized', ...
   'Style','text', ...
   'String','Parameter t_0/s');
```

```
    'Box','on');

text = uicontrol('Parent',fig,...
    'Units','points',...
    'FontSize',13,...
    'BackgroundColor',background_color1, ...
    'Position',[50 370 250 26], ...
    'Units','normalized', ...
    'Style','text', ...
    'String','Approximation der Sprungfunktion');

btn1 = uicontrol('Parent',fig, ...
    'Units','points',...
    'BackgroundColor',background_color1, ...
    'Position',[370 15 50 23.25], ...
    'Callback','sprungfunktion("default")',...
    'String','Default');

btn2 = uicontrol('Parent',fig, ...
    'Units','points',...
    'BackgroundColor',background_color1, ...
    'Position',[430 15 50 23.25], ...
    'Callback','sprungfunktion("close")',...
    'String','Close');

frm = uicontrol('Parent',fig, ...
    'Units','points',...
    'BackgroundColor',[0.9 0.9 0.9], ...
    'Position',[355.5 50 135 280], ...
    'Style','frame');

text = uicontrol('Parent',fig, ...
    'Units','points',...
    'FontSize',12, ...
    'BackgroundColor',background_color2, ...
    'String','Parameter T/s',...
    'Position',[360 280 75 26], ...
    'Units','normalized', ...
    'Style','text');

txtfield1 = uicontrol('Parent',fig, ...
    'Units','points',...
    'Position',[438 284.25 40 26], ...
    'BackgroundColor',[1 1 1], ...
    'FontSize',14, ...
    'Units','normalized', ...
    'Callback','sprungfunktion("refresh")',...
    'String','5',...
    'Style','edit');

  text = uicontrol('Parent',fig, ...
     'Units','points',...
     'FontSize',12, ...
     'BackgroundColor',background_color2, ...
     'Position',[360 170 75 26], ...
     'Units','normalized', ...
     'Style','text', ...
     'String','Parameter n:');

  wert_n = uicontrol('Parent',fig, ...
     'Units','points',...
     'FontSize',12, ...
     'BackgroundColor',background_color2, ...
     'Position',[440 170 20 26], ...
     'Units','normalized', ...
     'Style','text', ...
     'String','1');

  sld = uicontrol('Parent',fig, ...
     'Units','points', ...
     'BackgroundColor',background_color1, ...
     'Position',[368.25 153 106.5 20.25], ...
     'Callback','sprungfunktion("refresh")',...
     'Units','normalized', ...
     'Style','slider', ...
     'Value',[0.01]);

 % ======================================
  % Speichern aller Handling-Nummern in global DATA
  %DATA =[1    2   3   4    5          6       7  ];
  DATA = [fig ax chbx sld txtfield1 txtfield2 wert_n];
 % ======================================

sprungfunktion('start');
```

```
elseif strcmp(action,'start')


  t = linspace(-30,30,1000);
  t_0_string = get(DATA(6),'String');
  t_0 = str2num(t_0_string);
  T_string = get(DATA(5),'String');
  T = str2num(T_string);
  n = 100*get(DATA(4),'value');

  y=2/pi*atan(exp(n*(t-t_0)/T));

  axes(DATA(2));
  xlabel('Zeit t/s');
  ylabel('Amplitude');
  formel = title('2/\pi \cdot arctan(e^{n(t-t_0)/T})');
  set(formel,'Position',[45    1.15    9.1603],'FontSize',12);

  grid

  l=line(t,y);
  set(l,'Color','green','LineWidth',[1.5]);

  DATA(8) = l;
  l_ideal = line([-30;0;0;30],[0;0;1;1]);
  set(l_ideal,'Color','red','LineWidth',[1.5]);
  DATA(9) = l_ideal;
```

```
elseif strcmp(action,'default')

    set(DATA(3),'Value',[1]);
    set(DATA(4),'Value',[0.01]);
    set(DATA(5),'String','5');
    set(DATA(6),'String','0');

    sprungfunktion('refresh')


elseif strcmp(action,'close')

    delete(DATA(1));
    clear DATA;

elseif strcmp(action,'refresh')

    t_0_string = get(DATA(6),'String'); % Daten aus Eingabefeld auslesen
    t_0 = str2num(t_0_string);
    T_string = get(DATA(5),'String'); % Daten aus Eingabefeld auslesen
    T = str2num(T_string);
    T = max(T,eps); % soll Nulldivision verhindern
    n = 100*get(DATA(4),'value'); % Daten aus Eingabefeld auslesen

    set(DATA(7),'String',num2str(n)); % Angabe für n aktualisieren

    unten = min(t_0,-30);
    oben = max(t_0,30);

    t = linspace(unten,oben,1000);
    y=2/pi*atan(exp(n*(t-t_0)/T));

    set(DATA(8),'Xdata',t,'Ydata',y);

    if get(DATA(3),'Value')
       set(DATA(9),'Visible','on');
    else
       set(DATA(9),'Visible','off');
    end

end
```



3.5) Erstellung einer Animation:

Das obere Beispiel "Sprungfunktion" soll durch einen Druckknopf "Autoplay" erweitert werden, nach dessen Drücken n stetig um 0,1 erhöht und das Ergebnis direkt auf der GUI-Oberfläche angezeigt wird. Nach n = n_{max} (z.B. 100) beginnt die Animation von vorn. Close beendet die Animation und es soll eine Abfrage "Exit? Yes/No" erscheinen. Bei 'No' kehrt das Programm in den Anfangszustand zurück.
Name der Animation: sprungfunktion_auto.m

> alle Aufrufe der Funktion sprungfunktion('...') müssen durch sprungfunktion_auto('...') ersetzt werden

Am obigen Programm müssen folgende Änderungen vorgenommen werden:

- **Knopf hinzufügen:**

```
btn3 = uicontrol('Parent',fig, ...
   'Units','points',...
   'BackgroundColor',background_color1, ...
   'Position',[400 100 50 23.25], ...
   'Callback',' ',...
   'String','Autoplay');
```

Achtung: Frame muss vorher definiert werden, da sonst der Button hinter ihm liegt
Callback-Funktion siehe später

- **Unterprogramm (elseif) dazu:**

```
elseif strcmp(action,'auto')

   % Daten auslesen

   n = linspace(0,100,1001);
   k = 1;
      while get(DATA(2),'Userdata')

         y=2/pi*atan(exp(n(k)*(t-t_0)/T));

         set(DATA(8),'Xdata',t,'Ydata',y);
         set(DATA(7),'String',num2str(n(k))); % Angabe für n aktualisieren
         pause(0.1);

         k = k+1;

         if k > length(n)
            k = 1;
         end
      end
```

Trick: Wir brauchen eine Endlosschleife. Ist Matlab beschäftigt (z.B. in einer Endlosschleife), führt es nicht – oder nur langsam – einen weiteren Prozess aus. Benutze die Eigenschaft 'Userdata' eines GUI-Objekts (hier der Achsen), in der beliebige Strings oder Zahlenwerte als Informationen geführt werden können.
Es sei: 1 → autoplay; 0 → stop

- **globale Definitionen:**

  Da T, t_0 und t auch in diesem Teil der Funktion sprungfunktion_auto.m verfügbar sein müssen, werden sie als global deklariert:

```
global t_0;
global T;
global t
```

- **Anpassen aller Callback Eigenschaften:**

  Durch Anklicken eines beliebigen Interaktionselements soll die Animation gestoppt,

```
'Callback','global DATA; set(DATA(2),''Userdata'',[0]);sprungfunktion_auto(''refresh'')',...
```

  bei Drücken des Autoplay Knopfes gestartet werden.

```
'Callback','global DATA; set(DATA(2),''Userdata'',[1]);sprungfunktion_auto(''auto'')',...
```

  Das, was als String in der Callback-Eigenschaft enthalten ist, muss quasi als unabhängige Funktion betrachtet werden. Deshalb ist die Angabe global DATA notwendig, damit die Variable DATA auch innerhalb dieser Funktion verfügbar ist. (Deshalb kann auch nicht einfach die Variable ax (=DATA(2)) genommen werden, es sei denn, sie ist global deklariert)

- **Eigenschaft der Achsen erweitern:**

```
'Userdata',[0],...
```

  muss von vorn herein bei den Definition der Eigenschaften der Achsen hinzugefügt werden [bei if strcmp(action,'init') ... unter ax = axes(...)].

- **Zeichenmodus anpassen:**

  Als Eigenschaft für den Graphen muss der Zeichenmodus folgendermaßen geändert werden: 'Erasemode' = 'xor'

```
set(DATA(8),'Erasemode','xor');
```

- **Schließen- Funktion optimieren: verwende Abfragefenster**

```
wahl = questdlg('Programm Beenden?');
if strcmp(wahl,'Yes')
   delete(DATA(1));
   clear DATA;
   clear T;
   clear t;
   clear t_0;
end
```

- **Fehlerausgabe vermeiden:**

  Durch Öffnen des Schließen-Abfragefensters wird ein Interrupt ausgelöst und die Aktivität von Matlab angehalten. Wählt man bei dem Abfragefenster 'Programm

Beenden? Yes', will Matlab nach dem Löschen der GUI-Animation auf Variablen und Objekte zugreifen und die unterbrochene Animation weiterlaufen lassen, die jedoch nicht mehr existiert. Um eine Fehlermeldung zu unterdrücken, wird die while-Schleife in einen try- Block gesetzt:

```
try
   while get(DATA(2),'Userdata')
      ...
   end
end
```

4) Anhang:

A] Eigenschaften verschiedener grafischer Objekte: >>set(handle)

figure:	axes:
Alphamap BackingStore: [{on} | off] CloseRequestFcn: string -or- function handle -or- cell array Color Colormap CurrentAxes CurrentCharacter CurrentObject CurrentPoint Dithermap DithermapMode: [auto | {manual}] DoubleBuffer: [on | {off}] FileName IntegerHandle: [{on} | off] InvertHardcopy: [{on} | off] KeyPressFcn: string -or- function handle -or- cell array MenuBar: [none | {figure}] MinColormap Name NextPlot: [{add} | replace | replacechildren] NumberTitle: [{on} | off] PaperUnits: [{inches} | centimeters | normalized | points] PaperOrientation: [{portrait} | landscape | rotated] PaperPosition PaperPositionMode: [auto | {manual}] PaperSize PaperType: [{usletter} | uslegal | A0 | A1 | A2 | A3 | A4 | A5 | B0 | B1 | B2 | B3 | B4 | B5 | arch-A | arch-B | arch-C | arch-D | arch-E | A | B | C | D | E | tabloid | <custom>] Pointer: [crosshair | fullcrosshair | {arrow} | ibeam | watch | topl | topr | botl | botr | left | top | right | bottom | circle | cross | fleur | custom] PointerShapeCData PointerShapeHotSpot Position Renderer: [{painters} | zbuffer | OpenGL] RendererMode: [{auto} | manual] Resize: [{on} | off] ResizeFcn: string -or- function handle -or- cell array SelectionType: [normal | open | alt | extend] ShareColors: [{on} | off] Units: [inches | centimeters | normalized | points | {pixels} | characters] WindowButtonDownFcn: string -or- function handle -or- cell array WindowButtonMotionFcn: string -or- function handle -or- cell array WindowButtonUpFcn: string -or- function handle -or- cell array WindowStyle: [{normal} | modal] ButtonDownFcn: string -or- function handle -or- cell array Children Clipping: [{on} | off] CreateFcn: string -or- function handle -or- cell array DeleteFcn: string -or- function handle -or- cell array BusyAction: [{queue} | cancel] HandleVisibility: [{on} | callback | off] HitTest: [{on} | off] Interruptible: [{on} | off] Parent Selected: [on | off] SelectionHighlight: [{on} | off] Tag UIContextMenu UserData Visible: [{on} | off]	ALim ALimMode: [{auto} | manual] AmbientLightColor Box: [on | {off}] CameraPosition CameraPositionMode: [{auto} | manual] CameraTarget CameraTargetMode: [{auto} | manual] CameraUpVector CameraUpVectorMode: [{auto} | manual] CameraViewAngle CameraViewAngleMode: [{auto} | manual] CLim CLimMode: [{auto} | manual] Color ColorOrder DataAspectRatio DataAspectRatioMode: [{auto} | manual] DrawMode: [{normal} | fast] FontAngle: [{normal} | italic | oblique] FontName FontSize FontUnits: [inches | centimeters | normalized | {points} | pixels] FontWeight: [light | {normal} | demi | bold] GridLineStyle: [- | -- | {:} | -. | none] Layer: [top | {bottom}] LineStyleOrder LineWidth MinorGridLineStyle: [- | -- | {:} | -. | none] NextPlot: [add | {replace} | replacechildren] PlotBoxAspectRatio PlotBoxAspectRatioMode: [{auto} | manual] Projection: [{orthographic} | perspective] Position TickLength TickDir: [{in} | out] TickDirMode: [{auto} | manual] Title Units: [inches | centimeters | {normalized} | points | pixels | characters] View XColor XDir: [{normal} | reverse] XGrid: [on | {off}] XLabel XAxisLocation: [top | {bottom}] XLim XLimMode: [{auto} | manual] XMinorGrid: [on | {off}] XMinorTick: [on | {off}] XScale: [{linear} | log] XTick XTickLabel XTickLabelMode: [{auto} | manual] XTickMode: [{auto} | manual] YColor YDir: [{normal} | reverse] YGrid: [on | {off}] YLabel YAxisLocation: [{left} | right] YLim YLimMode: [{auto} | manual] YMinorGrid: [on | {off}] YMinorTick: [on | {off}] YScale: [{linear} | log] YTick YTickLabel YTickLabelMode: [{auto} | manual] YTickMode: [{auto} | manual] ZColor ZDir: [{normal} | reverse]

plot:

Color
EraseMode: [{normal} | background | xor | none]
LineStyle: [{-} | -- | : | -. | none]
LineWidth
Marker: [+ | o | * | . | x | square | diamond | v | ^ | > | < | pentagram | hexagram | {none}]
MarkerSize
MarkerEdgeColor: [none | {auto}] -or- a ColorSpec.
MarkerFaceColor: [{none} | auto] -or- a ColorSpec.
XData
YData
ZData

ButtonDownFcn: string -or- function handle -or- cell array
Children
Clipping: [{on} | off]
CreateFcn: string -or- function handle -or- cell array
DeleteFcn: string -or- function handle -or- cell array
BusyAction: [{queue} | cancel]
HandleVisibility: [{on} | callback | off]
HitTest: [{on} | off]
Interruptible: [{on} | off]
Parent
Selected: [on | off]
SelectionHighlight: [{on} | off]
Tag
UIContextMenu
UserData
Visible: [{on} | off]

ZGrid: [on | {off}]
ZLabel
ZLim
ZLimMode: [{auto} | manual]
ZMinorGrid: [on | {off}]
ZMinorTick: [on | {off}]
ZScale: [{linear} | log]
ZTick
ZTickLabel
ZTickLabelMode: [{auto} | manual]
ZTickMode: [{auto} | manual]
ButtonDownFcn: string -or- function handle -or- cell array
Children
Clipping: [{on} | off]
CreateFcn: string -or- function handle -or- cell array
DeleteFcn: string -or- function handle -or- cell array
BusyAction: [{queue} | cancel]
HandleVisibility: [{on} | callback | off]
HitTest: [{on} | off]
Interruptible: [{on} | off]
Parent
Selected: [on | off]
SelectionHighlight: [{on} | off]
Tag
UIContextMenu
UserData
Visible: [{on} | off]

uicontrol:

BackgroundColor
Callback: string -or- function handle -or- cell array
CData
Enable: [{on} | off | inactive]
FontAngle: [{normal} | italic | oblique]
FontName
FontSize
FontUnits: [inches | centimeters | normalized | {points} | pixels]
FontWeight: [light | {normal} | demi | bold]
ForegroundColor
HorizontalAlignment: [left | {center} | right]
ListboxTop
Max
Min
Position
String
Style: [{pushbutton} | togglebutton | radiobutton | checkbox | edit | text | slider | frame | listbox | popupmenu]
SliderStep
TooltipString
Units: [inches | centimeters | normalized | points | {pixels} | characters]
Value

ButtonDownFcn: string -or- function handle -or- cell array
Children
Clipping: [{on} | off]
CreateFcn: string -or- function handle -or- cell array
DeleteFcn: string -or- function handle -or- cell array
BusyAction: [{queue} | cancel]
HandleVisibility: [{on} | callback | off]
HitTest: [{on} | off]
Interruptible: [{on} | off]
Parent
Selected: [on | off]
SelectionHighlight: [{on} | off]
Tag
UIContextMenu
UserData
Visible: [{on} | off]

title:

Color
EraseMode: [{normal} | background | xor | none]
Editing: [on | off]
FontAngle: [{normal} | italic | oblique]
FontName
FontSize
FontUnits: [inches | centimete
FontWeight: [light | {normal} | demi | bold]
HorizontalAlignment: [{left} | center | right]
Position
Rotation
String
Units: [inches | centimeters | normali
Interpreter: [{tex} | none]
VerticalAlignment: [top | cap | {middle} | baseline | bottom]

ButtonDownFcn: string -or- function handle -or- cell array
Children
Clipping: [{on} | off]
CreateFcn: string -or- function handle -or- cell array
DeleteFcn: string -or- function handle -or- cell array
BusyAction: [{queue} | cancel]
HandleVisibility: [{on} | callback | off]
HitTest: [{on} | off]
Interruptible: [{on} | off]
Parent
Selected: [on | off]
SelectionHighlight: [{on} | off]
Tag
UIContextMenu
UserData
Visible: [{on} | off]

uimenu:	
Accelerator Callback: string -or- function handle -or- cell array Checked: [on | {off}] Enable: [{on} | off] ForegroundColor Label Position Separator: [on | {off}] ButtonDownFcn: string -or- function handle -or- cell array Children Clipping: [{on} | off] CreateFcn: string -or- function handle -or- cell array DeleteFcn: string -or- function handle -or- cell array BusyAction: [{queue} | cancel] HandleVisibility: [{on} | callback | off] HitTest: [{on} | off] Interruptible: [{on} | off] Parent Selected: [on | off] SelectionHighlight: [{on} | off] Tag UIContextMenu UserData Visible: [{on} | off]	

B] Übersicht einiger wichtiger Funktionen