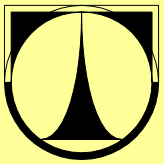


TECHNICKÁ UNIVERZITA V LIBERCI
Hálkova 6, 461 17 Liberec 1, CZ

Fakulta mechatroniky a mezioborových inženýrských studií

Einführung in MATLAB

Doc. Ing. Osvald Modrlák, CSc.

Juni 2004

Katedra řídicí techniky

Inhalt

Dieser Studientext unterstützt die Vorlesungsreihe „Einführung in MATLAB und eine Anwendung in der Regelungstechnik“ im Rahmen der bilateralen Zusammenarbeit der TU Liberec und FH Zittau/Görlitz im Programm Erasmus.
Ich danke hier Prof. Ch. Rähder, dem langjährigen verantwortlichen Partner für das Austauschpraktikum, und Prof. F. Worlitz, der der verantwortliche Partner für Projektierungspraktikum ist, für Ihren Einsatz bei Vorbereitung und Durchführung der Praktika. Im diesen Rahmen konnte der vorgelegte Taxt entstehen.
Besonders möchte ich meinem langjährigen Ansprechpartner bei dem Austauschpraktikum TU Liberec - FH Zittau/Görlitz, den Herrn Gerold Anders für zahlreiche Anregungen und kritischen Durchsicht des Manuskripts, danken.

Der vorgelegte Studientext ist eine Einführung in die MATLAB-Umgebung, wo interaktive und automatische Berechnungen durchgeführt werden können. Dieser Beitrag stellt selbstverständlich keinen erschöpfenden Überblick über das System MATLAB. Er soll eher einen vereinfachten Einblick in die ganze Problematik anbieten.

1 VORWORT

MATLAB ist eine hochleistungsfähige Software zur Lösung und graphischen Darstellung von wissenschaftlich-technischen Aufgaben. MATLAB vereint mathematische Berechnungen, Visualisierung und eine mächtige Programmiersprache zu einer flexiblen Umgebung von technischen Berechnungen. Durch den offenen Aufbau lassen sich MATLAB und die weiteren Systeme der MATLAB Familie unkompliziert kombinieren, z.B. zur Datenauszuwertung.
MATLAB enthält :

- Bibliothek mit einer Vielzahl von mathematischen Funktionen
- Bausteine für die Lösung von wissenschaftlich-technischen Aufgabenstellungen
- Entwicklungsmittel für Algorithmusbildung
- Modellbildung
- Simulation
- Graphik 2D und 3D
- Datenanalyse und- Auswertung
- Interaktive Arbeitsweise mit Hilfefunktionen

SIMULINK – ist ein System für nichtlineare Simulation. Durch die Signalflussbilder werden modellierte Systeme dargestellt und simuliert. Durch das MATLAB-Interface werden die Signalflussbilder mit mathematischen Funktionen, Bausteinen und Graphik verbunden.

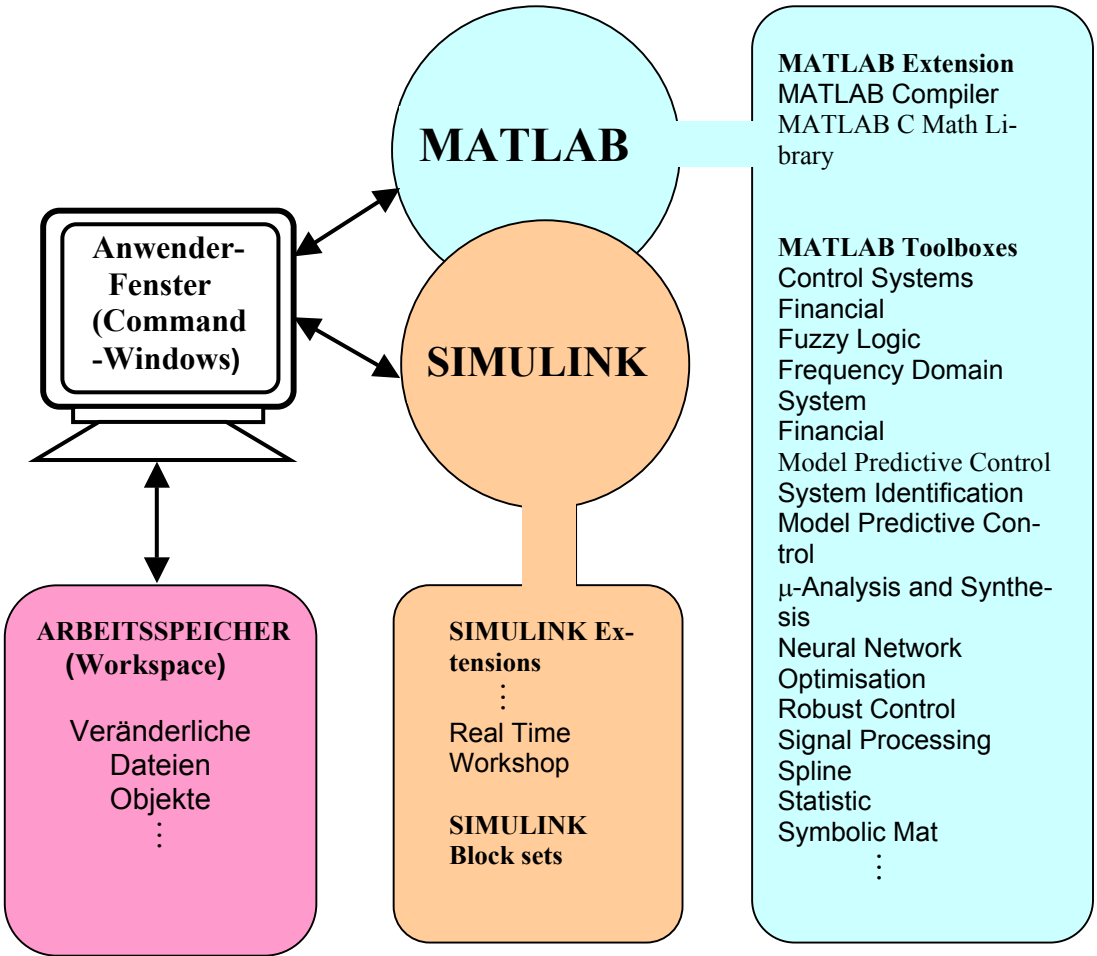


Bild 1.1 *MATLAB/SIMULINK Tools und Erweiterungen*

Eine Vielzahl von mathematischer Funktionen bildet die Grundlage, die durch Bausteine, wie z.B. im Bankwesen durch **Financial Toolbox**, oder in der Regelungstechnik durch die **Control System Toolbox**, und ihren vielen Erweiterungen und Toolboxes wie **System Identifikation**, **Signal Processing, Fuzzy Logic, Neuronal Network, Robust Control, Symbolic Math** fachspezifisch ergänzt wird. Ein Überblick über Bausteine und Tools ist im Bild1.1 dargestellt.
Diese Vorlesung erhebt keinesfalls den Anspruch, MATLAB detailliert zu beschreiben. Das Ziel ist es, eine klare und vereinfachte Einführung anzubieten, so dass die Leser und Hörer so schnell wie möglich selbständig mit MATLAB arbeiten können.
Die Unterlagen sind erreichbar unter der Internet-Adresse

http://www.fm.vslib.cz/~krt/krt_cz/vyuka/text/matlab/html.htm
http://www.fm.vslib.cz/~krt/krt_cz/praktikum/Einfuhrung_MATLAB.pdf

2 MATLAB - ANWENDERUMGEBUNG (ENVIRONMENT)

Der MATLAB – Anwender arbeitet in dem ***Anwenderfenster*** (Command Windows, siehe Bild 1.2), wo er die Anweisungen direkt über die Tastatur eingeben kann und wo auch alle Rechnungsergebnisse, graphische Darstellungen und Meldungen dargestellt werden. Es ist auch möglich, Anweisungen indirekt mit Hilfe des **MATLAB Editors** als eine Skriptdatei – **script file -** mit der Extension **.m**, einzugeben und zu editieren.

MATLAB bildet bei jeder Sitzung (MATLAB Session) einen ***Arbeitsspeicher*** (Workspace), wo Veränderliche, Dateien, Objekte usw. während der Sitzung gespeichert werden (siehe Bild 1.2). Deswegen müssen die Ergebnisse, Dateien usw., die weiter benutzt werden sollen, gespeichert werden!

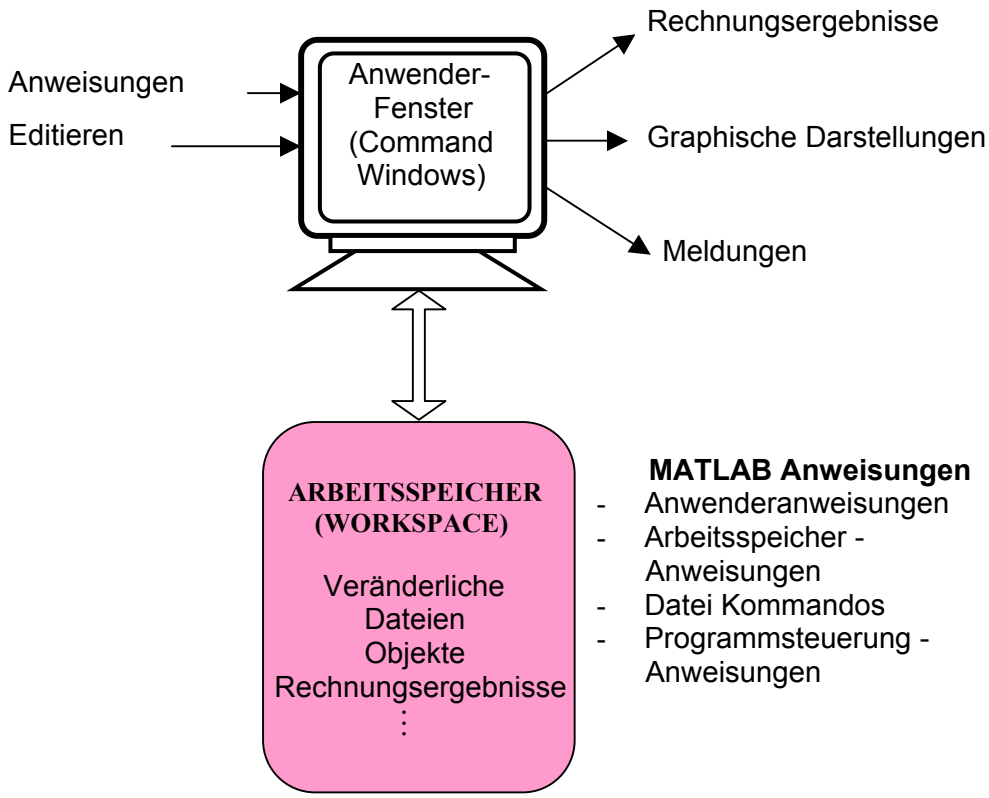


Bild 1.2. MATLAB Struktur: Anwenderfenster und Arbeitsspeicher

Aus der Struktur auf dem Bild 1.2 folgt, dass außer den Anwenderanweisungen noch Arbeitsspeicheranweisungen, Anweisungen der Programmsteuerung und Dateikommandos für die Verwaltung des Arbeitsspeichers verwendet werden. Diese werden später beschrieben.

2.1 MATLAB ANWENDERFENSTER, ANWEISUNGEN, EDITIEREN

2.1.1 Direkte und Indirekte Eingabe, direkte Eingabe von Matrizen und Vektoren

Jede Eingabeanweisung im MATLAB Command Window beginnt hinter dem Promtzeichen >> und wird durch das Betätigen der Return-Taste ↵ abgeschlossen, was zu ihrer Durchführung führt.

Direkte Eingabe: z.B. die Addition von drei Zahlen

```
>> 12+3+7
```

liefert

```
ans =
     22
```

Wenn für das nummerische Ergebnis keine Variable vereinbart ist, wird es der Variable `ans-(ansver)` zugewiesen.
Soll dagegen das Ergebnis der Addition einer Variablen zugewiesen werden, dann ist zu schreiben

```
>> x=12+3+7
```

und es ergibt sich

```
x =
     22
```

Wenn bei einer direkten Eingabe ein Fehler entstanden ist, muss die Anweisung neu geschrieben werden.

> In MATLAB werden alle Anweisungen in den Arbeitsspeicher gespeichert, so dass bei einer Fehlerkorrektur durch die Taste ↑ oder ↓ die gewünschte Anweisung gefunden werden kann. Die Anweisung wird korrigiert und durch das Drücken der Taste ↵ wird die Korrektur abgeschlossen und die Anweisung wird durchgeführt.

Indirekte Eingabe*:* Diese erfolgt mit Hilfe eines Texteditors über eine Skriptdatei im ASCII-code. Skriptdateien werden als M-Datei bzw. M-Files bezeichnet. Das Fenster des Texteditors (Editor/Debugger) im MATLAB Command Window ist im Bild.1.3 dargestellt. Nach dem Zeichen % werden alle weiteren Zeichen bis zum Ende der Zeile als Bemerkungen genommen.
Mit Hilfe des Menüs können Sie einen Abschnitt des Programms zur Auswertung im Command Window auswählen.

Eine M-Datei mit dem Namen "**`name.m`**" wird über das Roll-Menü " **Tool**" durch „**Run**" oder vom MATLAB Command Window aus mit „**`name`**" gestartet.

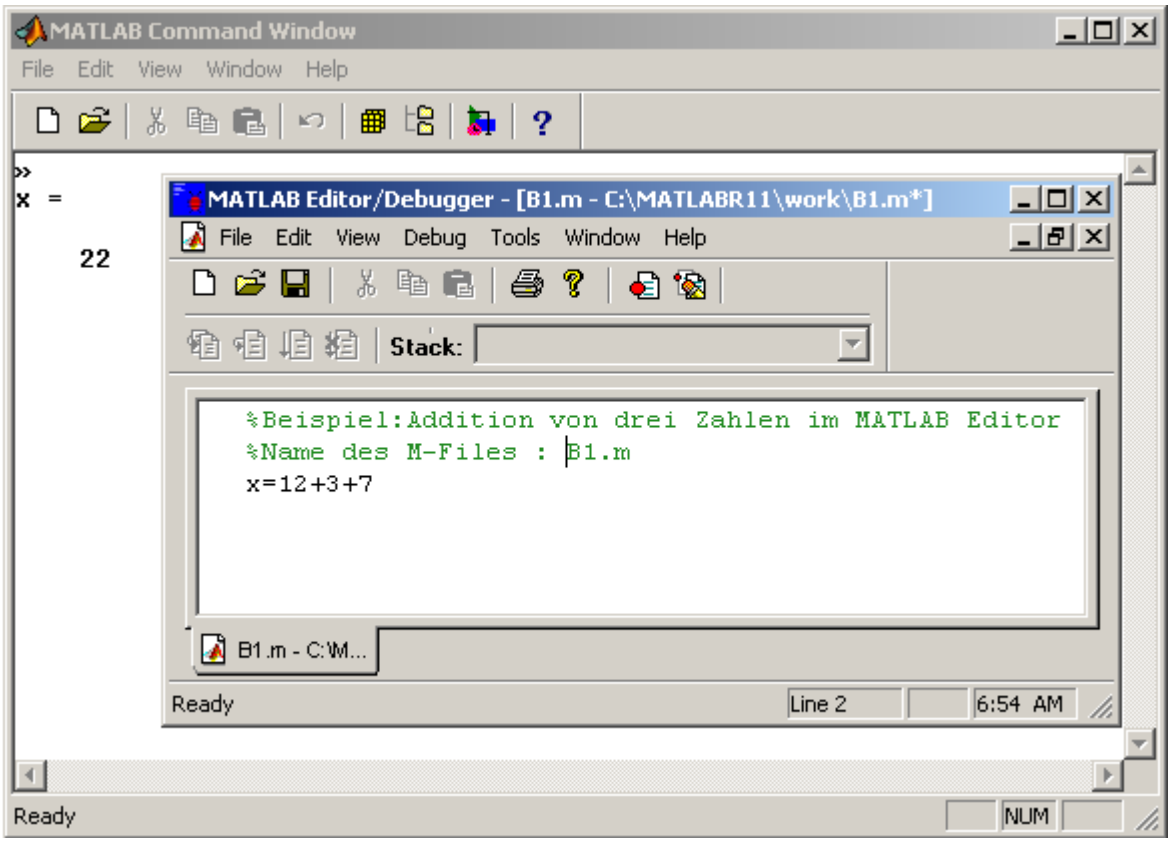


Bild.1.3 *MATLAB Editor/Debugger*

Matrixeingabe : Direkte Eingabe:

[Matrixelement Leerzeichen .. Semikolon]

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

[Matrixelement Leerzeichen ... Return-Taste
Matrixelement Leerzeichen ...]

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

Kennzeichnung und Aufruf von Elementen einer Matrix

Das Element in der `i`-ten Zeile `j`-ten Spalte einer Matrix A vom Format `(m,n)` wird wie folgt abgebildet `A(i,j)`

```
>> w=A(1,3)
   w=
     3
```

```
>> j=2;w1=A(1+j,1)
   w1=
       7
```

```
>> A(1,3)+A(2,3)
   ans=
       9
```

Auswahl von Elementen oder Reihen einer Matrix

Ein Doppelpunkt anstelle des Zeilenzählers bedeutet, dass alle Zeilen angesprochen werden. Entsprechendes gilt auch für den Spaltenzähler

Beispiel

```
» Data=[0 0 0;1 2 0.5;2 4 2;3 6 4.5;4 8 8;5 10 12.5]

Data =

         0         0         0
    1.0000    2.0000    0.5000
    2.0000    4.0000    2.0000
    3.0000    6.0000    4.5000
    4.0000    8.0000    8.0000
    5.0000   10.0000   12.5000

» t=Data(:,1); y1=Data(:,2); y2=Data(:,3);

» t              » y1             » y2
t =              y1 =             y2 =

     0                0                0
     1                2           0.5000
     2                4           2.0000
     3                6           4.5000
     4                8           8.0000
     5               10          12.5000
```

In MATLAB existieren Funktionen, die Grundmatrizen generieren.

`zeros`	Alle Elemente einer Matrix im Format `(n,m)` sind gleich Null
`ones`	Alle Elemente einer Matrix im Format `(n,m)` sind gleich Eins
`eye`	Einheitsmatrix vom Format `(n,n)`
`rand`	Matrixelemente einer Matrix im Format `(n,m)` gleichen Zufallszahlen mit der gleichmäßigen Verteilung
`randn`	Matrixelemente einer Matrix im Format `(n,m)` gleichen Zufallszahlen mit der normalen Verteilung

```
>> Z=zeros(2,3)
   Z=
     0   0   0
     0   0   0
```

```
>> M=2*ones(3,3)
   M=
     2   2   2
     2   2   2
     2   2   2
```

```
>> R=randn(3,2)
   R=
    -0.4326   0.2877
    -1.6656  -1.1465
     0.1253   1.1909
```

```
>> R=rand(1,2)
   R=
     0.4868   0.8913
```

```
>> E=eye(2)
   E=
     1   0
     0   1
```

```
» AS=[zeros(3,1) eye(3);-2 -5 -3 -1]

AS =

     0     1     0     0
     0     0     1     0
     0     0     0     1
    -2    -5    -3    -1
```

2.1.2 Operatoren, Operationen, Spezielle Werte, Matrixoperationen

Operatoren und spezielle Zeichen sind in der Tabelle 1 dargestellt.

Zeichen	Beschreibung	Zeichen	Beschreibung
+	Plus-Addition	=	Zuordnung
-	Minus-Subtraktion	==	Gleichheit
*	Matrix-Multiplikation	< >	Relationsoperatoren
.*	Multiplikation von Matrix-Elementen	&	Logisches AND
^	Potenz	|	Logisches OR
.^	Potenz von Matrixelementen	~	Logische Negation
\	Dividieren von links	XOR	Log. Exclusive OR
/	Dividieren von rechts	.	Dezimalpunkt
;	Semikolon	'	Transposition
…	Fortsetzung	%	Kommentar
[]	Eckige Klammer	()	Runde Klammer

Tab.1

Arithmetische Operationen sind: Addition, Subtraktion, Multiplikation, Division von rechts und links und Potenzen. Die Beispiele mit den Veränderlichen x, y, z sind im Bild 1.3 dargestellt:

```
» x=2      » z=5
x =        z =
     2          5
» y=x+z  » y=x-z  » y=x*z  » y=z/x  » y=z\x  » y=z^x
y =      y =      y =      y =      y =      y =
     7         -3        10        2.50      0.40      25
```

Bild 1.3. Arithmetische Operationen

Spezielle Werte

1) Imaginäre Zahl $i = j = \sqrt{-1}$:

```
>> x = i
   x =
         0 + 1.0000i
```

2) Infinity entspricht dem Zeichen ∞, z.B. das Ergebnis bei Division durch Null

```
>> x =1/0
Warning: Divide by zero
   x=
     Inf
```

3) Division Null mit Null, d.h. Not-a-Number

```
>> x =0/0
```

```
Warning: Divide by zero
  x=
    NaN
```

4) π, das Verhältnis des Kreisumfangs U zum Durchmesser d : $\pi = \frac{U}{d}$

```
>> x = pi
   x =
       3.14159265358979
```

Matrixoperationen

5) **Funktion inv** *Inversion einer Matrix*

```
» AI=inv(AS)

AI =

   -2.5000   -1.5000   -0.5000   -0.5000
    1.0000         0         0         0
         0    1.0000         0         0
         0         0    1.0000         0
```

```
» AS*AI

ans =

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

6) **Funktion rank** *Rank einer Matrix*

```
» rank(AS)

ans =

     4
```

7) **Funktion det** *Determinante einer Matrix*

```
» det(AS)

ans =

     2
```

8) **Funktion norm** *Norm einer Matrix*

```
» norm(A)

ans =

    3.1249
```

9) **Funktion trace** *Summe der diagonalen Elemente einer Matrix*

```
» trace(AS)

ans =

    -1
```

10) Funktion **global** Definiert globale Variable

Variablen einer Datei einer Funktions- oder Unterfunktionsroutine sind lokale Variable. Die Variablen einer Unterfunktionsroutine werden über die Argumente der Funktion und über globale Variable übertragen.

Syntax der Funktion **global X Y Z** Veränderliche **X Y Z** sind global in ganzem Arbeitsspeicher

Auf dem Bild 1.4a ist ein Anwendungsbeispiel der Funktion „global“ bei einer externen Funktion abgebildet und auf dem Bild 1.4b ist die Anwendung der „global Funktion“ bei Übertragung der Variablen zwischen dem Programm **PIDopt.m** und SIMULINK – Modelle **PIDkri.mdl** und **PIDsim.mdl** dargestellt.

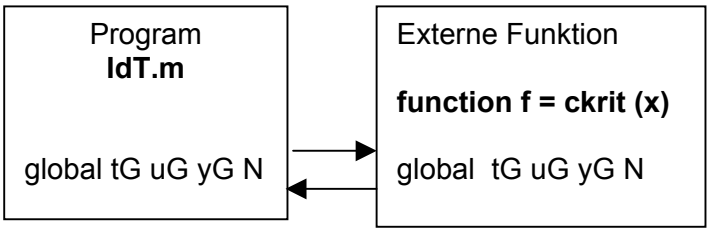


Bild 1.4a. Anwendungsbeispiel der Funktion global bei einer externen Funkti-

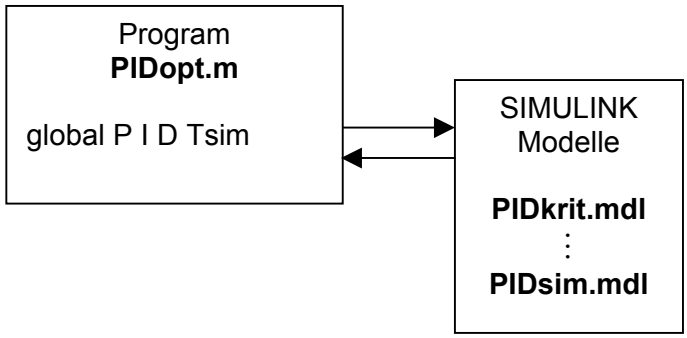


Bild 1.4b. Anwendungsbeispiel der Funktion global bei einem SIMULINK Mo-

11) Funktion **conv** Definiert die Polynommultiplikation

Syntax der Funktion

W=conv (A,B) Führt die Multiplikation von Polynomen **A,B** durch.

Polynommultiplikation

$$W(k)=\sum_{j} A(j)B(k+1-j)$$

12) Funktion roots Lösungen eines Polynoms (Wurzelberechnung)

Syntax der Funktion : **R = roots (A)** Berechnung der Wurzeln vom Polynom **A** und Speicherung in den Spaltenvektor **R**

Beispiel :

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

roots(A)

ans =

  -2.3247
  -0.3376 + 0.5623i
  -0.3376 - 0.5623i
```

```
» roots(B)

ans =

         0
   -0.5000
```

2.1.3 Mathematische elementare Funktionen

Eine Übersicht von elementaren mathematischen Funktionen ist in der Tabelle 2 zusammengestellt.

Funktion	Funktionsbezeichnung	Funktion	Funktionsbezeichnung
acos (x)	Arkuskosinus	***exp(x)***	Exponentialfunktion e^x
acosh (x)	Arkuskosinus hyperbolicus	***log(x)***	Logarithmusfunktion $\log_e x$
asin(x)	Arkussinus	***log10(x)***	Logarithmusfunktion $\log_{10} x$
asih(x)	Arkussinus hyperbolicus	***sqrt(x)***	Quadratwurzel
atan(x)	Arkustangens	***abs(x)***	Absolutbetrag einer komplexen Zahl
atanh(x)	Arkustangens hyperbolicus	***conj(x)***	Konjugiert komplexe Zahl
cos(x)	Kosinus	***imag(x)***	Imaginärteil einer komplexen Zahl
cosh(x)	Kosinus hyperbolicus	***real(x)***	Realteil einer komplexen Zahl
sin(x)	Sinus	***sign(x)***	Signumfunktion
sinh(x)	Sinus hyperbolicus	***round(x)***	Abrundung
tan(x)	Tangens	***rem(x,y)***	Rest nach Division x/y
tanh(x)	Tangens hyperbolicus		

Tabelle 2

Beispiele:

1) Berechnung von $\sin(\pi/2)+\cos(2\pi)*\exp(2.5)$

```
>> sin(pi/2)+cos(pi)*exp(2.5)
ans =
     11.825
```

2) Berechnung von: $z=-3+4i, \|z\|=\sqrt{-3^2+4^2}$

```
>> z=-3+4i
   z =
      -3.0000+4.0000i
>> za = abs(z)
   za =
       5
```

3) Berechnung vom Rest nach Division 23/3

```
>> rem(23,3)
ans=
    2
```

2.2 GRAPHISCHE DARSTELLUNG

Die Daten sind in Form von Vektoren- oder Matrizenvariablen abgespeichert und können durch Funktionen graphisch zwei- oder dreidimensional vielfältig dargestellt werden. Dazu stehen im MATLAB verschiedene Funktionen zur Verfügung:

- Funktionen für graphische Darstellung, Tab. 3a
- Anweisungen zu Beschriftungen, Tab. 3b
- Steuerbefehle der Graphik, Tab. 3c

plot	zweidimensionale x-y Darstellung mit linearen Achsen
plot3	dreidimensionale Darstellung mit linearen Achsen x-y
loglog	zweidimensionale x-y Darstellung mit beiden Achsen logarithmisch
semilogx	zweidimensionale x-y Darstellung mit der x - Achsen logarithmisch
semilogy	zweidimensionale x-y Darstellung mit der y - Achsen logarithmisch
plotyy	Graphik mit der y-Achse auf der linken Seite

Tabelle 3a. Funktionen für graphische Darstellung

title	Überschrift der Graphik
xlabel	x-Achse Beschriftung
ylabel	y-Achse Beschriftung
zlabel	z-Achse Beschriftung
legend	fügt eine Legende in die Graphik ein
text	fügt einen positionierten Text in die Graphik ein
gtext	fügt einen positionierten Text mit der Maus in die Graphik ein

Tabelle 3b. Anweisungen zur Beschriftungen

axis	Manuelle Einstellungen der Bereiche der x- und y-Achse
hold	Einfrieren der Darstellung für Überlappen der Grafiken
clf	Löschen des laufenden Bildes
subplot	Unterteilung des Bildes
ginput	Eingabe mit der Maus

Tabelle 3c. Steuerbefehle der Graphik

Die Funktionen und Anweisungen können über **help** aufgerufen werden. In diesem Text werden nur die wichtigsten Funktionen erklärt.

2.2.1 Grundform des plot- Befehls

Funktion plot **Zeichnet zweidimensionale lineare x-y Darstellung**

Syntax der Funktion

```
plot (Y)
plot (X1,Y1,…)
plot (X1,Y1,LineSpec,…)
```

plot (Y) Bildet einen Graphen, in dem an der Ordinate die Spaltenelemente der Matrix **Y** und die Abszisse die Zeilenindices der Matrix **Y** eingetragen sind **.**

plot (X1,Y1,…) Bildet einen Graphen, in dem an die Abszisse der Vektor **X1** eingetragen wird und an die Ordinate die Spaltenelemente des Vektors **Y1** eingetragen werden usw.

plot (X1,Y1,LineSpec,…) Bildet einen Graphen, in dem an die Abszisse der Vektor **X1** eingetragen wird und an die Ordinate Spaltenelemente des Vektors **Y1** eingetragen werden. Durch die Anweisung **LineSpec** werden einzelne Linien spezifiziert. Eine genaue Beschreibung finden Sie im Help (help plot).

Beispiel für die Anwendung von plot : Bild 2.1, 2.2

```
» plot (Data)
» t=Data(:,1); y1=Data(:,2); y2=Data(:,3);
» plot (t,y1,t,y2)
```

```
Data =
         0         0         0
    1.0000    2.0000    0.5000
    2.0000    4.0000    2.0000
    3.0000    6.0000    4.5000
    4.0000    8.0000    8.0000
    5.0000   10.0000   12.5000
```

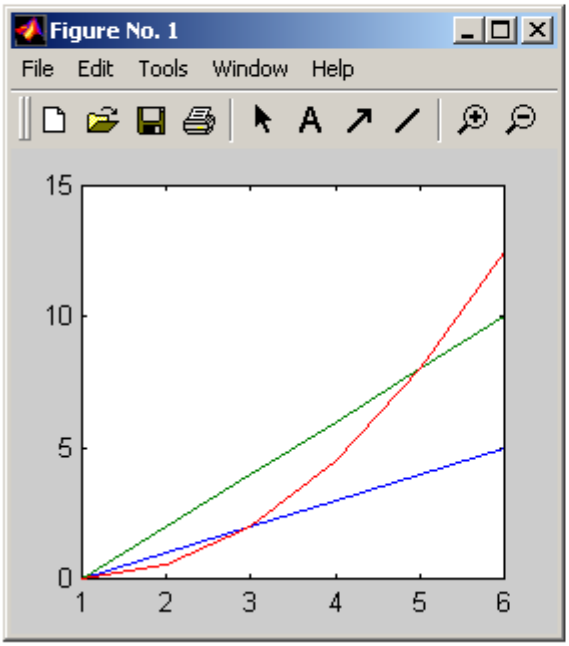


Bild 2.1 Graphdarstellung **plot** (Data)

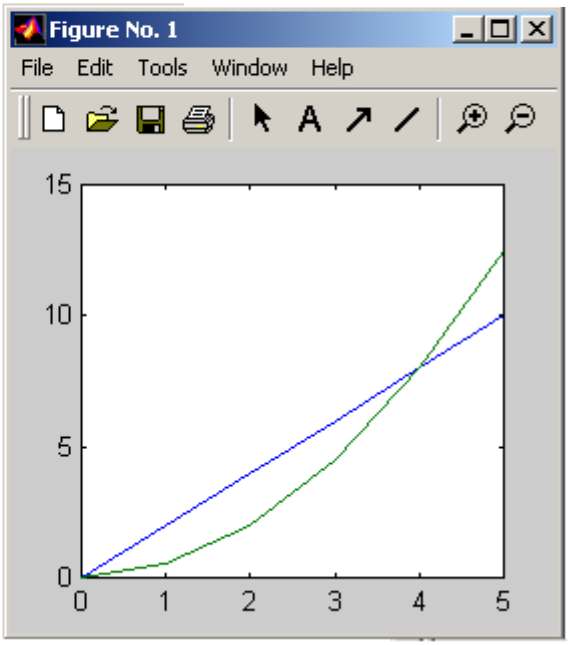


Bild 2.2 Graphdarstellung **plot** (t,y1,t,y2)

Die Grenzen der Achsen werden normalweise automatisch durch MATLAB bestimmt. Für Grenzänderung wird die Funktion **axis** benutzt. Die Syntax der Funktion :

axis ([xmin xmax ymin ymax])
xmin ... untere Grenze von x
xmax ...obere Grenze von x
ymin ... untere Grenze von y
ymax ...obere Grenze von y

Eine Funktion **plot** öffnet automatisch ein Bild-Fenster, das aus einem graphischen Objekt besteht. Wenn ein Fenster schon geöffnet ist, wird dies mit dem neuen Graphen überschrieben, d.h. in dem Bild-Fenster wird nur der letzte Graph dargestellt. Um dies zu vermeiden,

wird die Anweisung **figure (n)** im Programm benutzt, wobei **n** die Graphnummer (Figure No.n) im Kopfbild ist. Die Anwendung der Funktion **axis** und **figure** wird in dem folgenden Beispiel gezeigt. Die Graphen sind auf dem Bild 2.3 dargestellt.

```
%B_Plot11.m Beispiel"plot"
plot(Data)
figure(1)
t=Data(:,1);y1=Data(:,2);y2=Data(:,3);
figure(3)
plot(t,y1,t,y2)
axis ([1 5 2 8])
```

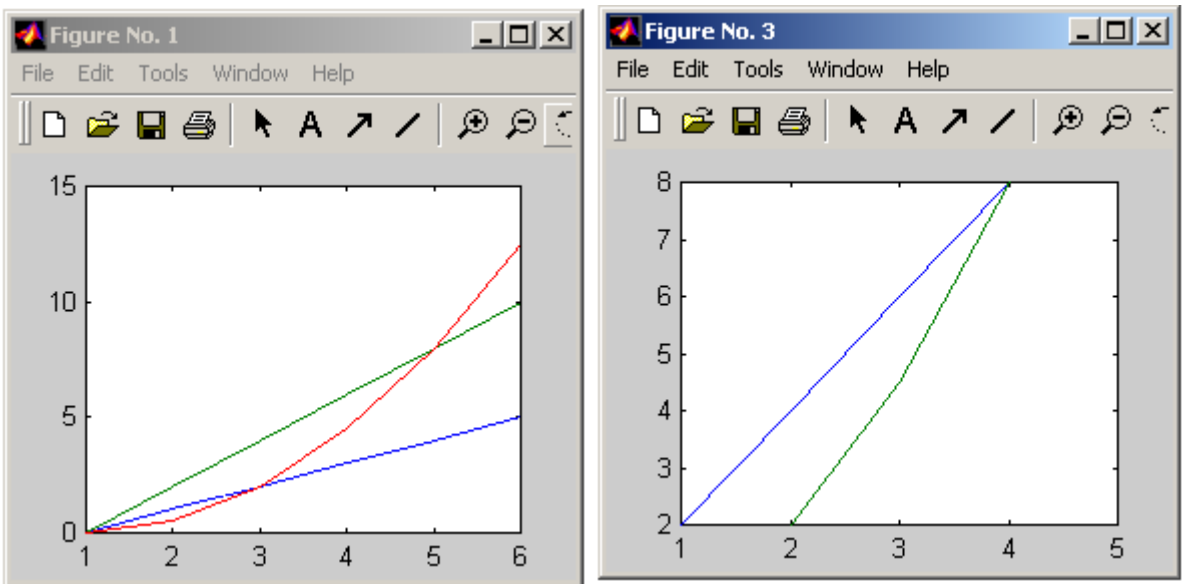


Bild 2.3 Graphische Darstellung mit der Anweisung **figure** und **axis**

Es ist auch möglich das Grafikfenster zu teilen. Durch den Befehl **subplot** wird das Graphikfenster unterteilt in Subgraphen.

Funktion subplot **bestimmt, welche Unterteilung für den nächsten Befehl plot belegt wird**.

Syntax der Funktion :

subplot (m,n,p)
m Zahl der Zeilenfenster
n Zahl der Spaltenfenster
p Definiert das resultierende Fenster

Die Anwendung der Funktion **subplot** wird am folgenden Beispiel demonstriert. Die Teilung des Graph-Fensters ist im Bild 5 dargestellt. Das Graph-Fenster wurde mit Hilfe des Rollmenü Edit →Copy Figure in die Word transformiert.

```
%B_Plot111.m Beispiel"subplot"
plot(Data)
t=Data(:,1);y1=Data(:,2);y2=Data(:,3);
subplot(2,1,1); plot(t,y1)
subplot(2,2,3); plot(t,y2)
xis([0 5 0 12.5])
subplot(2,2,4); plot(t,y2,t,y1)
```

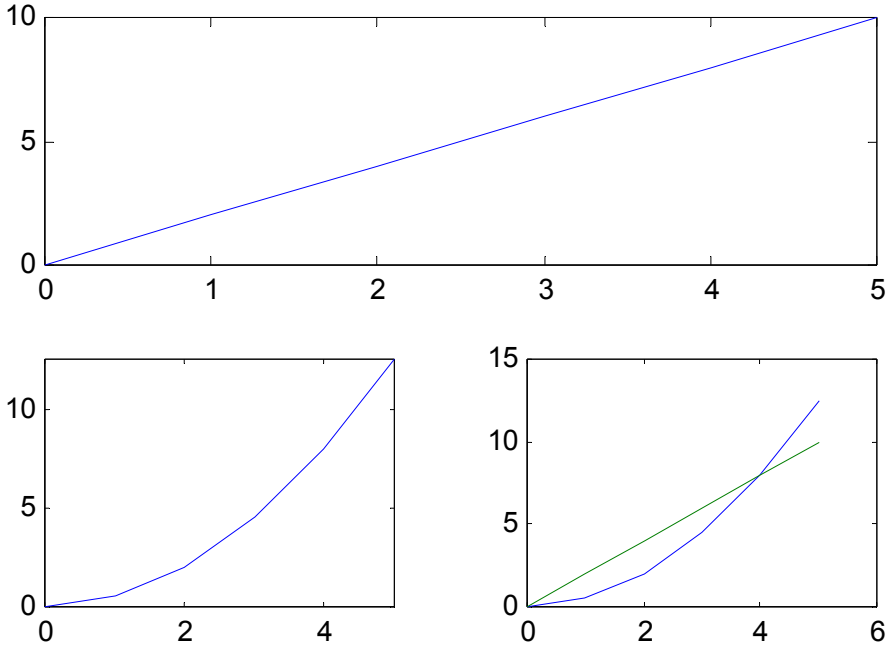


Bild 2.4. Anwendung der Funktion Subplot

2.2.2 Beschriftung der Graphen

Der Graph und die Achsen können einfach beschriftet werden. Die Anwendungen der Beschriftungsbefehle werden am folgenden Beispiel gezeigt. Die Graph- und Achsenbeschriftung sind auf den Bildern 2.3 und 2.4 zu sehen.

```
%Beispiel B_Plot2 - Funktion:  title, ylabel, xlabel
t=Data(:,1);y1=Data(:,2);y2=Data(:,3);
plot(t,y1,t,y2)
title('Beispiel plot2')
ylabel('v[m/s],y[m]')
xlabel('t[sec]')
```

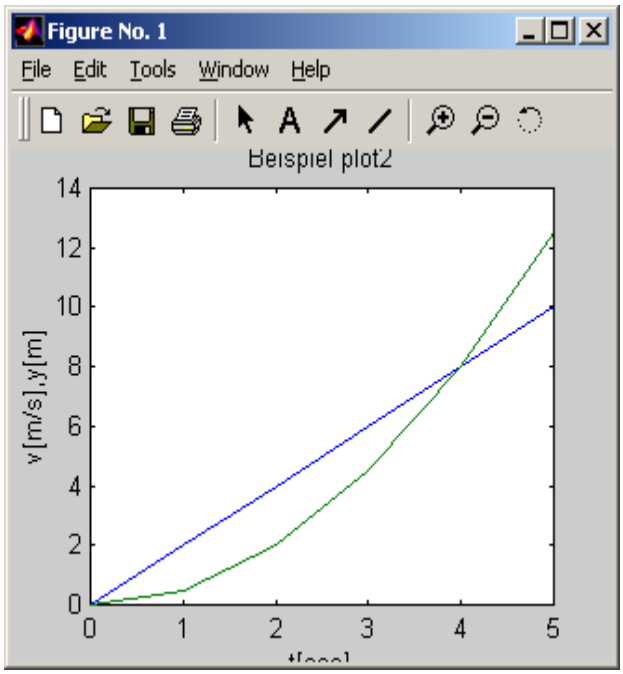


Bild 2.3 Graph-Beschriftungen

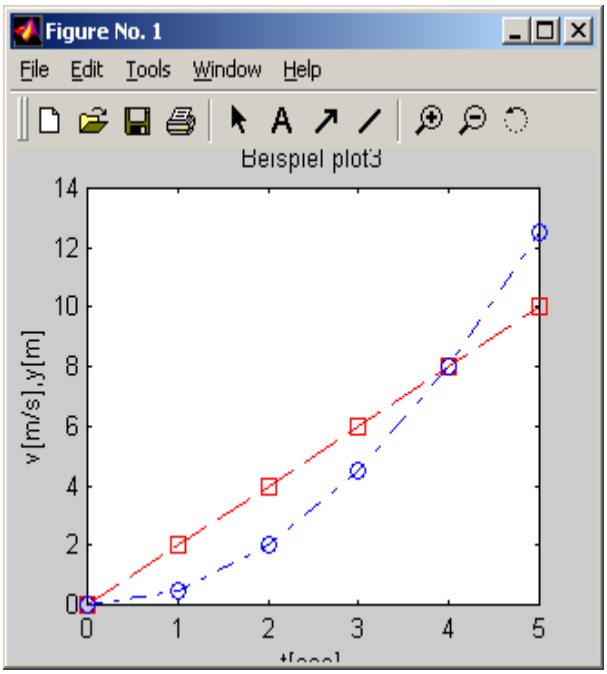


Bild 2.4 Graph-Markierung, Linetype

2.2.3 Spezifikation der Graphlinien

Die **plot**- Graphlinien können noch genauer spezifiziert werden, wenn man definiert:

- **Linietyp**

Die Linietypen sind: für Volllinie (solid) "**-**", für gestrichelt (dashed) "**--**", für punktiert (dotted) "**:**", für strichpunktiert (dash-dot) "**-.**".

- **Liniestärke** - ist definiert durch die anzahl der Punkte
  Die Anweisung:' LineWidth','**Zahl der Punkte**',

- **Liniefarbe**

Farben der Linien sind durch Farbzeichen definiert.: rot (red) '**r**', grün (green) '**g**', blau (blue) '**b**', zyan (cyan) '**c**', magneta (magneta) '**m**', gelb (yellow) '**y**', schwarz (black) '**k**', weiß (white) '**w**'.

- **Typ der Markierung der Punkte**
  Markierung der Punkte: '+', '**o**', '*', '**.**'(Punkt),'**x**'(cross), '**s**'(Quadrat), '**d**', (Raute), '^', '>', '<', '**p**', (five-point star), '**h**', (six-point star).

Ein einfaches Beispiel dokumentiert die Anwendung von der Spezifikation der Graph-, Farb- und Markierungspezifikation. Die Graphen sind auf dem Bild 2.4 dargestellt.

```
%Beispiel B_Plot3
%Beispiel für Linientyp, Farbe und Typ der Markierung
t=Data(:,1);y1=Data(:,2);y2=Data(:,3);
plot(t,y1,'--rs',t,y2,'-.bo')
title('Beispiel plot3')
ylabel('v[m/s],y[m]')
xlabel('t[sec]')
```

- **Markierungsgröße** - ist definiert durch die Zahl der Punkten
  Die Anweisung:' MarkerSize','**Zahl der Punkten**',

- **Markierungsfüllung**
  Die Anweisung:' MarkerFaceColor','**k**',

Ein Beispiel dokumentiert die Anwendung von Markierungsgrößen und Markierungsfüllung. Die Graphen sind auf dem Bild 2.5 dargestellt.

```
%Beispiel B_Plot5
%Beispiel für Markierungsgröße und Markierugsfühlung
t=Data(:,1);y1=Data(:,2);y2=Data(:,3);
plot(t,y1,'--rs','LineWidth',2,'MarkerEdgeColor','k',...
                 'MarkerFaceColor','g','MarkerSize',10)
hold on
plot(t,y2,'--bo','LineWidth',2,'MarkerEdgeColor','k',...
                 'MarkerFaceColor','y','MarkerSize',20)
title('Beispiel plot1')
ylabel('v[m/s],y[m]')
xlabel('t[sec]')
```

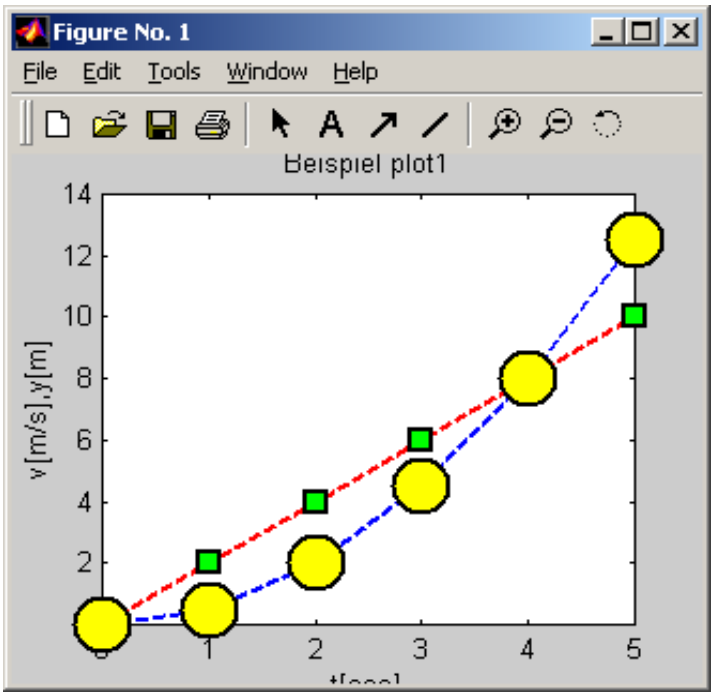


Bild 2.5 Graph Markierungsgröße, Markierungsfüllung

2.2.4 Dreidimensionale Darstellung

Die dreidimensionale Darstellung wirdin diesem Lehrmaterial anhand der Funktionen für die Gitter- und Höhenlinien-Darstellung beschrieben. Es wird nur ein Überblick vermitteln und deswegen wird die Beschreibung nur an einem Beispiel demonstriert. Folgende Funktion ist dreidimensional darzustellen :

$$z = 10 \cdot x \cdot y \cdot \exp(-x^2 - y^2)$$

Eine Gitterdarstellung wird mit der Funktion **mesh** durchgeführt und eine Höhenlinien-Darstellung wird durch die Funktion **contour** gebildet. Für beide Funktionen ist eine Aufbereitung der darzustellenden Daten nötig. Dazu dient die Funktion **meshgrid**, die generiert die notwendigen Matrizen für den 3-D-Plot.
Die Syntax der Funktion:

[X,Y] = meshgrid (x, y)
[X,Y] = meshgrid (x)
[X,Y,Z] = meshgrid (x, y,Y)
x,y Vektoren, die enthalten die Werte der unabhängigen Variablen
X,Z Matrizen für 3-D Darstellung

Durch die MATLAB –Anweisung

```
>> Z=(10.*(X).*(Y)).*exp(-X.^2-Y.^2)
```

erhält man eine Gitter-Darstellung der Funktion $z = 10 \cdot x \cdot y \cdot \exp(-x^2 - y^2)$.

Gitter-Darstellung
Die Gitter-Darstellung wird durch die Funktion **mesh** gebildet.

Funktion mesh **Bildet eine 3-D Gitter-Darstellung**

Syntax der Funktion ist es:

mesh (X,Y,Z)
mesh (Z)
meshc (X,Y,Z)
meshz (X,Y,Z)
X,Z Matrizen für 3-D Darstellung

Wie die Befehle benutzt werden und ihre Eigenschaften werden an dem folgendem Beispiel demonstriert. Bild 2.6 zeigt eine Gitterdarstellung.

```
%Beispiel B_Plot6
%Beispiel für die Funktionen meshgrid, mesh, meshc
[X,Y]=meshgrid(-2:.2:2);
Z=(10.*(X).*(Y)).*exp(-X.^2-Y.^2);
subplot(2,2,1); mesh(X,Y,Z);
subplot(2,2,2); meshc(X,Y,Z);
subplot(2,2,3); meshz(X,Y,Z);
subplot(2,2,4); mesh(Z);
```

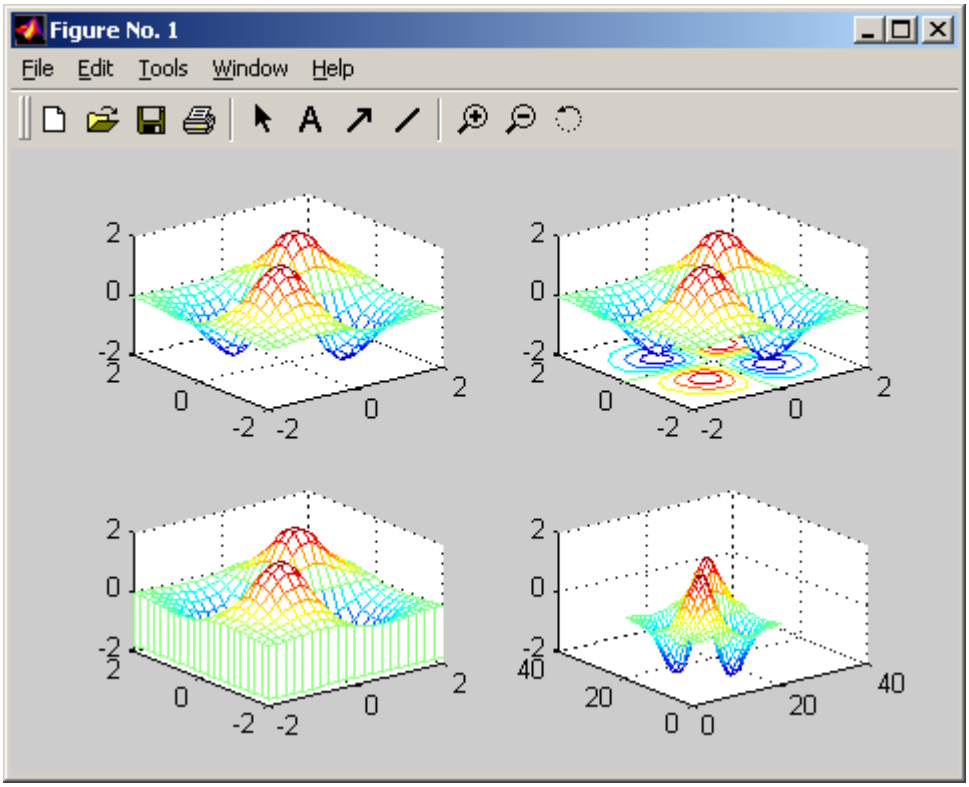


Bild 2.6: Gitter-Darstellung mit der Funktion **mesh**

Höhenlinien-Darstellung
Die Höhenlinien-Darstellung wird durch die Funktion **contour** gebildet.

Funktion contour **Bildet eine 3-D Höhenlinien-Darstellung**

Syntax der Funktion ist es:

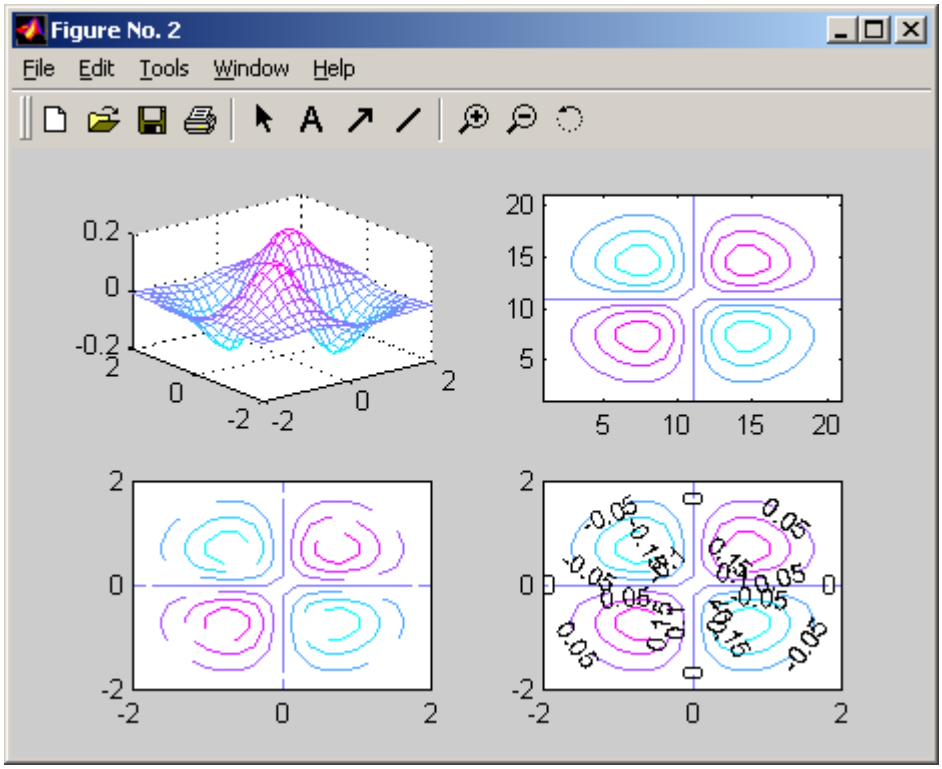
contour (Z)
contour (Z, n)
[C,h]=contour(Z)
Z Matrix für 3-D Darstellung

contour (Z) **Bildet eine 3-D Höhenlinien-Darstellung**
contour (Z,n) **Bildet eine 3-D Höhenlinien-Darstellung mit n Schnitten**
[C,h]=contour(Z) **Bildet eine contour Matrix C. Durch Funktion clabel (C,h) wird die Höhenlinien-Beschreibung vorbereitet.**

Wie die Befehle benutzt werden können, wird auf dem folgenden Beispiel demonstriert.

```
%Beispiel B_Plot7
%Beispiel für die Funktionen meshgrid, mesh, contour
[X,Y]=meshgrid(-2:.2:2);
Z=X.*Y.*exp(-X.^2-Y.^2);
[C,h]=contour(X,Y,Z);
clabel(C,h)
subplot(2,2,1); mesh(X,Y,Z);
subplot(2,2,2); contour(Z);
subplot(2,2,3); contour(Z,20);
[C,h]=contour(X,Y,Z);
subplot(2,2,4);clabel(C,h)
colormap cool
figure(2)
[C,h]=contour(X,Y,Z);
clabel(C,h)
colormap cool
```

Im Bild 2.7 ist die Graphik als Ergebnis der Anwendung der Funktion **contour** zu sehen. Im ersten Teil (Subplot (2,2,1)) ist der Graph mit der Funktion **mesh** dargestellt. Das Subplot (2,2,2) ist mit Hilfe der Funktion **contour(Z)** durchgeführt. Das Subplot (2,2,3) wird mit dem Parameter **n=20** und das Subplot (2,2,4) mit der Beschreibung dargestellt.



2.3 MATLAB ARBEITSSPEICHER (WORKSPACE)

Anweisungen für Arbeit mit Arbeitsspeicher (Workspace)

1)	**dir**	Schreibt den Inhalt des aktuellen Verzeichnisses
2)	**who**	Ausgabe einer Liste mit den aktuellen Variablen im Arbeitsspeicher.
3)	**whos**	Ausgabe einer erweiterten Variabelliste des Arbeitsspeichers
4)	**workspace**	Öffnet das Workspace Browser Fenster.
5)	**edit**	Öffnet das MATLAB Editor/Debugger Fenster
6)	**edit name**	Öffnet das MATLAB Editor/Debugger Fenster mit der angegebenen Datei
7)	**clear all**	Diese Anweisung löscht alle Daten aus dem Arbeitsspeicher
8)	**clear name**	Löscht die Veränderliche unter dem Namen
9)	**clc**	Löscht Bildschirm
10)	**size**	Bestimmt den Typ einer Matrix (n, m)
11)	**length**	Bestimmt die Vektorlänge
12)	**disp**	Ausgabe von Text oder einer Matrix **in M-Datei**
13)	**pause(n)**	Hält die Bearbeitung einer Anweisungsfolge auf **n** Sekunden an. Wenn **n** nicht eingegeben ist, läuft die Fortsetzung nach Drücken beliebiger Taste
14)	**type**	Inhaltsausgabe einer M-Datei (mit der Zufix .m)

2.4 DATEI KOMMANDOS

Für die Arbeit mit Dateien gebrauchen wir die Befehle save a load .

1) **Funktion save** ***Speichert eine Datei.m im Arbeitsspeicher auf Hard Disc***

Syntax der Funktion

```
save
save filename
save filename variables
save filename options
save filename variables options
```

a) save überträgt alle Variablen im Arbeitsspeicher in die Datei matlab.mat in das aktuelle Verzeichnis

Beispiel 1

Im Bild 2.8 ist zu sehen, welche Variable aus dem Workspace mit Hilfe der Anweisung **save** in die Datei gespeichert werden.
Es geht um die Vektoren **A1, A2, B1, B2, t** und **y,** die als Variable gespeichert werden Auch die zwei LTI-Objekte **s, s1** müssen übertragen werden.

```
clear all
A1=[1 3 2 1];A2=[2 1];B1=2;B2=[-1 2];

s=tf(B1,A1)                  » s1=tf(B2,A1)

Transfer function:           Transfer function:
          2                          -s + 2
---------------------        ---------------------
s^3 + 3 s^2 + 2 s + 1        s^3 + 3 s^2 + 2 s + 1

step(s,s1)

 » [y,t]=step(s1);
 » save

 Saving to: matlab.mat
```

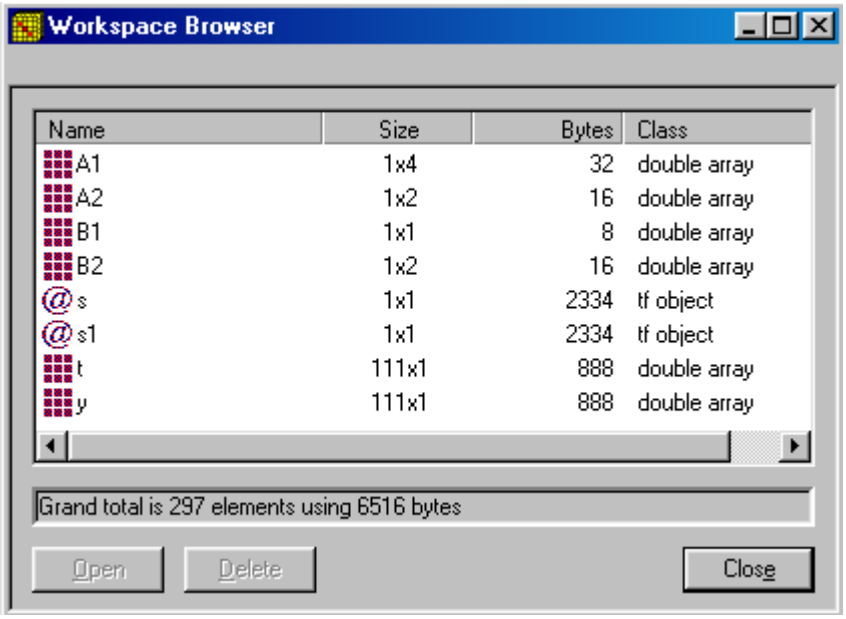


Bild 2.8 Workspace Browser

b) save filename - Überträgt alle Variablen im Arbeitsspeicher in die Datei filename.mat in das aktuelle Verzeichnis

Beispiel 2

Überträgt den Arbeitsspeicher („Workspace“) als Datei unter den Namen **24FEB** in das aktuelle Verzeichnis**.**

```
› save 24FEB
```

c) save filename variables – Überträgt nur die Variable im Arbeitsspeicher, die in der Liste **variables** angegeben sind, unter den Dateinamen filename.mat in das aktuelle Verzeichnis

Beispiel 3

Überträgt Veränderlichen *y, t, A1, B1* aus dem Arbeitsspeicher („Workspace“) als Datei unter den Namen **24FEB2** in das aktuelle Verzeichnis

```
save 24FEB2 y t A1 B1
```

d) save filename options – Überträgt wie in **b)** , aber definiert unter **options** das Format, in dem die Zahlen gespeichert werden. Siehe Tab.4

Options	Dateiformat
-ascii	Speichert im 8-zähligen ASCII Code
-ascii -double	Speichert im 16-zähligen ASCII Code
-ascii -tabs	Speichert im 8-zähligen ASCII Code, getrennt durch Tabulatorzeichen
-ascii –double -tabs	Speichert im 16-zähligen ASCII Code, getrennt durch Tabulatorzeichen
-V4	Speichert im Format MATLAB 4.0
-append	Speichert am Ende der Datei

Tab.4

Beispiel 4

Übertragung der Variablen *y, t* aus dem Arbeitsspeicher unter den Dateinamen **24FEB3**.mat in das aktuelle Verzeichnis im Format ascii und double **.**

```
save 24FEB3 y t -ascii -double
```

2) **Funktion load** Übernahme der in Datei.mat gespeicherten Variablen in den Arbeitsspeicher

Syntax der Funktion

```
load
load filename
load ('filename')
load filename.ext
load filename -ascii
load filename –mat
S=load( … )
```

a) load - Übernahme der in Datei „**matlab.mat**“ gespeicherten Variablen in den Arbeitsspeicher

Beispiel 5

```
» load

Loading from: matlab.mat
```

Im Bild 2.9 ist in der Workspace Browser – Fenster zu sehen, dass durch die Anweisung **matlab.mat** die entsprechenden Variablen in die Workspace übertragen wurden.

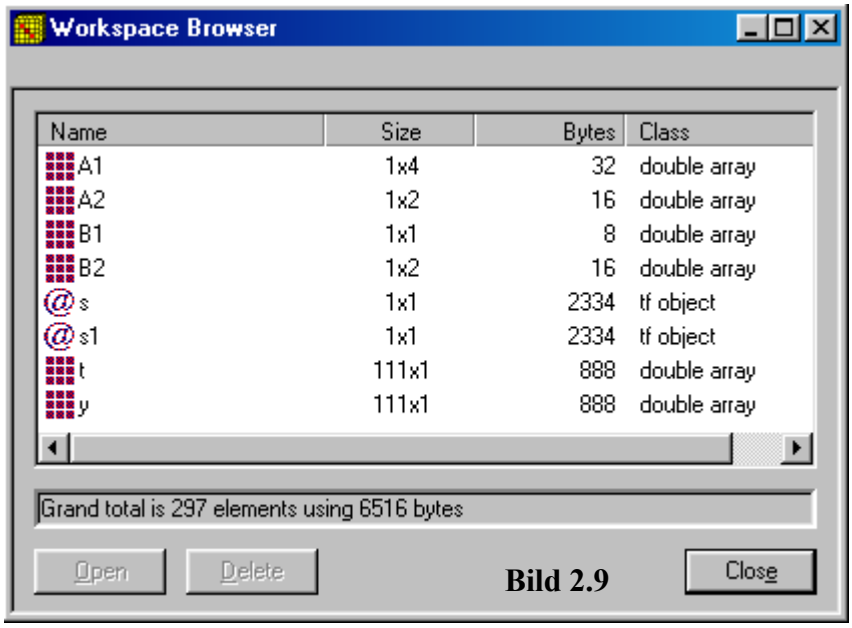


Bild 2.9

b) load filename -Übernahme der in Datei „**filename.mat“** gespeicherten Variablen in

```
» A1
  A1 =  1     3     2     1
» A2
  A2 =  2     1
» B1
  B1 =  2
» B2
  B2 = -1     2
```

```
» s

Transfer function:
          2
---------------------
s^3 + 3 s^2 + 2 s + 1
```

```
» s1

Transfer function:
       -s + 2
---------------------
s^3 + 3 s^2 + 2 s + 1
```

den Arbeitsspeicher. Die Datei nach der Dateinameerweiterung muss in MATLAB gebildet worden sein.

Beispiel 6

Als Beispiel kann die Datei aus dem Beispiel 2 und 3 dienen.

```
» load 24FEB
```

```
load 24FEB2 y t A1 B1
```

Auf dem Bild 2.10 "Workspace Browser" sind Veränderlichen mit dem entsprechenden Format dargestellt.

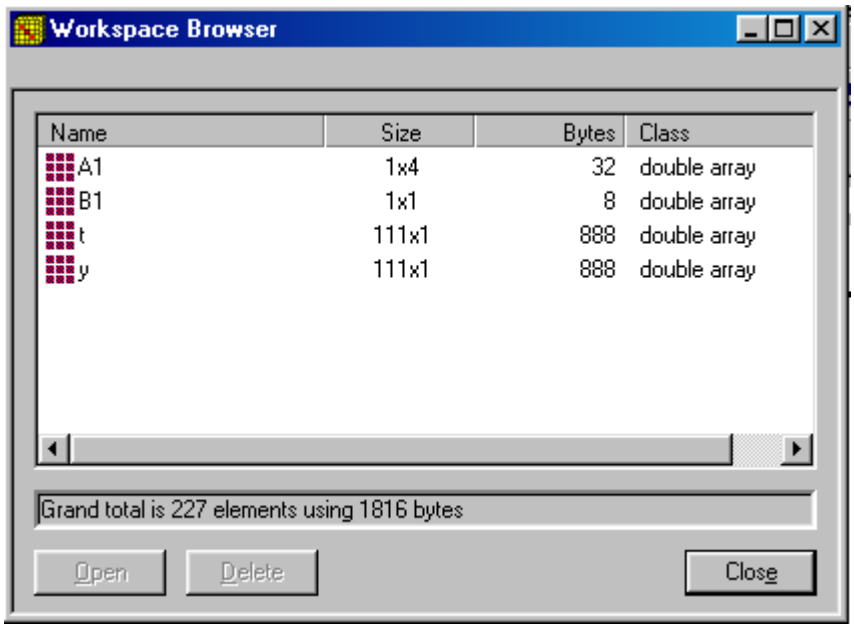


Bild 2.10

c) load ('filename')-Übernahme der in Datei '**filename.mat'** gespeicherten Variablen in den Arbeitsspeicher. Dabei ist es nötig, den vollständigen Pfad (Weg, Path) zu der Datei anzugeben.

Beispiel 7

Als Beispiel wird die Übernahme einer Messung, die unter den Dateinamen **tyu1,** in ORDNER gespeichert wird. Die Zeitverläufe des Eingangs und Ausgangs sind zu zeichnen. Die Messung liegt in einer Matrix, die als Variable **dat1** bezeichnet ist, vor. In der ersten Matrixzeile ist die Zeit, in der zweiten Zeile der gemessene Ausgang und in der dritten Zeile der Eingang abgebildet. Der vollständige Pfad (Weg) zu der Datei ist:

C:\Dokumenty\PEDAG\Vyuka\TAR\TAR I\MATL\Ident \tyu1

Das Bild 2.11 zeigt den Fall, wenn man aus der Datei dat1 nur Vektor **t** und **y** separiert.
Die Anweisung hat die folgende Form

```
» load('C:\Dokumenty\PEDAG\Vyuka\TAR\TAR I\MATL\Ident\tyu1')
```

Die Plot-Anweisung verlangt folgende Berechnungen und ist auf dem Bild 2.12 dargestellt:

```
» yG=dat1(2,:)';
» uG=dat1(3,:)';
» tG=dat1(1,:)';
» plot(tG,uG,tG,yG)
»
```

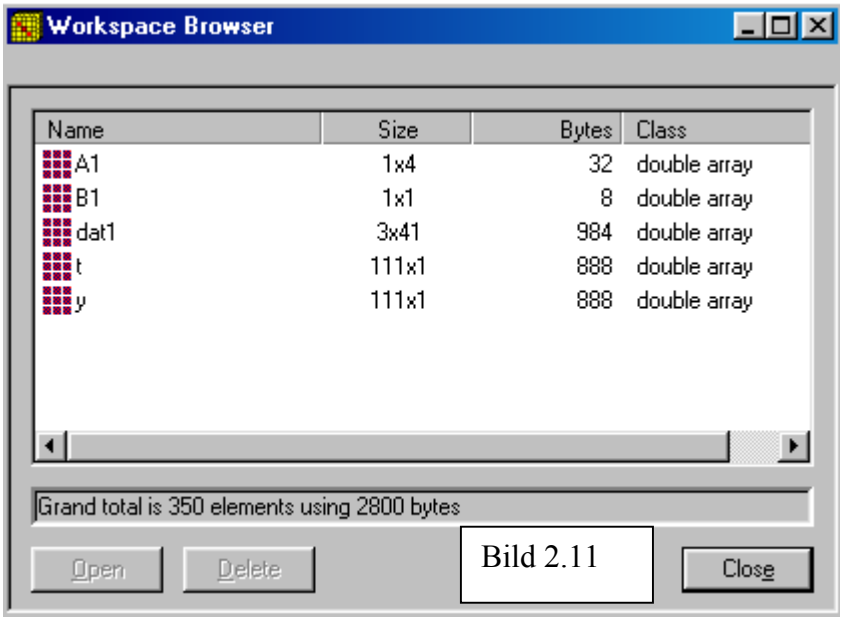


Bild 2.11

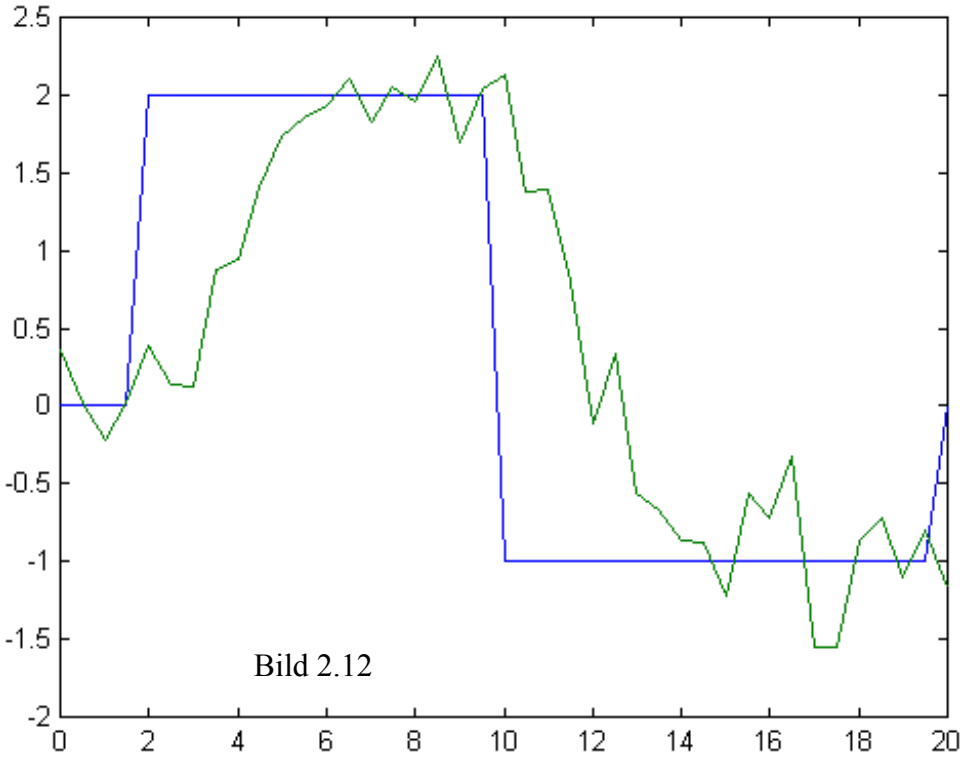


Bild 2.12

d) load filename.ext - Übernahme der in Datei „**filename.ext“** gespeicherten Variablen in den Arbeitsspeicher unter denselben Namen ohne der Dateinameerweiterung **ext.**

e) load filename –ascii - Übernahme der in Datei **filename-ascii** gespeicherten

Variablen in den Arbeitsspeicher unter denselben Namen.

f) load filename –mat - Übernahme der in Datei **filename-mat** gespeicherten Variablen in den Arbeitsspeicher unter denselben Namen

2.5 MATLAB ANWEISUNGEN FÜR PROGRAMMSTEUERUNG

Bei derProgrammierung werden Anweisungen für die Programmverzweigung und Programmwiederholung benutzt.

1) **Funktion if** **Bedingte Bearbeitung der nachfolgenden Anweisungen bis „end“**

Syntax der Funktion

```
if variable = expression
        statement
end
```

oder

```
if  expression1
        statements
elseif expression2
        statements
else
        statements
end
```

Anwendung der Anweisung **if** und Dateierfassung mit dem Zufix **.m**

2) **Funktion for** **Wiederholte Bearbeitung der nachfolgenden Anweisung bis „end“**

Syntax der Funktion

```
for variable = expression
        statement
          …
        statement
end
```

Beispiel 1 Anwendungsbeispiel der Anweisung **for** in MATLAB Editor. Die Datei hat den Namen PR2.m. Die Berechnung wird gestartet im Anwenderfenster (Command Window) durch die Anweisung **PR2** oder im Rollmenü Tools durch **Run.** Das Programm ist auf dem Bild 2.12 dargestellt und die Matrix A ist als Ergebnis des Programms **PR2** dargestellt.

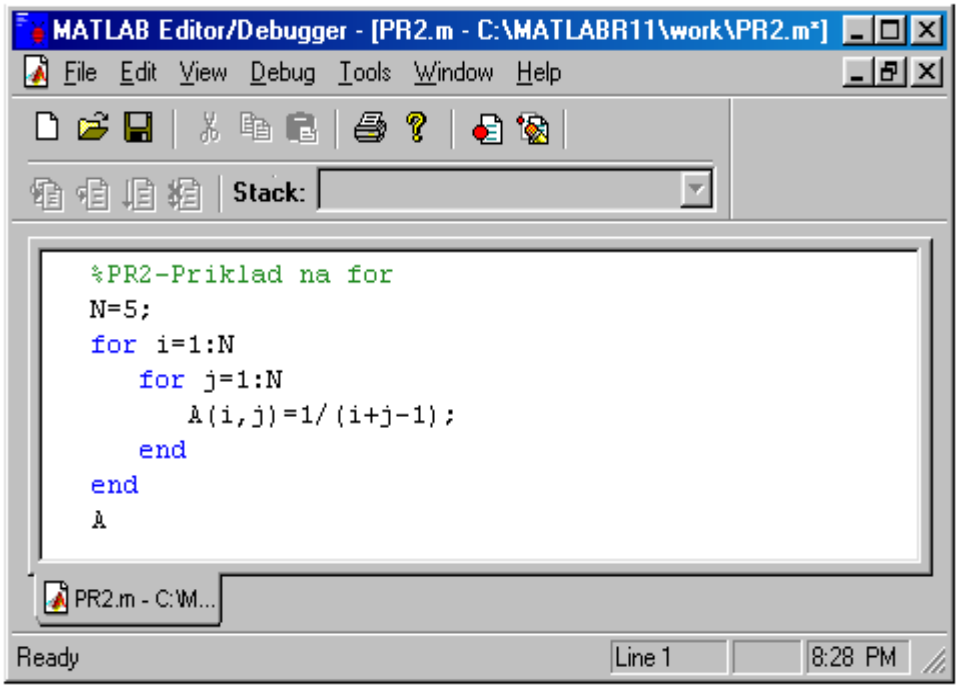


Bild 2.12

```
» PR2

A =

    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
```

2.6 SIMULATIONSAUFRUF EINES MODELLS AUS EINEM MATLAB - PROGRAMM

Aus einem MATLAB – Programm, einer Funktionsroutine oder direkt aus dem Kommandofenster kann man eine Simulation eines Modells aus der MATLAB – Umgebung starten. Das Modell kann sein:

1) ein SIMULNK- Modell, oder
2) ein LTI Modell.

2.6.1 Aufruf einer Simulation von einem SIMULINK – Modell

Die Simulation eines SIMULINK – Modells erfolgt mit Hilfe der Funktion **sim.**:

Funktion sim **ruft eine Simulation eines SIMULINK- Modells auf.**

ie Syntax dieser Funktion ist

```
              sim( 'modell')
              sim( 'modell', timespan)
              sim( 'modell', timespan, options)
              sim( 'modell', timespan, options, ut)
    [t, x, y] = sim( 'modell')
    [t, x, y] = sim( 'modell', timespan)
    [t, x, y] = sim( 'modell', timespan, options)
    [t, x, y] = sim( 'Modell', timespan, options, ut)
  modell. . . Name der Datei des Modells
timespan  . . . Simulationszeit
  options  . . . Simulationsparameter
       ut    . . . Externer Eingangsignale
       t     . . . Zeitvektor
       x     . . . Matrix der Zustandsvektoren
       y     . . . Matrix der Ausgangssignale
```

Beispiel 2

Wir haben zwei SIMULNK Programme SK1, SK2 (Bilder2.13a,c). Als Variable „**Sprung**" ist bei den beiden Programmen der Eingangssprung bezeichnet. Das SIMULINK Programm SK2 hat die Simulationszeit als Variable „**Tsim**" bezeichnet. Die Anweisungen sind auf dem Bild 2.13 , der Zeitverlauf ist auf den Bildern 2.13b,d dargestellt. Simulationsparameter sind auf dem Bild 2.13e,f zu entnehmen.

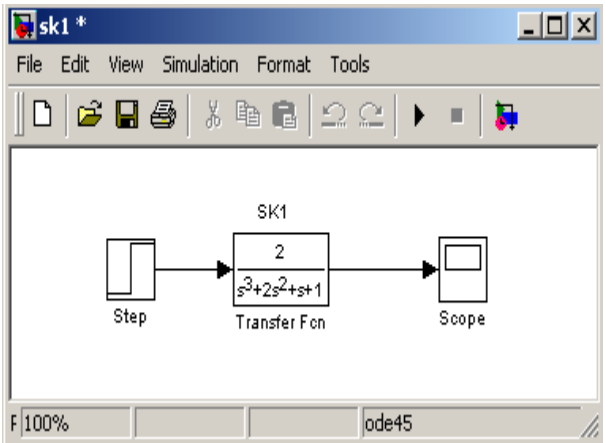
```
» Sprung=2

Sprung =

     2
```

```
» sim('SK1')
» sim('SK2',10)
» sim('SK2',20)
```

Bild 2.13 Anweisung **sim**



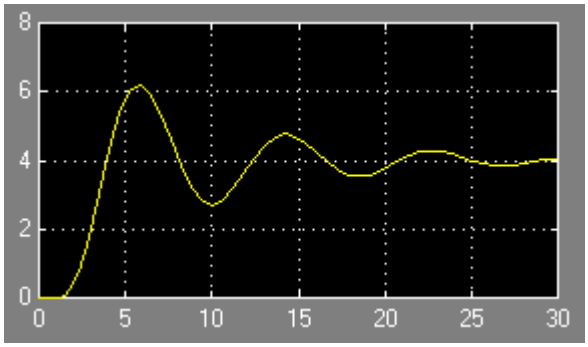


Bild 2.13a,b SIMULINK Programm SK1, Zeitverlauf der Simulation.

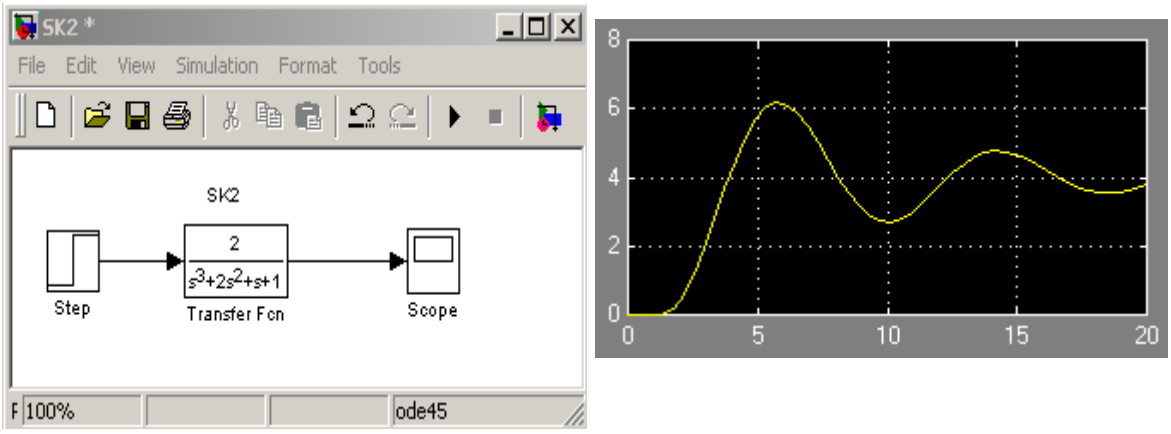


Bild 2.13c,d SIMULINK Programm SK2, Zeitverlauf der Simulation.

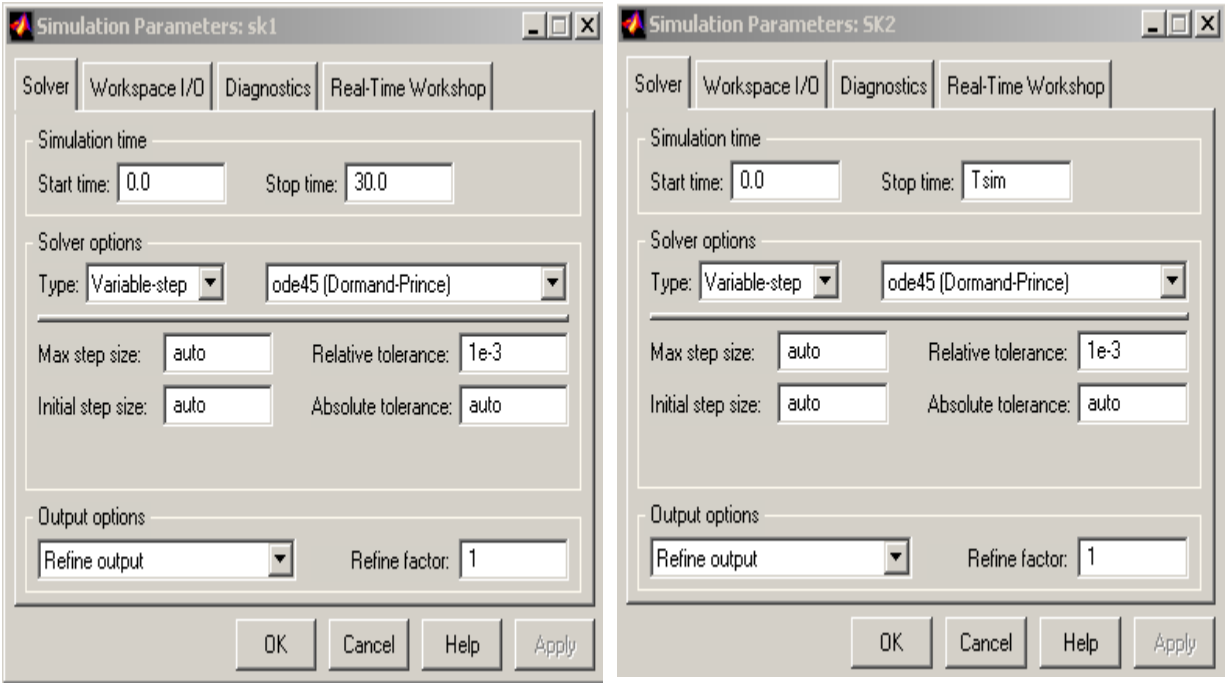


Bild 2.13e,f Parametrisierungsfenster des Programms SK1, SK2

2.6.2 Aufruf einer Simulation eines LTI – Modell

LTI Modell (Linear Time Invariant Model) ist ein lineares zeitinvariantes Modell. Das Modell kann ganz allgemein ein Mehrgrößensystem approximieren und ist kontinuierlich oder diskret. In der technischen Praxis sind diese Modelle als Übertragungsfunktionen bekannt, siehe Bild 2.14. In dem MATLAB werden diese LTI Modelle überwiegend mit Hilfe der MATLAB Funktionen **tf, ss, zpk** gebildet. Über diese Funktionen wird in dem Teil „MATLAB Anwendungen in der Regelungstechnik“ detailliert gesprochen. Es ist leicht zu verstehen, das für eine Simulation von einem LTI Modell, der Zeitvektor t, der Eingang u und die Anfangsbedienun-

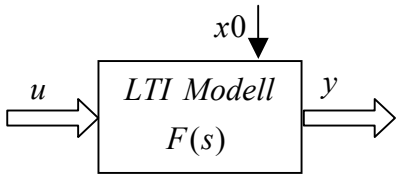


Bild 2.14 Ein- und Ausgänge eines LTI Modells

gen $x0$ einzugeben sind. Aufruf der Simulation wird durch die Funktion **lsim** durchgeführt.

Funktion lsim — **Ruft die Simulation des Ausgangs eines LTI Modells und stellt graphisch die Ausgänge dar.**

Die Syntax dieser Funktion ist

```
            lsim( sys, u, t)
            lsim( sys, u, t, x0)

            lsim( sys1, sys2, sys3, ... u, t)
            lsim( sys1, sys2, sys3, ... u, t,x0)

[t, x, y] =  lsim( sys, u, t, x0)

   sys  . . .  Name des LTI Modells
    t   . . .  Zeitvektor: t = 0:dt:Tsim
    u   . . .  Externer Eingangsignale, eine Matrix, die muss so
               viel Zeilen haben, wie viel Zeilen hat der Zeitvektor t
   x0 . . .  Vektor der Anfangsbedienungen an Zustandsgrößen

    y     . . . Matrix der Ausgangssignale
```

Beispiel 3

Eine Anwendung der Anweisung **lsim** ist in diesem Beispiel vorgestellt. Es werden die Ausgänge von zwei LTI- Objekte mit dem Eingang **u** und Zeit **t** simuliert. Das Ergebnis ist im Bild 2.15 dargestellt.

```
» s1

Transfer function:
          2
--------------------
s^3 + 2 s^2 + s + 1

» s2

Transfer function:
     -2 s + 2
---------------
4 s^2 + 4 s + 1

» t=0:0.05:100;
» length(t)

ans =

        2001

» u(1:1:1000)=1;
» u(1001:1:2001)=1.5;

» lsim(s1,s2,u,t)
```

3 LITERATUR

[1] GRACE, A.-LAUB, J.A-LITTLE, J.N.-THOMPSON, C.M.: *Control System Toolbox. For Use with MATLAB. User's Guide.* The Math Works, Inc.1995

[2] BODE, H.: *MATLAB in der Regelungstechnik. Analyse linearer Systeme.* B. G. Teubner Stuttgart, Leipzig, 1998. ISBN-3-519-06252-6

[3] HOFFMAN, J.: *MATLAB und SIMULINK. Beispielorientierte Einführung in die Simulation dynamischer Systeme.* Addison - Wesley –Longman,1998. ISBN-3-8273-1077-6

[4] *The Student Edition of MATLAB. High /Performance Numeric Computation and Visualization Software. Version 4.User'Guide*. The Math Works, Inc.1995

Dieses Spiegelprojekt wird teilweise aus Finanzmittel der Europäischer Union gefördert.