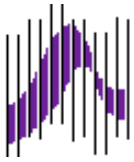
Hochschule Ravensburg-Weingarten	ALLGEMEIN	03/2007	
Labor Regelungstechnik	**Einführung in das Programm MATLAB / SIMULINK**		

Inhalt

1. Einleitung 2

2. Start der Arbeit mit MATLAB 3

2.1 Interaktive Eingabe im „Command Window“ 3

2.2 MATLAB Hilfe und Beschreibungen zu MATLAB-Befehlen 3

3. Zahlen, Vektoren und Matrizen 4

3.1 Zahlenformate 4

3.2 Definition von Variablen als Skalare, Vektoren oder Matrizen 4

4. Einfache Arithmetik 7

4.1 Grundrechenarten für Skalare (d.h. 1×1 Matrizen) 7

4.2 Trigonometrische Funktionen 7

4.3 Elementare mathematische Funktionen 7

4.4 Relationale Operatoren 7

4.5 Besonderheiten beim Rechnen mit Vektoren und Matrizen 7

4.6 Grafische Darstellungen von Funktionen 9

5. Programmieren in MATLAB 12

5.1 Programmcode generieren und abspeichern mit Hilfe des MATLAB Editors (M-Files) 12

5.2 Kontrollstrukturen 13

5.3 Funktionen in MATLAB 14

6. Einführung in die „Control Toolbox“ (spezielle Befehle & Werkzeuge für die Regelungstechnik) 15

6.1 Übertragungsfunktion Gs eines Regelkreis 15

6.2 Charakteristika einer Übertragungsfunktion 17

6.3 Grafische Darstellungsmöglichkeiten einer Übertragungsfunktion 16

6.4 Zusammenschaltung von Modellen (Signalflussplan-Algebra) 17

6.5 Eingabe eines Totzeitglieds 18

7. Einführung in SIMULINK 19

7.1 Erste Schritte in SIMULINK 19

7.2 Kurzbeschreibung der wichtigsten SIMULINK Schaltblöcke 20

7.3 Simulation eines SIMULINK-Modells 22

7.4 Tipps & Tricks für Regelkreis-Simulationen 23

7.5 Auswertung grafischer Darstellungen mit dem Scope 24

8. Grundlagen zum Reglerentwurf mit MATLAB 25

8.1 Bestimmung des Verstärkungsfaktors K_V mit Hilfe des BODE-Diagramm 25

8.2 Bestimmung des Verstärkungsfaktors K_V mit Hilfe der Wurzelortskurve (WOK) 26

8.3 „SISO Design Tool“ zur Reglerbestimmung mittels Wurzelortskurve - `rltool` 26

Anhang 1: Hilfethemen zu Matrizenoperationen: Erklärung der Operatoren und speziellen Zeichen 29

Anhang 2: Zusammenstellung der in der Regelungstechnik wichtigsten MATLAB Befehle und Funktionen 30

Anhang 3: Übungsaufgabe zum Reglerentwurf mit MATLAB 35

1. Einleitung

MATLAB / SIMULINK ist ein leistungsfähiges interaktives Programmpaket für numerische Berechnungen im Ingenieurbereich und stellt zur Modellierung und Simulation technischer Systeme sowohl an den Hochschulen als auch in der Industrie weltweit den Standard dar. Es ist daher sinnvoll, den Umgang mit dieser Software im Rahmen des Regelungstechnik Praktikums zu vermitteln. MATLAB ist ein leistungsfähiges interaktives Programmpaket für numerische Berechnungen im Ingenieurbereich, welches sich durch folgende Besonderheiten auszeichnet:

- Die interaktive Bedienung gestaltet sich sehr einfach mit Hilfe einer Interpretersprache im MATLAB Befehlsfenster („Command Window").
- Alternativ oder in Ergänzung zur interaktiven Bedienung können MATLAB-Befehlsfolgen als Batchprogramme, so genannte M-Files, ablaufen.
- Für spezielle Bereiche der Ingenieurwissenschaften, z. B. der Modellierung und Simulation, bietet MATLAB so genannte Toolboxen an. Diese Toolboxen selbst sind vorgefertigte M-Files, die dann für das entsprechende Thema zusätzlich zu den normalen MATLAB-Befehlen spezifische Befehle anbieten. Eine spezielle Toolbox ist z.B. das Simulations-Tool SIMULINK, mit dessen Hilfe mathematische Modelle grafisch aus vorgefertigten Blöcken erstellt und dann simuliert werden können. Weitere Toolboxen wären die „Control Toolbox" mit spezifischen Befehlen für regelungstechnische Aufgaben oder die „Signal Processing Toolbox" zur Signaldatenverarbeitung.[1]
- Der Name MATLAB kommt von *"matrix laboratory"*. Damit wird die spezielle Bedeutung von Matrizen bei der Arbeit mit MATLAB klar. Die Darstellung von Datenfeldern erfolgt grundsätzlich als Matrix bzw. Vektor. Daraus ergeben sich einige Besonderheiten bei Rechnungen mit Variablen.
- Typische Anwendungen von MATLAB sind:
  - Mathematische Berechnungen;
  - Entwicklung von Algorithmen;
  - Datenerfassung und –bearbeitung;
  - Datenanalyse, -auswertung und –visualisierung;
  - Modellbildung, Simulation und Erstellen von Prototypen;
  - Wissenschaftliche und technische grafische Darstellungen;
  - Entwicklung von Anwendungen, inklusive der Gestaltung von grafischen Benutzeroberflächen.

Tipp: Die Fa. Mathworks, die MATLAB / Simulink vertreibt, bietet auch eine Studentenversion der aktuellen Version MATLAB 14 an. Mehr Informationen unter http://www.mathworks.de/.

[1] Welche Toolboxen auf dem jeweiligen PC installiert sind, sowie die jeweiligen Versionen, lässt sich über den Befehl „`ver`", einzugeben im MATLAB „Command Window", abfragen.

2. Start der Arbeit mit MATLAB

2.1 Interaktive Eingabe im „Command Window“

Alle Befehle und Variablenzuweisungen werden immer im „Command Window“, dem Befehlfenster von Matlab, hinter dem „>>“-Eingabezeichen eingegeben. Das Befehlsfenster füllt normalerweise die rechte Hälfte des Bildschirms, falls nicht, kann mit *„View“* (Taskleiste oben) → *„Desktop Layout“* → *„Default“* die in *Abb. 1* dargestellte Standardeinstellung wiederhergestellt werden.

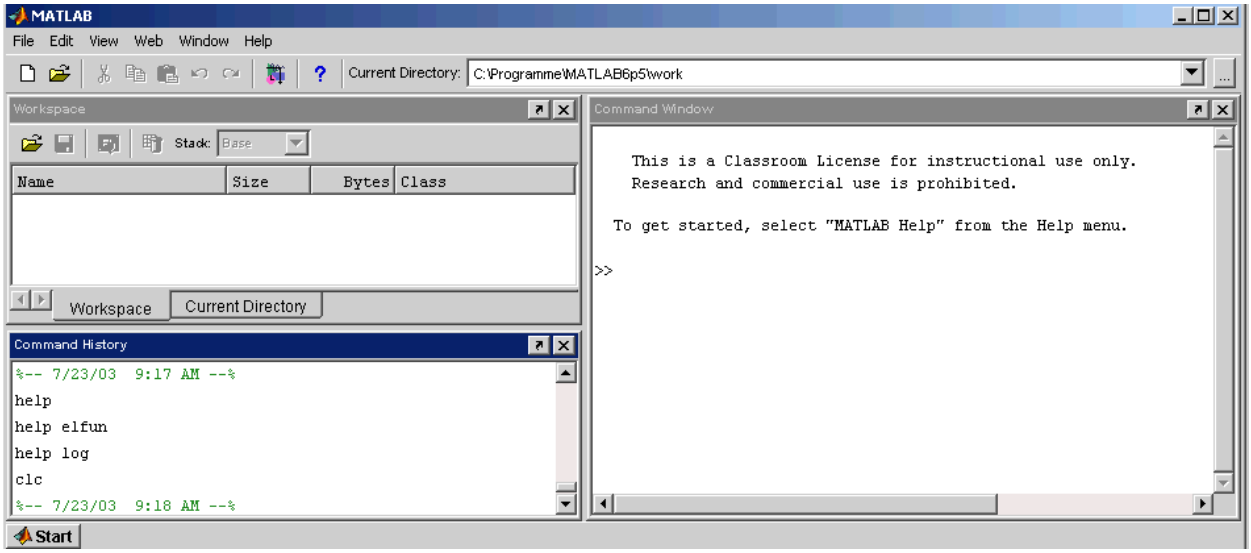


Abb. 1: MATLAB Oberfläche, rechts das „Command Window“ zur interaktiven Befehlseingabe

2.2 MATLAB Hilfe und Beschreibungen zu MATLAB-Befehlen

» help	Aufruf aller Hauptkategorien, zu denen es MATLAB-Befehle gibt.
» help elfun	Verzweigung in eine der gelisteten Hauptkategorien, z.B. matlab\elfun (elementare mathematischen Funktionen)
» help log	Aufruf der Hilfefunktion eines bestimmten Befehls, z.B. der mathem. Funktion Logarithmus

Tipp: Die Suche nach Hilfetexten über das „Command Window“ ist nur empfehlenswert, wenn der Befehl bekannt ist, zu dem die Hilfe gesucht wird. Andernfalls ist es schwierig, die richtige Hauptkategorie zu finden.
Einfacher ist es dann, über die Taskleite mit „Help“ → “Full Product Family Help“ → „Search“ die komplette MATLAB Dokumentation nach einem bestimmten Suchbegriff (in Englisch!) zu durchsuchen.
Um die Arbeit mit MATLAB zu vereinfachen, wurde eine „Übersicht der in der Regelungstechnik wichtigsten MATLAB–Befehle und –Toolboxen“ erstellt, siehe Anhang 2. Mit den Befehlen dieser Liste können alle Aufgaben zu den Versuchen im Regelungstechnik Praktikum durchgeführt werden.

3. Zahlen, Vektoren und Matrizen

Die grundlegende Datenstruktur von MATLAB ist die komplexe Matrix; alle anderen Datenstrukturen wie Vektoren und Skalare sind Spezialfälle von Matrizen. Auch für spezielle Funktionen werden viele Parameter als Vektoren dargestellt; Messwerte werden ebenfalls als Vektoren oder Matrizen abgespeichert und verarbeitet. Matrizen müssen in MATLAB in einer bestimmten Weise eingegeben werden.

3.1 Zahlenformate

Zunächst sollen die grundlegenden Arten der Zahlendarstellung in MATLAB näher beschrieben werden. Folgende Arten der Zahlendarstellung sind erlaubt:

```
3       -99       0.00001       9.6397235       1.6021E-20       6.001e23
```

Die grundlegenden Regeln bei der Zahlendarstellung werden bei diesen Beispielen schnell klar. Zu beachten ist, dass bei der Eingabe der Zehnerpotenzen vor dem Exponenten kein Leerzeichen stehen darf. Falsch ist z. B. `1.001 e-10`.

Der Zahlenbereich in MATLAB reicht von 10^{-308} bis 10^{308}. In komplexen Zahlen kann als imaginäre Zahl wahlweise i oder j verwendet werden:

```
3+4*i        3+4*j
```

MATLAB kennt verschiedene Arten von Zahlenformaten, d. h. der jeweiligen Anzahl von Ziffern einer Zahl, die auf dem Bildschirm dargestellt werden. Das Format kann mit dem `format`-Befehl geändert werden. Standard ist das Format `short`. An folgenden Beispielen soll dies verdeutlicht werden:

Zahlenformat (Befehl)	Ausgabe der Zahl 4/3	Ausgabe der Zahl 1.2345e-6
`format short`	`1.3333`	`0.0000`
`format short e`	`1.3333E+000`	`1.2345E-006`
`format long`	`1.3333333333333338`	`0.000001234500000`
`format long e`	`1.3333333333333338E+000`	`1.234500000000000E-006`
`format hex`	`3FF555555555555555`	`3EB4B6231ABFD271`

3.2 Definition von Variablen als Skalare, Vektoren oder Matrizen

Bei der Eingabe von einzelnen Werten in Vektoren oder Matrizen ist zu beachten, dass die Gesamtheit der Elemente in eckigen Klammern stehen muss.

- Variablen Werte zuordnen

» 5 ans = 5	Bei Eingabe einer beliebigen Ziffer und anschließendem Drücken der <Enter>-Taste ordnet MATLAB den Wert der temporären Variablen "**ans**" (Abkürzung für engl. "answer") zu.
» a=5 a = 5	MATLAB gibt den Inhalt der Variable normalerweise erneut auf der Oberfläche aus.
» a=5;	Mit abschließendem Semikolon ";" wird die erneute Ausgabe der Variable unterdrückt.
» a=5, b=2 a = 5 b = 2	Mit Trennung Kommata zwischen den Variablen, d.h. in der nächsten Zeile werden die Variablen erneut ausgegeben, oder ...
» a=5; b=2;	... Semikolon (erneute Ausgabe unterdrückt) können mehreren Variablen gleichzeitig in einer Zeile Werte zugeordnet werden.

- Spaltenvektoren Werte zuordnen

» z=[6;7;8];	Trennung der einzelnen Elemente des Spaltenvektors durch Semikolon, erneute Ausgabe unterdrückt.
» x=[5;7;9] x = 5 7 9	Bei Weglassen des Semikolons hinter der eckigen Klammer wird der Spaltenvektor erneut ausgegeben. Das ist gut zur Kontrolle, ungünstig, wenn es sich um einen Spaltenvektor mit 1 Mio. Elementen bzw. Zeilen handelt.

➢ Zeilenvektoren Werte zuordnen

» y=[1 2 3 4 5]	Trennung der einzelnen Elemente des Zeilenvektors durch Leerzeichen oder ...
» y=[1,2,3,4,5];	... Trennung durch Kommata. Abschließendes Semikolon unterdrückt die erneute Ausgabe des Variableninhalts.
» y=2:5 y = 2 3 4 5	Der "**:**"-Operator ist beim Umgang mit Vektoren und Matrizen sehr wichtig. Diesen Operator könnte man mit "von ...bis" übersetzen.
» y=1:2:9 y = 1 3 5 7 9	Es können auch Vektoren mit bestimmten Anfangs- und Endwerten und vorgegebenem Elementabstand erzeugt werden. Im Beispiel erzeugt der "**:**"-Operator Vektorelemente mit konstantem Abstand 2.

➢ Matrizen Werte zuordnen

» A=[1 2 3;4 5 6];	Eingabe der Zeilen, mit Trennung der einzelnen Elemente durch Leerzeichen, dann der Spalten durch Semikolon getrennt, oder ...
» A=[1,2,3;4,5,6] A = 1 2 3 4 5 6	... Eingabe der Zeilen, mit Trennung der einzelnen Elemente durch Kommata, dann der Spalten wieder durch Semikolon getrennt.
» A=[1 2 3 **4 5 6]** A = 1 2 3 4 5 6	Alternativ kann die Matrix auch in dieser Form eingegeben werden, wobei jede Zeile mit der *<Return>*-Taste abgeschlossen werden muss.
» A=[1 2 3*4 **8/4 5+6 7]** A = 1 2 12 2 11 7	Matrizen können auch als Ergebnis einer Berechnung eingegeben werden.
» A=[x z] A = 5 6 7 7 9 8	Wenn man zwei Vektoren zu einer Matrix zusammenfassen will, verwendet man [ ]. Sind `x` und `z` zwei Spaltenvektoren gleicher Länge, so wird die Matrix `A` erzeugt, deren Spalten aus `x` und `z` bestehen.
» A=[y;y-1;y*3] A = 1 3 5 7 9 0 2 4 6 8 3 9 15 21 27	Auch mehrere Zeilenvektoren gleicher Länge können über das Semikolon als Trennzeichen zu einer im Beispiel 3-reihigen Matrix verbunden werden. Dabei können auch Rechenoperationen durchgeführt werden.

➢ Herausnehmen einzelner Elemente, ganzer Spalten oder Zeilen aus einer Matrix

» x=A(2,3) x = 4	Ausgabe eines einzelnen Elements einer Matrix; wobei die erste Zahl die Zeile, die zweite Zahl die Spalte bezeichnet. Hier wird das Element aus der 3. Spalte und der 2. Zeile der Matrix `A` in x kopiert
» y=A(:,2) y = 3 2 9	Wenn aus einer Matrix eine Spalte als Spaltenvektor extrahiert werden soll, kann ebenfalls der "**:**"-Operator verwendet werden. Alle Elemente der zweiten Spalte, über alle Zeilen hinweg, werden der Variable `y`, einem Spaltenvektor, zugeordnet.
» y=A(1:2,2) y = 3 2	Es können auch bestimmte Zeilen ausgewählt werden, die in den Spaltenvektor übernommen werden. Vor dem Doppelpunkt steht die Startzeile, dahinter die letzte Zeile des neuen Vektors `y`.
» z=A(2,:) z = 0 2 4 6 8	Alle Elemente der zweiten Zeile, über alle Spalten hinweg, werden der Variable `z`, einem Zeilenvektor, zugeordnet.
» z=A(2,2:3) z = 2 4	Es können auch bestimmte Spalten ausgewählt werden, die in den Zeilenvektor übernommen werden. Vor dem Doppelpunkt steht die Startspalte, dahinter die letzte Spalte des neuen Zeilenvektors `z`.

Tipp: Die Möglichkeit, einzelne Zeilen oder Spalten aus einer Matrix herauszunehmen, ist hilfreich bei der Bearbeitung von zeitdiskreten Messdaten, die in Form von Matrizen abgespeichert sind und von denen nur eine Teilmenge für die weitere Bearbeitung oder grafische Darstellung benötigt wird.

- Erzeugen einer Null-Matrix

`» Z=zeros(2,3)`
`Z = 0 0 0`
`0 0 0`
	Der erste Index gibt wiederum die Anzahl der Zeilen, der zweite Index die Anzahl der Spalten an.

- Erzeugen einer Einheitsmatrix

`» E=eye(3)`
`E = 1 0 0`
`0 1 0`
`0 0 1`
	Eine Einheitsmatrix der Dimension 3x3 wird mit dem `eye`-Befehl erzeugt.

- Übersicht über alle erzeugten Variablen

`» who`
`our variables are:`
`A a ans x y z`
	Alle Variablen, die momentan auf der MATLAB Oberfläche gespeichert sind, werden aufgelistet.

`» whos`	Dieser Befehl gibt auch noch genauere Beschreibungen von Art und Größe der Variablen an (vergleichbar *Abb. 2*).

Tipp: *Bei MATLAB ist unbedingt auf Groß- und Kleinschreibung von Variablennamen zu achten! Wie in Abb. 2 deutlich zu sehen ist, unterscheidet MATLAB die Variablen a und A. Viele Fehlermeldungen resultieren aus einer unterschiedlichen Schreibweise oder inkonsistenter Groß- und Kleinschreibung, deshalb unbedingt einfache, logische Variablennamen wählen!*

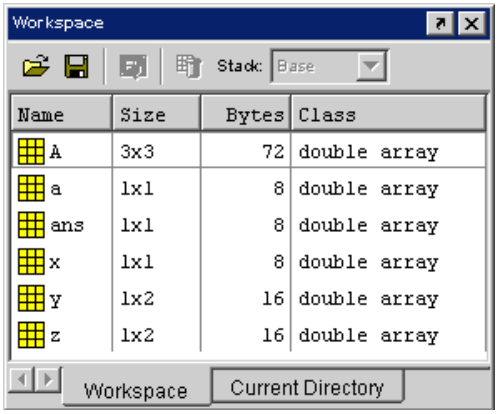


Abb. 2: ***Die Variablen inklusive der genaueren Beschreibung werden in allen MATLAB Versionen ab Version 6 auch im linken oberen Fenster, Tab "Workspace", aufgelistet.***

- Speichern von Variablen
  Variablen, die auf der MATLAB Oberfläche („Workspace") gespeichert sind, bleiben nur eine Sitzung lang erhalten. Wird MATLAB verlassen, gehen alle eingegebenen Variablen verloren. Variablen (z. B. die Matrix `A`), die erhalten bleiben sollen, müssen mit dem `save`-Befehl gespeichert werden.

`» save A`	Ohne Eingabe eines Dateinamens wird die Matrix `A` als so genanntes mat-File in der Form `A.mat` im aktuellen Verzeichnis (engl. „Current Directory") gespeichert. Dies ist standardmäßig das Verzeichnis „*work*" im MATLAB-Hauptverzeichnis.
`» save Test A x y z`	Sollen mehrere Variablen gespeichert werden, können diese durch Leerzeichen getrennt aufgelistet werden. „`Test`" ist in diesem Fall der Dateiname des erzeugten mat-Files.
`» save Test`	Wird hinter dem `save`-Befehl nur der Dateiname des mat-Files eingegeben, werden alle Variablen der MATLAB-Oberfläche gespeichert. Die komplette Oberfläche kann auch von der Taskleiste aus mit → *„File"* → *„Save Workspace as ..."* gespeichert werden. **Der Dateiname darf noch nicht als Variablenname verwendet worden sein!**

- Laden von gespeicherten Variablen

`» load test`	Eine mit dem `save` Befehl gespeicherte Variable kann mit `load Dateiname` wieder auf die Arbeitsoberfläche geladen werden. In diesem Fall spielt Groß- und Kleinschreibung keine Rolle!

4. Einfache Arithmetik

4.1 Grundrechenarten für Skalare (d.h. 1×1 Matrizen)

MATLAB befolgt die Punkt-vor-Strich-Regelung und das Setzen von Klammern kann in der bekannten Weise erfolgen.

`» c=a+b`	Addition:	+	
`» c=a-b`	Subtraktion:	-	
`» d=a*b`	Multiplikation:	*	

`» e=a/b`	Division:	/
`» f=a^b`	Potenzieren:	^

4.2 Trigonometrische Funktionen

`» sin(x)`	`» asin(x)`	`» sinh(x)`	`» asinh(x)`
`» cos(x)`	`» acos(x)`	`» cosh(x)`	`» acosh(x)`
`» tan(x)`	`» atan(x)`	`» tanh(x)`	`» atanh(x)`

Tipp: Bei Funktionen sind runde Klammern zu verwenden, bei der Zuordnung von Vektoren oder Matrizen immer eckige Klammern!

4.3 Elementare mathematische Funktionen

`» abs(x)`	Absolutwert (Betrag) von x
`» sign(x)`	Vorzeichen („signum“)
`» sqrt(x)`	Quadratwurzel
`» exp(x)`	Exponentialfunktion
`» log(x)`	Natürlicher Logarithmus
`» log10(x)`	Logarithmus zur Basis 10

`» angle(x)`	Phasenwinkel
`» real(x)`	Realteil
`» imag(x)`	Imaginärteil
`» conj(x)`	Komplex Konjugierte
`» round(x)`	Rundung auf Integerwert
`» rem(a,b)`	Modulo (Rest nach Division)

4.4 Relationale Operatoren

Weiterhin gibt es relationale Operatoren, deren Ergebnis entweder *0* oder *1* ist. Diese Operatoren sind auch auf Vektoren und Matrizen gleicher Dimension anwendbar. Das Ergebnis ist dann wieder ein Vektor oder eine Matrix.

`» x < y`	Kleiner
`» x <= y`	Kleiner gleich

`» x == y`	Gleich
`» x > y`	Größer

`» x >= y`	Größer gleich
`» x ~= y`	Ungleich

4.5 Besonderheiten beim Rechnen mit Vektoren und Matrizen

ACHTUNG: Beim Rechnen mit Vektoren und Matrizen sind unbedingt die Dimensionen zu beachten (spezielle Rechenregeln beim Umgang mit Matrizen und Vektoren)!

4.5.1 Matrix-Operationen

Weitere Hilfsthemen zu Matrix-Operatoren sind unter MATLAB z.B. mit „**`help *`**“ aufzulisten, bzw. im *Anhang 1* zu finden.

`» u=[1 2 3];v=[4 5 6];W=[8 9 0;4 5 6; 1 2 3];`	

`» u+v`
`ans = 5 7 9`
`» u-v`
	Addition / Subtraktion
	Elementweise Addition bzw. Subtraktion

`» u+[3 4]`
`??? Error using ➔ +`
`Matrix dimensions must agree.`
	Fehlermeldung, sobald versucht wird, zwei ungleich große Vektoren (Dimensionen stimmen nicht überein) miteinander zu addieren. In diesem Beispiel wird versucht, einen 1x3 mit einem 1x2 Vektor zu addieren.

» T=W'
`T =  8  4  1`
`  9  5  2`
`  0  6  3`
	Transponieren: Aus den Zeilen einer Matrix oder eines Vektors werden Spalten, bzw. umgekehrt. Falls `A` eine komplexe Matrix ist, so liefert `A'` die konjugiert komplexe Matrix zurück.

» inv(W)	Invertieren einer quadratischen Matrix.
» rank(W)	Rang einer Matrix (Anzahl der linear unabhängigen Zeilen/Spalten).
» det(W)	Determinante einer quadratischen Matrix.

» W*T
`ans = 145  77  26`
`  77  77  32`
`  26  32  14`
	Matrizenmultiplikation.

» u*v
`??? Error using ➔ *`
`Inner matrix dimensions must agree.`
	Fehlermeldung beim Multiplizieren von Vektoren gleicher Dimension.

» u'*v
`ans =  4  5  6`
`  8  10  12`
`  12  15  18`
	Skalarprodukt zweier Vektoren
	Das Ergebnis der Multiplikation eines Spaltenvektors mit einem Zeilenvektor gleicher Dimension ist eine Matrix.

» u*v' `ans =  32`	Äußeres Produkt zweier Vektoren Das Ergebnis der Multiplikation eines Zeilenvektors mit einem Spaltenvektor ist ein Skalar.
» u*3 `ans =  3  6  9`	Multiplikation mit einem Skalar.
» 3*u `ans =  3  6  9`	Bei der Multiplikation eines Vektors oder einer Matrix mit einem Skalar ist die Reihenfolge egal.

» W^2
`ans = 100  117  54`
`  58  73  48`
`  19  25  21`
	Potenzieren einer Matrix (im Beispiel 2. Potenz der Matrix `W`)

» u*W `ans = 19  25  21`	Vektor-Matrix-Produkt Multiplikation eines Zeilenvektors mit einer Matrix.

» W*u'
`ans = 26`
`  32`
`  14`
	Matrix-Vektor-Produkt
	Multiplikation einer Matrix mit einem Spaltenvektor.

» X=W\T
`X = 7.300  -6.700  -3.700`
`  -5.600  6.400  3.400`
`  1.300  -0.033  -0.033`
	Division (1. Fall): Lösung der Gleichung W*X=T
	(`help mldivide`)

» x=W\u'
`x =  -3.7000`
`  3.4000`
`  -0.0333`
	Wenn bei der Matrix-Division anstelle der Matrix `T` der Spaltenvektor `u'` verwendet wird, so stellt `x` die Lösung des linearen Gleichungssystems `W*x= u'` dar. Verwendet wird der Gaußsche Algorithmus.

» X=T/W
`X =-0.1000  4.2333  -8.1333`
`  -0.1000  4.5667  -8.4667`
`  0.9000  -4.1000  9.2000`
	Division (2. Fall): Lösung der Gleichung X*W=T
	(`help mrdivide`)

» x=eig(W)
`x = 13.5485`
`  3.1536`
`  -0.7021`
	Vektor `x` der Eigenwerte der Matrix `W`.

» [X,D]=eig(W)	Matrix `X` der Eigenvektoren, Diagonalmatrix `D` der Eigenwerte

4.5.2 Feldoperationen: Elementweise Multiplikation bzw. Division von zwei Vektoren

`» u.*v` `ans = 4 10 18`	x*y würde eine Vektormultiplikation zur Folge haben. Um eine elementweise Multiplikation zu erreichen, wird diese spezielle Operation mit einem Punkt eingeleitet. (`help times`)
`» x=u.\v` `x = 4.000 2.500 2.000`	Auch die elementweise Division wird mit einem Punkt vor dem Operator eingeleitet: Division (1. Fall): Lösung der Gleichung u*x=v (`help ldivide`)
`» x=u./v` `x = 0.250 0.400 0.500`	Division (2. Fall): Lösung der Gleichung v*x=u (`help rdivide`)
`» x=u.^v` `x = 1 32 729`	Potenzierung, wobei `v` ein Vektor oder ein Skalar sein kann.
`» x=u.^3` `x = 1 8 27`	Auch beim Potenzieren eines Vektors mit einem Skalar, muss die Operation mit einem Punkt eingeleitet werden.

Tipp: Weitere Informationen und Matrizenoperationen sind im MATLAB-„Getting Started“-Handbuch unter „Manipulating Matrices“ nachzuschlagen.

4.6 Grafische Darstellungen von Funktionen

4.6.1 Einfache Grafiken

Der einfachste Befehl zur Ausgabe von Grafiken auf dem Bildschirm ist der Befehl `plot`. Dieser Befehl bezieht sich wieder auf einen Vektor, der die Ordinaten- bzw. y-Werte enthält, oder eine Matrix, die mindestens ein Paar an Abszissen- und Ordinatenwerte bzw. x- und y-Werte enthält. Wird nur ein Vektor mit Ordinatenwerten angegeben, so verwendet MATLAB generell die Folge 1, 2, 3, ... als x-Werte.

Die Grafiken werden dabei generell, wie in *Abb. 3* gezeigt, in einem separaten Grafikfenster ausgegeben. Werden mehrere Grafiken hintereinander ausgegeben, werden die vorherigen Grafiken überschrieben, solange kein neues Grafikfenster mit dem Befehl `figure` geöffnet wird oder mit „`hold`“ die bestehende Grafik beibehalten wird. Das Grafikfenster hat eine eigene Menüleiste, über die diese Grafik direkt ausgedruckt oder in Textverarbeitungsprogrammen eingebunden werden kann.

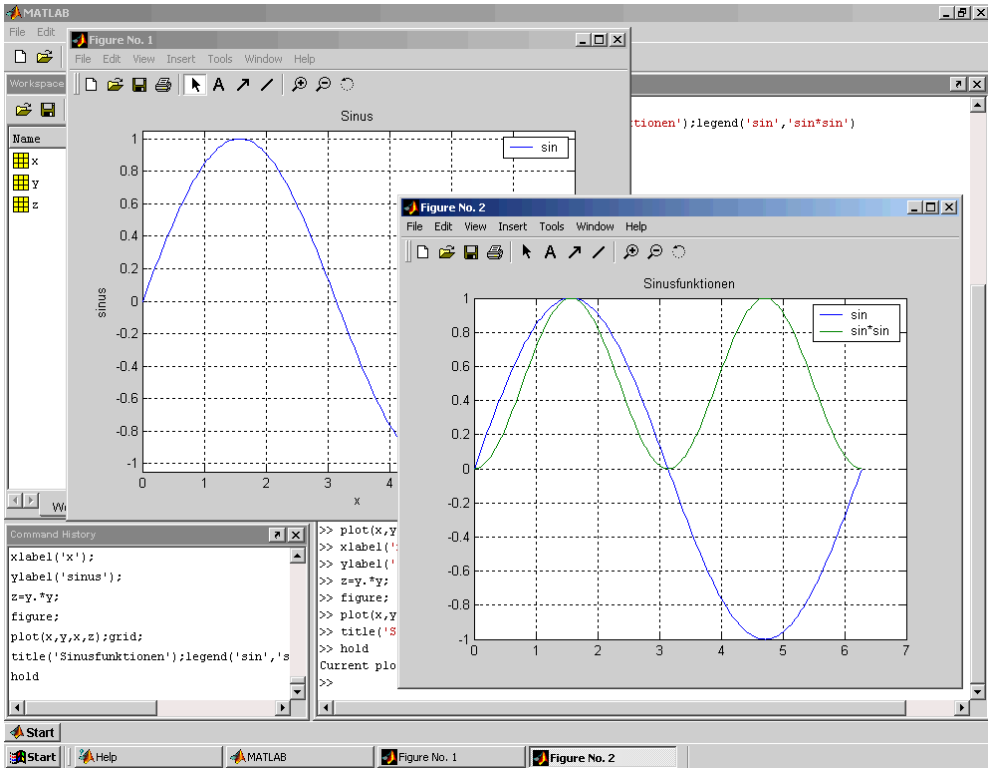


Abb. 3: Grafikfenster

➢ Beispiel: Grafische Darstellung von SINUS- und COSINUS-Funktionen

`» x=[0:pi/90:2*pi];`	Wertebereich definieren (Abszissen-, bzw. x-Werte) Linearer Wertebereich mit Anfangswert 0, Endwert 2π und einem gleichmäßigen Abstand von π/90 zwischen den Werten.
`» y=sin(x);`	Berechnung der SINUS-Werte für alle Werte von `x`, wobei `x` im Bogenmaß eingegeben wird.
`» plot(x,y)`	Grafikfenster wird geöffnet und die Sinuskurve zu den vorgegebenen x-Werten dargestellt. Ist vorher bereits ein Grafikfenster geöffnet worden, wird die bestehende Grafik überschrieben.
`» grid`	Gitternetzlinien werden erzeugt – oder bestehende wieder entfernt.
`» legend('sin')`	Einfügen einer Legende.
`» title('Sinus')`	Einfügen eines Grafiktitels.
`» xlabel('x');`	Einfügen der Beschriftung der x-Achse
`» ylabel('sin(x)');`	Einfügen der Beschriftung der y-Achse
`» z=y.*y;`	Berechnung der Funktion $z=\sin(x)^2$.
`» figure`	Öffnen eines neuen Grafikfensters
`» plot(x,y,x,z); grid`	In dem neuen Grafikfenster werden die Sinus- und die Sinus^2-Kurve dargestellt, der Befehl für Gitternetzlinien wird gleich angefügt.
`» legend('sin', 'sin^2');`	Einfügen der Legende; bei mehreren Kurven werden die jeweiligen Bezeichnungen durch Kommata getrennt hintereinander geschrieben.
`» hold`	Mit „`hold`“ oder „`hold on`“ bleibt die aktuelle Grafik mit allen Achseneinstellungen, Gitternetzlinien, Titeln, Legenden und anderen Eigenschaften bestehen. Weitere Kurven mit dem `plot`-Befehl werden zu den bestehenden Kurven gezeichnet. Mit wiederholtem „`hold`“-Befehl bzw. „`hold off`“ wird wieder zurück in den Standardmodus zurück gewechselt, so dass ein weiterer „`plot`“-Befehl die bestehende Grafik komplett ersetzt.
`» plot(x,cos(x))`	Nach dem „`hold`“-Befehl wird zusätzlich die COSINUS-Funktion im vorherigen aktuellen Grafikfenster dargestellt, alle Einstellungen bleiben erhalten.
`» legend('sin', 'sin^2','cos');`	Erweiterung der Legende um die COSINUS-Funktion.

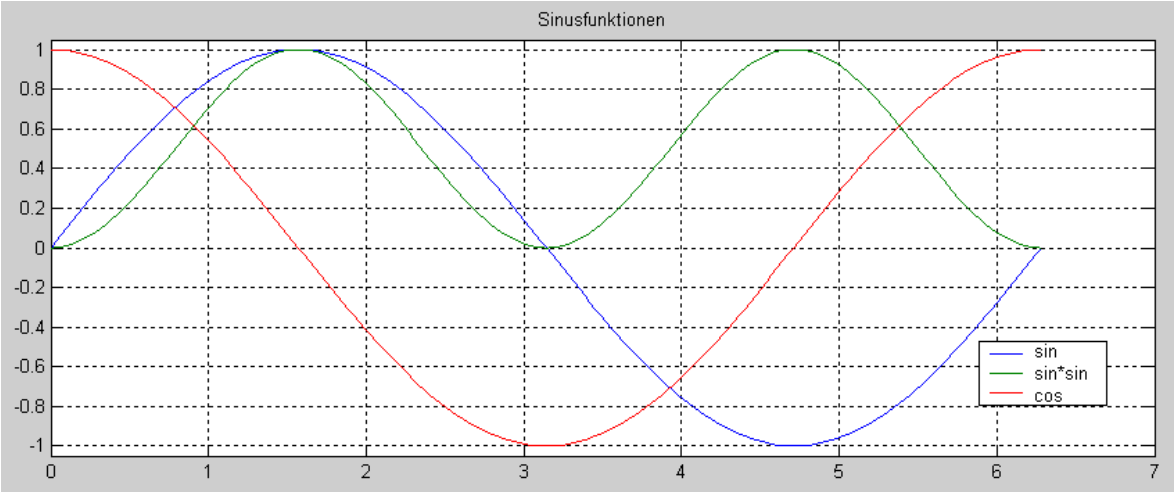


Abb. 4: Sinus-, Sinus^2- und Cosinus-Funktion

4.6.2 Grundsätzliche Grafiktypen

`» plot(x,y)`	Linearer x-y-Plot.
`» loglog(x,y)`	Grafik mit beiden Achsen logarithmisch.
`» semilogx`	Grafik mit x-Achse logarithmisch.
`» semilogy`	Grafik mit y-Achse logarithmisch.

4.6.3 Weitere Grafikfunktionen – Eigenschaften der Grafik

Bezüglich der Beschaffenheit und der Beschriftung von Grafiken gibt es noch weitere Befehle, von denen die Nützlichsten hier nur kurz aufgelistet werden sollen:

`» gtext('Text')`	Text, der mit der Maus in der Grafik positioniert wird; MATLAB wechselt dazu automatisch vom Befehlsfenster zum aktuellen Grafikfenster.
`» axis([xmin, xmax, ymin, ymax]);`	Die Achsen können auch nach Wusch des Benutzers skaliert werden; dazu dient der Befehl „`axis`“, wobei die Skalierungsfaktoren in der Form eines Vektors eingegeben werden.
`» plot(x,y,'y--');`	Schließlich gibt es noch Optionen, mit denen die Farbe und Stil der Linien festgelegt werden kann; diese folgen im „`plot`“-Befehl nach Festlegung der Vektoren für die Abszissen- und Ordinatenwerte in Hochkommas eingeschlossen, hier z.B. x-y-Plot in gelber gestrichelter Linie. Weitere Linien- oder Punkttypen sind:

Farbe		Punkttyp		Linientyp	
b	Blau	.	Punkt	-	Solid
g	Grün	o	Kreis	:	Dotted
r	Rot	x	x-Marker	-.	Dash dot
c	Cyan	+	Plus	--	Dashed
m	Magenta	*	Stern		
y	Gelb (yellow)	s	Quadrat		
k	Schwarz	d	Raute		
		v	Dreieck (nach unten)		
		^	Dreieck (nach oben)		
		<	Dreieck (nach links)		
		>	Dreieck (nach rechts)		
		p	Pentagram		
		h	Hexagram		

Hierbei geben die Punkttypen an, in welchem Symbol ein einzelner Punkt der Kurve angezeigt wird.

Weitere Eigenschaften der Grafik können auch direkt über die Menüleiste des Grafikfensters geändert werden, wie z.B. die Skalierung der Achsen über „Edit“ → „Axis Properties“.

MATLAB bietet noch viele weitere Möglichkeiten zur grafischen Ausgabe auf die an dieser Stelle nicht näher eingegangen werden soll; sie können bei Bedarf dem MATLAB-Handbuch entnommen, bzw. mit **`help plot`** von der MATLAB-Oberfläche aus aufgerufen werden.

5. Programmieren in MATLAB

Das Programmieren unter MATLAB wird vor allem bei der Verwendung von M-Files interessant. Kontrollstrukturen wie z.B. Schleifen können jedoch auch direkt im Befehlsfenster auf der MATLAB-Oberfläche eingegeben werden.

5.1 Programmcode generieren und abspeichern mit Hilfe des MATLAB Editors (M-Files)

MATLAB stellt eine voll funktionstüchtige Programmiersprache (basierend auf C) zur Verfügung, die es erlaubt, eine Folge von MATLAB Ausdrücken in eine Datei zu schreiben und sie mit einem einzigen Befehl auszuführen. Das Programm, das sogenannte M-File, kann über den M-File Editor (Aufruf über die Taskleiste in MATLAB → *„File“* → *„New“* → *„M-File“*) oder mit einem beliebigen Textprogramm erzeugt werden, hinter dem Dateinamen muss nur die Endung **`.m`** (`Dateiname.m`) stehen.

Der Dateiname wird der neue Befehl, mit dem das Programm von der Matlab-Oberfläche oder von Simulink aus gestartet werden kann. M-Files können Programme sein, die einfach nur eine bestimmte Reihenfolge von MATLAB Befehlen ausführen (z.B. sich wiederholende Aufgaben, wie sie oft in der Regelungstechnik vorkommen). Es können aber auch beliebige Funktionen sein, die Eingangsgrößen akzeptieren und bestimmte Ausgangswerte erzeugen.

5.1.1 Beispiel für eine Abfolge von MATLAB-Befehlen

```
% summe.m ist ein einfaches Beispielprogramm zur Berechnung der Summe der Zahlen 1,..,n
clc;                          % Bildschirm löschen
echo on;                      % alle Befehle, die im M-File ablaufen werden ausgegeben
diary('Beispiel.doc')         % Mitprotokollieren aller Ein- und Ausgaben in bel. Datei
clear n sum i tast            % Alle Variablen auf 0 setzen
n=input('Bitte eine ganze Zahl eingeben:    ') % Eingabe von n (letzter Summand)
% Überprüfen von n:
if isempty(n)                 % liefert "true" bzw. 1 wenn n eine leere Matrix ist.
    disp('Keine Eingabe erfolgt. Bitte neu starten (Taste).'),tast=input(' ');
    return                    % Zurück zur MATLAB-Oberfläche, wo Programm startete
elseif n==0                   % n=0
    disp('Bitte neu starten und andere Zahl eingegeben (Taste).'),tast=input(' ');
    return                    % Zurück zur MATLAB-Oberfläche, wo M-File gestartet wurde
else
    sum=0;
    for i=1:n
        sum=sum+i;
    end
end
disp('Die Summe der Zahlen 1 bis n beträgt: '),sum
echo off;                     % ausgeführte Befehle werden nicht mehr ausgegeben
```

Aufruf auf der MATLAB-Oberfläche: **`summe`**

5.2 Kontrollstrukturen

Kontrollstrukturen sind Befehlsfolgen, mit denen die Abarbeitung gezielt beeinflusst werden kann. Hierzu zählen Schleifen, die mehrmals durchlaufen werden können oder konditionale Kontrollstrukturen wie „if ... else“-Anweisungen.

- “FOR”-Schleife

`» for i=Ausdruck`
`  Anweisungen`
`  end`
	Wenn ein Anwender vorgeben will, wie oft hintereinander eine Anweisungen oder ein Block von Anweisungen ausgeführt werden soll, so verwendet er die „FOR“-Schleife. Eine „FOR“-Schleife muss immer mit `end` abgeschlossen werden. Anweisungen können dabei auch mehrere Befehle sein; diese müssen nicht in Klammern stehen. Der große Unterschied zu herkömmlichen Programmiersprachen liegt darin, dass in der MATLAB „FOR“-Schleife der *`Ausdruck`* prinzipiell eine Matrix ist. Bei jedem Durchgang der Schleife werden die Spalten der Matrix nacheinander der Variablen `i` zugewiesen.

`» A=[1 2`
`  4 5]`
`» for i=A`
`  x=i`
`  end`
`x =  1`
`  4`
`x =  2`
`  5`
	Beispiele für “FOR”-Schleife
	1. Möglichkeit: *`Ausdruck`* ist eine Matrix
	Der Variablen `i` wurden also bei jedem Durchgang die Spaltenvektoren zugewiesen, womit solche Konstruktionen insbesondere bei Matrizenberechnungen von Vorteil sind.

`» a=1:3;`
`» for i=a`
`  x=i`
`  end`
`x =  1`
`x =  2`
`x =  3`
	2. Möglichkeit: *`Ausdruck`* ist ein Vektor
	Hierbei war nun der Ausdruck ein Zeilenvektor, d. h. eine Matrix mit nur einem Element pro Spalte. Wieder wird beim Durchlaufen der „FOR“-Schleife bei jedem Durchgang der Variablen `i` eine Spalte, d.h. hier also ein Skalar, zugewiesen. Diese Art der „FOR“-Schleife entspricht derjenigen, die von den gewöhnlichen Programmier-sprachen her bekannt ist.

- “WHILE”-Schleife

`» while Ausdruck`
`  Anweisungen`
`  end`
	Hierbei ist *`Ausdruck`* eine logische Bedingung; solange diese Bedingung erfüllt ist, wird die Folge von *`Anweisungen`* in der Schleife wiederholt. Am Ende einer While-Schleife muss immer `end` stehen. Im Gegensatz zu den anderen Programmiersprachen ist das Ergebnis der logischen Bedingung eine Matrix; solange alle Elemente dieser Matrix ungleich Null sind, wird die Schleife durchlaufen.

`» A=[1 2;3 4];`
`» while A`
`  A(1,2)=A(1,2)-1`
`  end;`
`A =  1  1`
`  3  4`
`A =  1  0`
`  3  4`
	Beispiel für “WHILE”-Schleife
	1. `Ausdruck` ist eine Matrix
	Erst nach zweimaligem Schleifendurchlauf ist ein Element der Matrix zu Null geworden; damit wird die Schleife abgebrochen. Im Normalfall wird aber als Bedingung ein Skalar verwendet, also eine Matrix mit lediglich einem Element. Diese Art der Anwendung ist vom Umgang mit den anderen Programmiersprachen her bekannt.

`» n=10; x=1;`
`» while n>0`
`  n=n-x`
`  end;`
	2. `Ausdruck` ist ein Skalar

➢ "IF ... ELSE"-Schleife

`» if Ausdruck1`
`  Anweisung1`
`  elseif Ausdruck2`
`  Anweisung2`
`  else`
`  Anweisung3`
`  end`
	Als Abschluss einer solchen Kontrollstruktur muss in jedem Fall wieder `end` stehen. Die Abarbeitung solcher Kontrollstrukturen erfolgt wieder wie in anderen Programmiersprachen. Die Ausdrücke `Ausdruck1` und `Ausdruck2` sind wie bei der „While"-Schleife logische Bedingungen; das Ergebnis einer logischen Bedingung ist im allgemeinen wieder eine Matrix. Wenn z. B. alle Elemente der Matrix, die sich durch Auswerten von `Ausdruck1` ergibt, ungleich Null sind, so wird `Anweisung1` ausgeführt und die weiteren Anweisungen `Anweisung2` und `Anweisung3` werden übersprungen.

`» A=[1 1;3 4];`
`» if A`
`  A(1,2)=A(1,2)-1`
`  else`
`  A(1,2)=A(1,2)+2`
`  end`
`A =  1  0`
`  3  4`
	Beispiel für "IF ... ELSE"-Schleife
	Nach der ersten Abarbeitung erhält man das angegebene Ergebnis, da die Bedingung `A` erfüllt ist, da alle Elemente der Matrix `A` ungleich Null sind!
	Bei einer weiteren Abarbeitung der "IF ... ELSE"-Schleife wird die zweite Anweisung ausgeführt.
	Im allgemeinen wird aber die Bedingung ein Skalar und keine Matrix sein.

5.3 Funktionen in MATLAB

In MATLAB können auch eigene Funktionen programmiert werden. Diese Funktionen werden als separate M-Files abgespeichert. Das erste Wort des M-Files muss dabei `function` sein, um den Status als Funktion zu kennzeichnen. Einer Funktion können Argumente übergeben werden; innerhalb einer Funktion definierte Variablen sind lokal und werden nicht auf der MATLAB-Oberfläche abgespeichert. Mit MATLAB-Funktionen kann eine eigene neue Befehlsbibliothek erstellt werden.

➢ Syntax einer Funktion

```
» function [Ausgabeargumente] = Funktionsname(Eingabeargumente)
  Anweisungen
```

5.3.1 Beispiel für eine Funktion

Funktionen, wie in dem folgenden Beispiel können unterschiedliche Eingangsgrößen übergeben werden. Damit sind Funktionen flexibel einsetzbar, wenn die gleiche Operation für unterschiedliche Variablen ausgeführt werden soll. In Simulink können Funktionen kontinuierliche Ausgangswerte liefern.

```
function y = stat(u)
% Die Funktion y=stat(u) ist eine einfache Funktion zur Berechnung des Mittelwerts und
% der Standardabweichung der Elemente des Vektors u. Aufgerufen wird die Funktion mit
% stat(VARIABLE), wobei VARIABLE ein beliebiger definierter Vektor ist.
% Funktionen können auch in Simulink eingesetzt werden (Block "Fcn" unter
% "Functions & Tables"). "u" ist dann die Eingangs-, "y" die Ausgangsgroesse.
n = length(u);
mean = sum(u)/n;
stdev = sqrt(sum((u-mean).^2/n));
y=[mean,stdev];
disp('Der erste Wert des Ergebnisses ist der Mittelwert der Elemente des Vektors x.')
disp('Der zweite Wert ist die Standardabweichung.')
```

Aufruf auf der MATLAB-Oberfläche, nachdem erst ein Vektor x als Eingangsgröße definiert wurde:
`x=[1 3 5 7 9];ergebnis=stat(x)`

Damit können in MATLAB wie in anderen Programmiersprachen auch, neben Kontrollstrukturen auch Funktionen definiert und somit auch komplexe mathematische Berechnungen ausgeführt werden.

6. Einführung in die „Control Toolbox“ (spezielle Befehle & Werkzeuge für die Regelungstechnik)

Tipp: Eine genauere Erklärung zu den jeweils folgenden Befehlen der „Control Toolbox“ findet sich im Anhang 2, der „Zusammenstellung der in der Regelungstechnik wichtigsten MATLAB Befehle und Funktionen“.

6.1 Übertragungsfunktion Gs eines Regelkreis

Beispiel einer Übertragungsfunktion: $Gs(s) = \frac{K_S}{1 + s \cdot T_S}$ (PT1 Glied)

oder allgemein: $Gs(s) = \frac{a_n \cdot s^n + ... + a_2 \cdot s^2 + a_1 \cdot s + a_0}{b_n \cdot s^n + ... + b_2 \cdot s^2 + b_1 \cdot s + b_0}$

- Eingabe und Darstellung von Polynomen in MATLAB

`» num=[2 1 0]`
`num = 2 1 0`
`» den=[3 2 1 ]`
`den = 3 2 1`
	Polynome, z.B. $a_n \cdot s^n + ... + a_1 \cdot s + a_0$ werden als Zeilenvektoren bestehend aus den Koeffizienten in absteigender Reihenfolge dargestellt.

`» Gs=tf(num,den)`
`Transfer function:`
`2 s^2 + s`
`---------------`
`3 s^2 + 2 s + 1`
	Für eine Übertragungsfunktion bestehend aus Zähler und Nenner werden Zähler (engl. „numerator“) und Nenner (engl. „denumerator“) als separate Zeilenvektoren eingegeben. Mit der Funktion `tf(Zähler,Nenner)` (`tf` für engl. „transfer function“) wird eine Übertragungsfunktion in der bekannten Form definiert.

`» Ks=5;Ts=3;`
`Gs2=tf(Ks,[Ts 1])`
`Transfer function:`
`5`
`-------`
`3 s + 1`
	Die Zeilenvektoren können allerdings auch direkt innerhalb der runden Klammern des `tf`-Befehls eingegeben werden, wie hier in diesem Beispiel des PT1-Glieds.

`» num=conv([1 5],[1 6])`
`num = 1 11 30`
`» den=conv([1 6 10],`
`[1 3])`
`den = 1 9 28 30`
`» Gs=tf(num,den)`
`Transfer function:`
`s^2 + 11 s + 30`
`----------------------`
`s^3 + 9 s^2 + 28 s + 30`
	Eingabe einer Übertragungsfunktion als Multiplikation von mehreren Polynomen, z.B. wenn Pol- und Nullstellen des Systems bekannt sind: $Gs(s) = \frac{(s+5)\cdot(s+6)}{(s+3)\cdot(s^2+6\cdot s+10)}$
	Mit dem Befehl `conv(Polynom1,Polynom2)` können jeweils zwei Polynome einfach multipliziert werden. Die Polynome werden dazu wieder als Zeilenvektoren der Koeffizienten in absteigender Reihenfolge eingegeben, um dann folgendes Ergebnis zu erhalten: $Gs(s) = \frac{s^2 + 11 \cdot s + 30}{s^3 + 6 \cdot s^2 + 11 \cdot s + 6}$

- Umwandlung einer Übertragungsfunktion in die Polform (faktorisierte Form)

`» Gs_zpk=zpk(Gs)`
`Zero/pole/gain:`
`(s+6) (s+5)`
`----------------------`
`(s+3) (s^2 + 6s + 10)`
	Mit dem Befehl `zpk(Gs)` kann die Polform ausgegeben werden, d.h. die Übertragungsfunktion wird in die einzelnen Polynome zerlegt. Vorteil ist, dass die Kompensation von Polstellen durch die Nullstellen eines Reglers gleich deutlich und überprüfbar wird. Ein Polynom 2. Ordnung deutet auf ein konjugiert-komplexes Polpaar hin.

6.2 Grafische Darstellungsmöglichkeiten einer Übertragungsfunktion

➢ Übersicht über die wichtigsten grafischen Darstellungsmöglichkeiten[2]:

`» impulse(Gs)`	Impulsantwort (Gewichtsfunktion).
`» step(Gs);grid`	Sprungantwort (Übergangsfunktion).
`» bode(Gs);grid`	BODE-Diagramm (Frequenzlinien).
`» nyquist(Gs)`	NYQUIST Ortskurve. *Unerwünschte Darstellung des negativen Frequenzbereichs kann deaktiviert werden durch Rechtsklick mit der Maus auf eine freie Fläche neben der Nyquist-Kurve und bei „Show“ den Haken vor „Negative Frequencies“ entfernen. Diese Funktion ist nicht mehr sichtbar, sobald die Grafik in irgendeiner Weise bearbeitet wurde!*
`» pzmap(Gs)`	Pol- und Nullstellendiagramm des offenen Regelkreis.
`» rlocus(Gs)`	Wurzelortskurve (WOK) im geschlossenen Regelkreis.

`» subplot(2,2,3);`
`nyquist(Go_pi);`
`» subplot(2,2,1);`
`step(Gs);`
`» subplot(2,2,2);`
`bode(Gs,Gr_pi,Go_pi);`
`» subplot(2,2,4);`
`rlocus(Go_pi)`
	Mit dem **`subplot`**-Befehl wird ein Grafikfenster geöffnet (siehe Abb. 5), in dem Platz frei gehalten wird für z.B. 4 Grafiken, jeweils 2 Reihen (1. Ziffer) und je 2 Grafiken pro Reihe (2. Ziffer). Beliebige Anzahlen von Reihen und Spalten sind möglich, allerdings wird die Größe der einzelnen Grafiken immer kleiner. Die Grafiken werden durchnummeriert von links nach rechts, dann von oben nach unten. Die letzte Ziffer in der Klammer gibt den Platz an, auf dem die folgende Grafik abgebildet wird, z.B. Nyquist auf Platz 3, Sprungantwort auf Platz 1, BODE auf Platz 2 und WOK auf Platz 4. Die Reihenfolge der Belegung ist beliebig, Felder können auch überschrieben werden. Wird einer der Grafikplätze nicht belegt, bleibt das Feld grau.

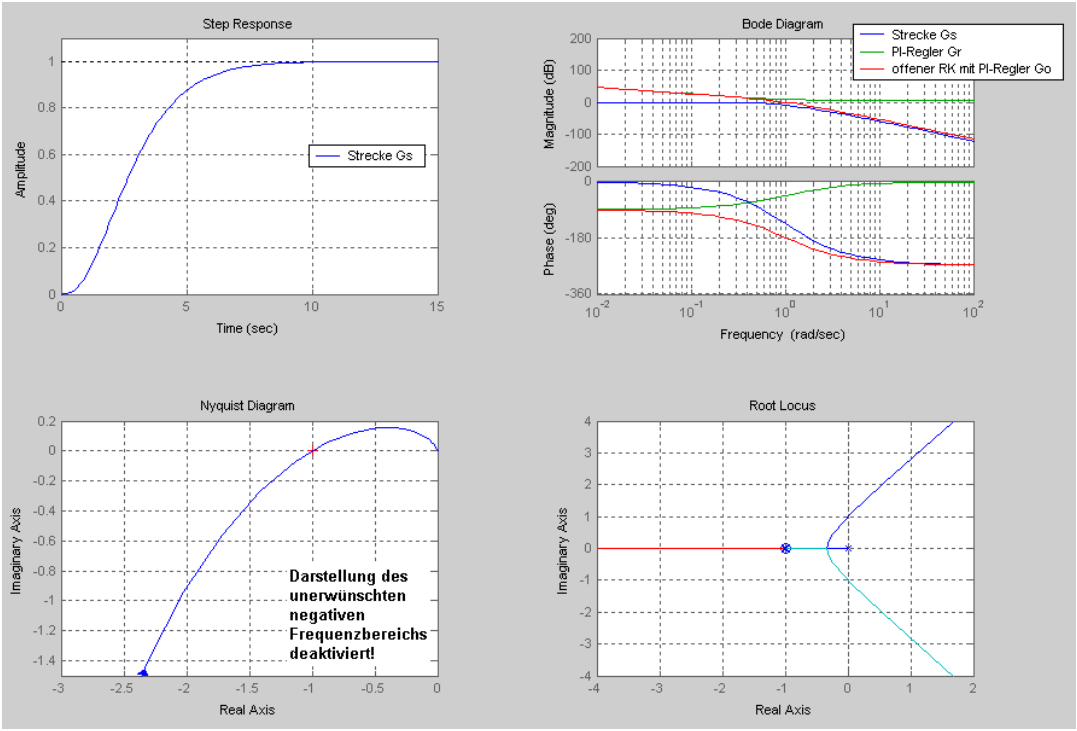


Abb. 5: Grafische Darstellungen von Übertragungsfunktionen (STEP, BODE, NYQUIST & WOK) mit Hilfe des `subplot`-Befehls in ein Grafikfenster gezeichnet.

[2] Varianten zu den einzelnen Befehlen können unter MATLAB mit „help funktionsname“ aufgerufen oder im „Anhang 2, in der „Zusammenstellung der in der Regelungstechnik wichtigsten MATLAB Befehle und Funktionen“ nachgelesen werden.

6.3 Charakteristika einer Übertragungsfunktion

- Pole einer Funktion berechnen

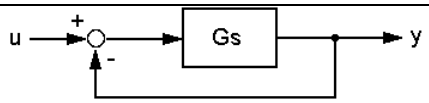
» PS=pole(Gs)
`PS = -3.0000 + 1.0000i`
`-3.0000 - 1.0000i`
`-3.0000`
	Berechnung der Polstellen einer Übertragungsfunktion.

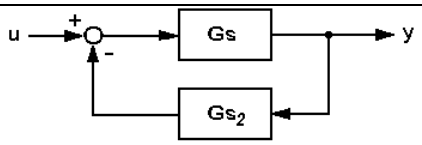
- Nullstellen einer Funktion berechnen

» NS=tzero(Gs2)
`NS = -6.0000`
`-5.0000`
	Berechnung der Nullstellen einer Übertragungsfunktion. Alternativ kann bei einfachen linearen Übertragungsfunktionen auch der Befehl `zero(Gs)` verwendet werden.

6.4 Zusammenschaltung von Modellen (Signalflussplan-Algebra)

- Geschlossener Regelkreis (Führungsübertragungsfunktion)

» Gw=feedback(Gs,1)
`Transfer function:`
`s^2 + 11 s + 30`
`-----------------------`
`s^3 + 10 s^2 + 39 s + 60`
	 $G_W = \frac{Gs}{1+Gs}$
	Die Führungsübertragungsfunktion mit negativer Rückführung berechnet sich mit dem einfachen `feedback`-Befehl.

» Gw2=feedback(Gs,Gs2)	 $G_{W2} = \frac{Gs}{1+Gs \cdot Gs_2}$ Befindet sich in der Rückführung ebenfalls Block, z.B. Gs2, so wird dieser durch Kommata getrennt eingefügt.
» Gw3=feedback (Gs,Gs2,+1)	Handelt es sich um eine positive Rückführung, so muss dies ebenfalls angegeben werden.

- Reihen- oder Kettenschaltung

» Gs_ser=series(Gs,Gs2)	Mit dem Befehl `series(Gs1,Gs2)` können die Übertragungsfunktionen von zwei Blöcken in Reihe geschalten werden.
» Gs_ser2=Gs*Gs2*Gs2	Das Multiplikationszeichen ***** bewirkt dasselbe wie der `series`-Befehl, jedoch ist es hier auch möglich, mehr als zwei Übertragungsglieder miteinander zu multiplizieren, also in Reihe zu schalten.

- Parallelschaltung

» Gs=parallel(Gs1,Gs2);	Mit dem Befehl `parallel(Gs1,Gs2)` können die Übertragungsfunktionen von zwei Blöcken parallel geschalten werden.

» Gr_PI= parallel (tf(1,[1 0]),5)
`Transfer function:`
`5 s + 1`
`-------`
`s`
	Beispiel: PI-Regler, bestehend aus P- und I-Anteil:
	$G_{r\_PI} = \frac{1}{s} + 5 = \frac{1}{s} + \frac{5 \cdot s}{s} = \frac{5 \cdot s + 1}{s}$

» Gs=Gs+Gs2+Gs2+Gs3;	Das Additionszeichen **+** bewirkt dasselbe, jedoch ist es hier möglich, mehr als zwei Übertragungsglieder miteinander zu addieren.

» Gr_PID=5+tf(1,[1 0]) +tf([1 0],1)
`Transfer function:`
`s^2 + 5 s + 1`
`-------------`
`s`
	Beispiel: PID-Regler, bestehend aus P-, I- und D-Anteil:
	$G_{r\_PID} = 5 + \frac{1}{s} + s = \frac{5 \cdot s}{s} + \frac{1}{s} + \frac{s^2}{s} = \frac{s^2 + 5 \cdot s + 1}{s}$

6.5 Eingabe eines Totzeitglieds

Totzeitglieder einzugeben ist erst ab MATLAB Vers. 5.3 möglich. Da es nicht einfach ist, mit Hilfe der MATLAB Hilfe den Trick zu finden, wie eine Totzeit (engl. „delay time“) für eine Übertragungsfunktion definiert wird, da es nämlich keine eigenständige Funktion für Totzeitglieder gibt, wird im Folgenden kurz darauf eingegangen:

Die Totzeit wird über die Eigenschaften einer Übertragungsfunktion („object properties“) eingegeben:

`» Gt=tf(1,1);`	Die Basisfunktion des Totzeitglieds muss als Übertragungsfunktion (transfer function) eingegeben werden, nur dann können die Eigenschaften einer Übertragungsfunktion geändert werden. Andernfalls erscheint eine Fehlermeldung beim Ausführen der folgenden Befehle.
`» set (Gt,'InputDelay',5)`	Von den möglichen Eigenschaften der Übertragungsfunktion wird die Eigenschaft „`InputDelay`“ auf den Wert 5 gesetzt. ***ACHTUNG: Es erfolgt keine Rückmeldung, ob der Befehl ausgeführt wurde, bzw. wie die geänderte Funktion nun aussieht.***

`» get(Gt)`
`num: {1}`
`den: {1}`
`Variable: 's'`
`Ts: 0`
`ioDelay: 0`
`InputDelay: 5`
`OutputDelay: 0`
`InputName: {''}`
`OutputName: {''}`
`InputGroup: {0x2 cell}`
`OutputGroup: {0x2 cell}`
`Notes: {}`
`UserData: []`
	Mit dem Befehl `get` können alle Eigenschaften und ihr momentaner Wert abgefragt werden. Hier eine Beschreibung der wichtigsten:
	`num`: Koeffizienten des Nenners als Zeilenvektor
	`den`: Koeffizienten des Zählers als Zeilenvektor
	`Variable`: Funktionsparameter ist „s“
	`Ts`: Abtastzeit (sample time) bei zeitdiskreten Systemen
	`ioDelay`: Eingangs- und Ausgangsverzögerungen als Matrix
	`InputDelay`: Eingangsverzögerung (Totzeit) als Vektor
	`OutputDelay`: Ausgangsverzögerung als Vektor
	Wenn mit Übertragungsfunktionen mit Totzeit gerechnet wurde, kann es sein, dass MATLAB in der Ergebnisfunktion die Totzeit in die Eigenschaft `ioDelay` schreibt. Im Zweifelsfall die Eigenschaften mit `get` abrufen und überprüfen. Mit `set(Gt,'ioDelay',10)` kann auch dieser Wert verändert werden.

`» set (Gt,'InputDelay',0)`	Verschiedene Funktionen der Control Toolbox funktionieren nicht mit Totzeiten, z.B. die Berechnung des geschlossenen Regelkreis `feedback(Gt,1)` oder die Berechnung der WOK mit `rlocus(Gt)`. Bei diesen Berechnungen muss die Totzeit vorher wieder zurückgesetzt, d.h. auf 0 gesetzt werden.

`» Gt`
`Transfer function:`
`exp(-5*s) * 1`
	Ausgabeformat einer Übertragungsfunktion mit Totzeit.
	ACHTUNG: Selbstverständlich kann einer beliebigen Übertragungsfunktion eine Totzeit. über das Ändern der Eigenschaften mit `get` zugeordnet werden.

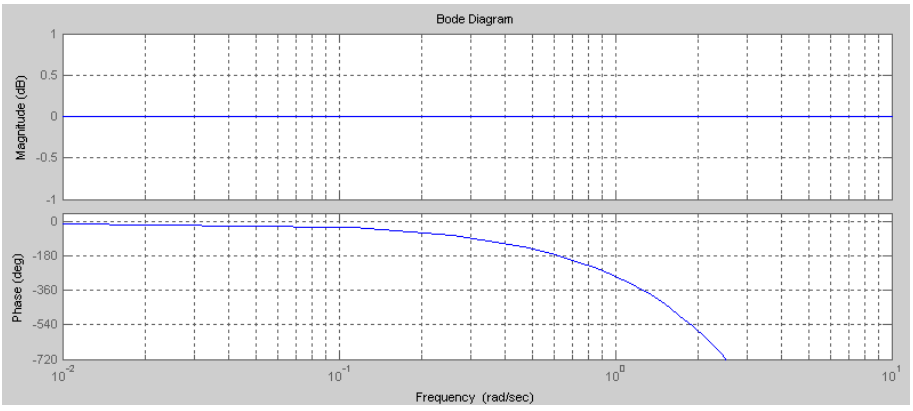


Abb. 6: Funktion Gt mit Totzeit, Betrag = 1, Totzeit = 5 s

7. Einführung in SIMULINK

7.1 Erste Schritte in SIMULINK

SIMULINK ist eine spezielle Toolbox von MATLAB mit der Systeme, Schaltkreise oder das Verhalten von Regelkreisen simuliert werden kann. Mit SIMULINK ist eine grafische Zusammenstellung von Modellen aus vorgefertigten oder selbst zu definierenden Blöcken möglich, die aus M-Files bestehen. Der Aufruf von SIMULINK erfolgt auf der MATLAB Oberfläche durch Eingabe von

```
» simulink
```

Danach öffnet sich ein separates Fenster mit einer grafischen Oberfläche, dem „Simulink Library Browser“, wie in *Abb. 7 (links)* dargestellt, mit einer Übersicht aller in SIMULINK verfügbaren Blöcke und vorgefertigten Modelle. Für die Gestaltung einer eigenen Simulation kann über die Taskleiste mit „*File*“ → „*New*“ → „*Model*“ (oder mit ) ein eigenes Fenster mit einem leeren Arbeitsblatt geöffnet werden, vgl. *Abb. 7 (rechts)*.

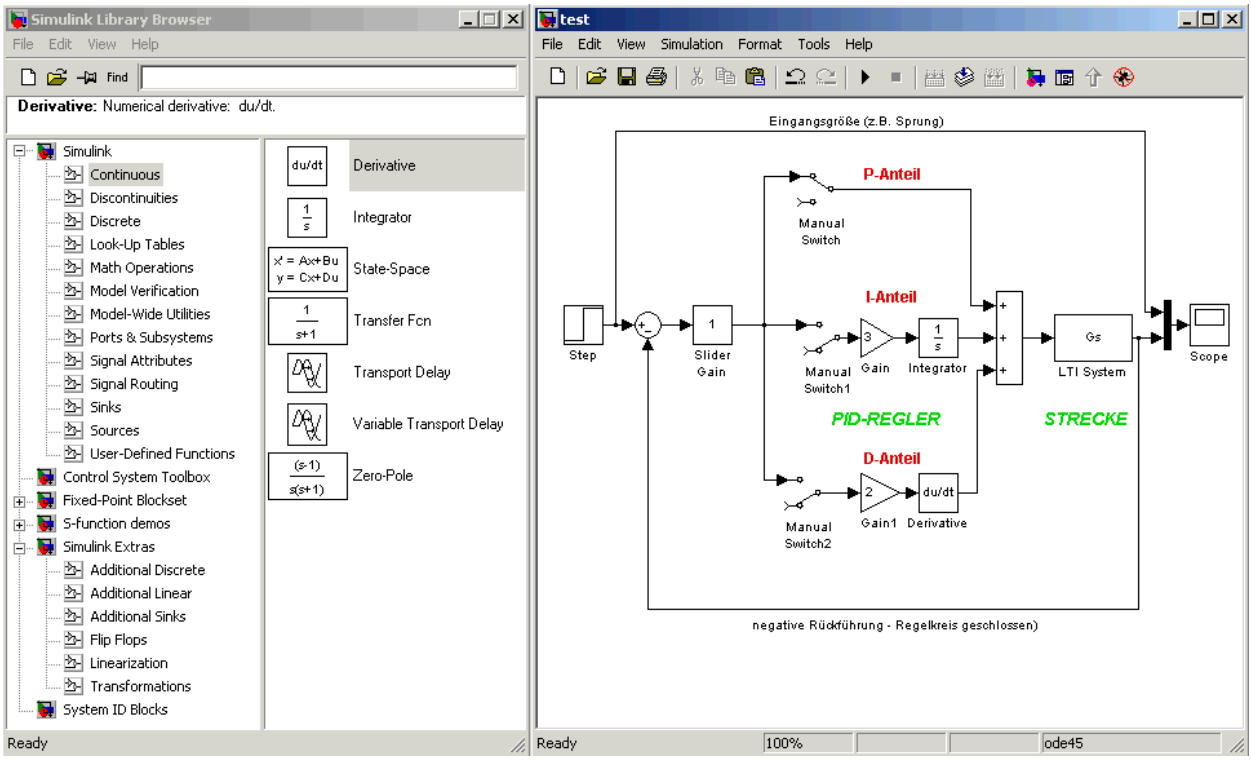


Abb. 7: Library Browser von Simulink und Beispiel-Simulationsschaltung eines Regelkreis

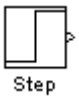
Der „SIMULINK Library Browser“ enthält eine Menüleiste und Icons für die unterschiedlichen Block-bibliotheken. Die wichtigsten Blöcke der für das Regelungstechnik-Praktikum relevanten Kategorien „Simulink“, „Control System Toolbox“ und „Simulink Extras“, und wo sie jeweils zu finden sind, sollen im Folgenden beschrieben werden.

Eine Kurzbeschreibung des jeweiligen Blocks und die einstellbaren Parameter, z.B. Verstärkungsfaktor, werden durch Doppelklicken auf den ausgewählten Block dargestellt. Durch Ziehen mit der linken Maustaste können die einzelnen Blöcke in ein Arbeitsblatt eingefügt werden.

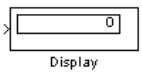
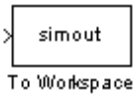
Verbunden werden die Blöcke entweder über direkte Verbindungslinien mit der linken Maustaste von jeweils markiertem Ausgang zum markierten Eingang des nächsten Blocks oder über die rechte Maustaste, die ein Aneinandersetzen einzelner Verbindungslinien erlaubt.

7.2 Kurzbeschreibung der wichtigsten SIMULINK Schaltblöcke

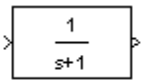
- **Simulink** → **Sources** (dt.: „Quellen“):Eingangsgrößen

	**Step**	Sprung mit variabler Sprunghöhe („Final Value“) und definierbarem Sprungbeginn („Step Time“).
	**Constant**	Konstanter einstellbarer Wert, während der laufenden Simulation nicht veränderbar.
	**Signal Generator**	Funktionsgenerator zum Erzeugen verschiedener Eingangssignale wie Sinus-, Sägezahn- oder Rechteckskurven mit einstellbarer Amplitude und Frequenz.
	**Clock**	Uhr zum Darstellen der vergangenen Simulationszeit, kann z.B. verwendet werden, um einen Zeitvektor als Variable mitzuschreiben.
	**Ground**	Vermeidet Fehlermeldungen, wenn unbenutzte Eingänge nicht mit anderen Blöcken verbunden sind.

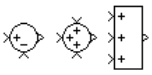
- **Simulink** → **Sinks** (dt.: „Senken“): Ausgangsblöcke

	**Scope**	Grafische Ausgabe von Werten auf einem Monitor, vergleichbar einem Oszilloskop. Messwerte können zusätzlich in einer Variablen auf der MATLAB Oberfläche gespeichert werden. Der Variablenname kann unter „Data History“ der „Scope Parameters“ gewählt werden (Doppelklick auf das 2. Symbol von links )
	**Display**	Numerische Ausgabe von Werten.
	**To Workspace**	Ausgangswerte werden in eine wählbare Variable geschrieben, die von der MATLAB Oberfläche aus bearbeitet werden kann, aber nicht dauerhaft gespeichert ist.
	**To File**	Ausgangswerte werden in Form eines Zeilenvektors dauerhaft in einer MAT-Datei gespeichert.
	**Terminator**	Vergleichbar dem Ground-Block, zur Vermeidung von Fehlermeldung, wenn unbenutzte Ausgänge nicht verbunden werden.

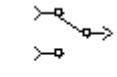
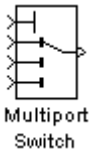
- **Simulink** → **Continuous:** Einfache Reglerblöcke bzw. Übertragungsfunktionen

	**Derivative**	D-Glied, Differenzierer, allerdings ohne weitere Parameter, so dass die Vorhaltezeit T_V mittels eines separaten Blocks (Verstärkungsfaktor) zusätzlich berücksichtigt werden muss.
	**Integrator**	I-Glied, Integrierer, bei dem die Nachhaltezeit T_N ebenfalls separat berücksichtigt werden muss. Es können Begrenzungen definiert werden, sowie ein bestimmter Ausgangswert („initial condition“).
	**Transfer Function**	Übertragungsfunktion, deren Zähler („Numerator“) und Nenner („Denumerator“) als Zeilenvektor in absteigender Folge der Koeffizienten (vgl. MATLAB-Befehl „`tf`“) eingegeben werden können. Es können auch bereits auf der MATLAB-Oberfläche definierte Variablennamen für Zähler und Nenner eingesetzt werden. Zu beachten ist, dass die Ordnung des Zählers kleiner gleich der des Nenners sein muss.

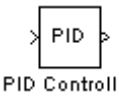
➢ **Simulink → Math Operations:** Mathematische Verknüpfungen

	**Gain**	Verstärkungsfaktor, bzw. Multiplikator, auch als P-Glied einsetzbar. Standardmäßig auf elementweise Multiplikation eingestellt, wenn als Faktor ein Vektor eingegeben wird; andere Varianten der Multiplikation mit Matrizen können gewählt werden.
	**Slider Gain**	Verstärkungsfaktor oder Multiplikator, der während der laufenden Simulation innerhalb der vordefinierten Grenzen verändert werden kann. Im Unterschied zum „Gain“-Block können jedoch nur skalare Werte eingegeben werden. Vorteilhaft z.B. um einen Regelkreis durch stetiges Erhöhen des Verstärkungsfaktors an die Schwinggrenze zu bringen.
	**Sum**	Addition oder Subtraktion verschiedener Signale, benötigt für die negative Rückführung im Regelkreis oder die Zusammenführung verschiedener parallel geschalteter Blöcke (z.B. PID-Regler). Die Form (rund oder rechteckig) ist variabel, ebenso wie die Anzahl und die Anordnung der Plus- und/oder Minuszeichen.

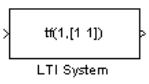
➢ **Simulink → Signal Routing:** Signalführung

	**Manual Switch**	Schalter der durch Doppelklicken zwischen den beiden möglichen Zuständen wechselt, z.B. zum Zu- oder Abschalten einzelner Regelglieder. Nicht beschaltete Eingänge sollten mit „Ground“ geschlossen werden.
	**Multiport Switch**	Schalter für mehr als zwei mögliche Zustände, die Anzahl der Eingänge ist dabei frei wählbar. Über den Steuereingang ganz oben (bzw. ganz links) können die Dateneingänge, nummeriert von oben nach unten, beginnend bei 1, ausgewählt werden.
	**Mux**	Führt mehrere Signale in ein Signal zusammen, z.B. um Eingangs- und Ausgangsgrößen eines Regelkreis direkt miteinander vergleichen zu können, siehe Beispiel in *Abb. 7*.
	**Demux**	Spaltet Vektorsignale in einzelne Skalare oder kleinere Vektoren, z.B. zum Trennen von Zeit- und Messwerten, die als gemeinsames Signal ankommen, in separate Variablen.

➢ **Simulink Extras → Additional Linear**

	**PID-Controller**	Extra Block für PID-Regler bei dem die Parameter für jeweils P-, I- und D-Anteil separat eingegeben werden können in der Form: P + I/s + D*s, d.h. als Verstärkungsfaktoren, nicht als Nachstell- und Vorhaltezeit.

➢ **Control System Toolbox**:

	**LTI Systems**	Universell einsetzbarer Block für Übertragungsfunktionen oder komplette Systeme. Statt der Übertragungsfunktion in der bekannten Form über Zähler und Nenner als Zeilenvektoren der Koeffizienten in absteigender Zeilenfolge, können auch Variablennamen der Übertragungsfunktionen eingesetzt werden. Aber auch hier gilt, dass die Ordnung des Zählers kleiner gleich der des Nenners sein muss.

Tipp: Dies ist nur eine Auswahl der wichtigsten SIMULINK-Blöcke. Weitere Blöcke können durch Mausklick auf den jeweiligen Oberbegriff aufgelistet werden, eine Beschreibung des jeweiligen Blocks durch Doppelklicken auf das Symbol.

7.3 Simulation eines SIMULINK-Modells

Als Beispiel soll der Regelkreis aus *Abb. 7 (rechts)* mit der bereits in *Kap. 6.1* definierten Strecke Gs in Simulink modelliert werden:

$$Gs(s) = \frac{s^2 + 11 \cdot s + 30}{s^3 + 6 \cdot s^2 + 11 \cdot s + 6}$$

Tipp: Die Funktion Gs muss bereits auf der MATLAB-Oberfläche definiert sein, bevor Gs als Parameter im „LTI Block" der SIMULINK-Simulation verwendet werden kann, um eine Fehlermeldung zu vermeiden.

Das Regelkreismodell soll nun simuliert werden. Das Modell, das in SIMULINK in Form von Blöcken grafisch eingegeben wurde, wird sicherheitshalber als *.mdl*-Datei unter einem frei wählbaren Dateinamen abgespeichert. Diese Datei enthält aber lediglich Information darüber, welche Blöcke eingegeben wurden, wie diese parametriert und wie die einzelnen Blöcke verbunden sind. Beim Start einer Simulation wird diese Information in ein Simulationsmodell umgewandelt, das dann aus Differentialgleichungen besteht und im Arbeitsspeicher abgelegt wird. Ausführen einer Simulation heißt nun, dass dieses Modell aus Differentialgleichungen numerisch integriert wird.

Um eine Simulation eines Regelkreis zu starten, müssen zunächst noch die geeigneten Parameter unter „*Simulation*" → „*Simulation Parameters...*" (Taskleiste der SIMULINK-Arbeitsfläche) für die Simulation eingestellt werden.

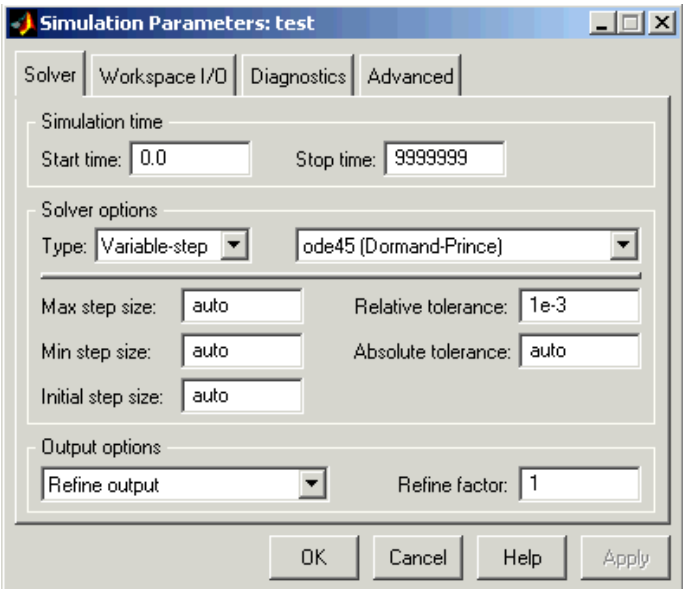


Abb. 8: Einstellen der Parameter einer SIMULINK-Simulation

Wie bereits erwähnt, ist zum Ablauf einer Simulation eine numerische Integration notwendig. Der für das jeweilige Problem passende Integrationsalgorithmus kann in diesem Fenster ausgewählt werden, wobei das standardmäßig eingestellte „ode45"-Verfahren von Dormand-Price bei einfacheren Regelkreisen akzeptable Ergebnisse ergibt, mehr Informationen zu den Integrationsverfahren sind im SIMULINK-Handbuch zu finden.

Wichtig ist vor allem die Vorgabe der Start- und Stop-Zeit der Simulation. Für das obige Beispiel aus *Abb. 7* mit einem Sprungbeginn bei 1 sek. Und nur dem P-Regler zugeschalten, soll als Start-Zeit „*0*" und als Stop-Zeit „*2*" (Sekunden) eingegeben werden. Weiterhin sind noch die minimale und maximale Schrittweite des Integrationsalgorithmus einzustellen, bei der Regelkreissimulation haben sich *Min. = 0.01* und *Max. = 0.1* bewährt, die Simulation funktioniert jedoch auch mit den standardmäßig vorgegebenen „*auto*"-Werten.

Die Simulation wird nun gestartet über die Taskleiste mit „*Simulation*" → „*Start*" oder durch Mausklick auf das Start-Symbol  . Durch Doppelklick auf den „Scope"-Block öffnet sich ein Grafikfenster, in dem der zeitliche Verlauf der Eingangs- und der Ausgangsgröße ausgegeben wird.

7.4 Tipps & Tricks für Regelkreis-Simulationen

Um die Sprungantwort eines Regelkreis zu simulieren und verschiedene Verstärkungsparameter im laufenden Betrieb zu testen, um so den Regelkreis an seine Schwinggrenze zu bringen, empfiehlt sich ein kleiner Trick:

1. Die STOP-Zeit der Simulation unter „Simulation" → „Simulation Parameters..." wird auf einen quasi unendlichen Wert eingestellt (z.B. *9999999*);
2. Statt eines Eingangssprungs wird ein kontinuierliches Rechtecksignal als Eingangsgröße verwendet, d.h. der „STEP"-Block wird durch den „SIGNAL GENERATOR"-Block ersetzt. Dessen Parameter werden auf Rechtecksignal (engl. „Square") gesetzt, die Amplitude kann bei „*1*" belassen werden, die Frequenz wird jedoch auf *0.1* Hz gesetzt (entspricht einer Periodendauer von 10 sec.).
3. Das SCOPE wird durch Doppelklick geöffnet und die „Scope Parameters" der Anzeige geöffnet durch Klicken auf das 2. Symbol von links (). Der Zeitbereich, der normalerweise auf „*auto*", d.h. automatische Anpassung, eingestellt ist, wird mit *10* (sec.) festgelegt.

Sobald alle Einstellungen entsprechend geändert wurden, kann die Simulation gestartet werden. Dadurch, dass im „Scope" genau die Periodendauer des Rechtecksignals angezeigt wird, ist genau ein Sprung zu sehen. Um nur einen Sprung von „*0*" auf „*1*" zu bekommen, kann die Amplitude des „Signal Generator" auf „*0.5*" gesetzt werden und das Eingangssignal um 0.5 nach oben verschoben werden (s. *Abb. 9*).

Nach Öffnen des „Slider Gain" durch Doppelklick auf den Block, kann nun in der laufenden Simulation der Verstärkungsparameter des Regelkreis geändert werden und das Ergebnis im „Scope" mitverfolgt werden (vgl. *Abb. 9, rechts*).

Falls die I-Anteile und/oder D-Anteile des Reglers dazugeschalten werden, kann es notwendig sein, die Frequenz des Eingangssignal entsprechend anzupassen und auch im „Scope" eine längere Periodendauer zu wählen, um das Verhalten des Regelkreis mitverfolgen zu können.

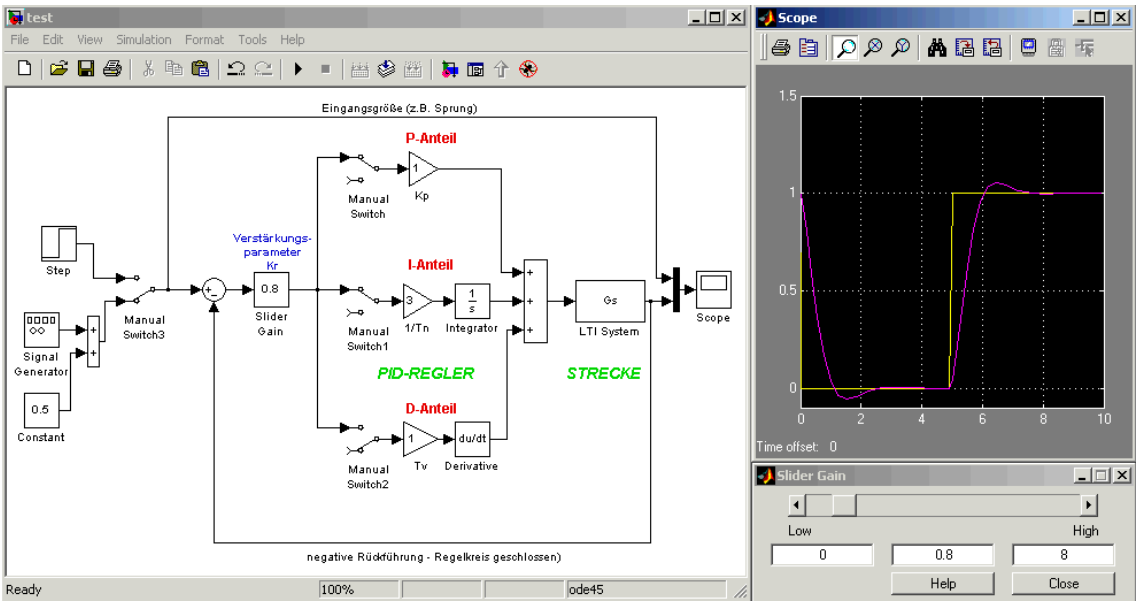


Abb. 9: Regelkreis mit Rechteckspannung als Eingang zum Testen von K_R im laufenden Betrieb (PI-Regler mit Kompensation der reellen Zeitkonstante, d.h. 1/Tn=3)

Tipp: Der D-Anteil kann Fehlermeldungen bezüglich der minimalen Schrittweite der Simulations-berechnungen verursachen. Oft ist es in diesem Fall empfehlenswerter, den PID-Block unter „Simulink Extras" zu verwenden.

7.5 Auswertung grafischer Darstellungen mit dem Scope

Die Ergebnisse im „Scope“ werden immer mit schwarzem Hintergrund dargestellt und diese Farbe lässt sich zumindest bis MATLAB Version 6.5 nicht so leicht ändern. Um den Verbrauch schwarzer Farbe beim Ausdruck mehrerer Ergebnisse zu reduzieren, bieten sich zwei Möglichkeiten an, den Hintergrund der grafischen Ausgabe in weiß zu ändern, eine sehr einfache und eine elegantere:

7.5.1 Einfaches Ändern der grafischen Darstellung im Bildbearbeitungsprogramm

Entweder den ganzen Bildschirminhalt über die „Druck“-Taste in den Zwischenspeicher laden und dann im Grafikprogramm als neues Bild einfügen und das Scope-Fenster manuell ausschneiden oder die „Schnappschuss“-Funktion mancher Grafikprogramme nutzen, die erlauben, ganze Fenster automatisch als Bild zu kopieren.

Sobald das Scope-Fenster als Bild abgespeichert ist, kann die Hintergrundfarbe mit Hilfe des jeweiligen Grafikprogramms geändert werden, z.B. indem die Farben weiß und schwarz vertauscht werden.

7.5.2 Elegantere Darstellung des Inhalts des Scope-Fensters über MATLAB

In den ‚Scope’ Parametern zu „Data History“ wechseln und „Save data to workspace“ markieren. Auf Wunsch kann ein eigener Variablenname vergeben werden, wichtig ist aber, dass das Format auf „Array“ geändert wird, damit die Daten in einem weiterverarbeitbaren Format abgespeichert werden.

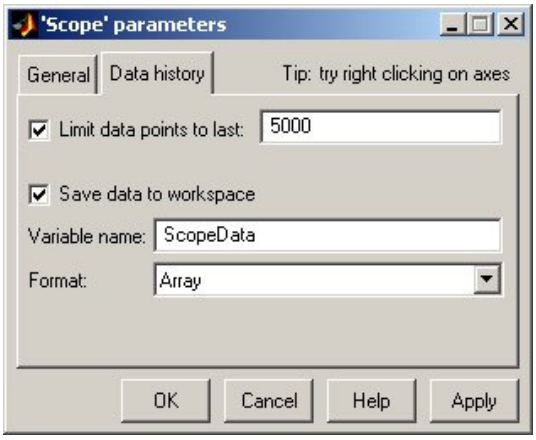


Abb. 10: Einstellungen der „Scope Parameter“

Nachdem die Simulation gestartet wird, werden nun die Daten in der Variablen `ScopeData` auf der MATLAB Oberfläche gespeichert, und zwar als 2-spaltige Matrix. Damit können die Daten mit Hilfe des `plot`-Befehls grafisch dargestellt werden und die Grafik auch nach Belieben und den Möglichkeiten von MATLAB verändert werden, wie in Kap. 4.6 beschrieben:

```
» plot(ScopeData(:,1),ScopeData(:,2)); grid; title('Simulink Scope')
```

bzw. bei zwei Messgrößen (z.B. Eingangs- und Ausgangsgröße:

```
» plot(ScopeData(:,1),ScopeData(:,2),ScopeData(:,1),ScopeData(:,3));grid
```

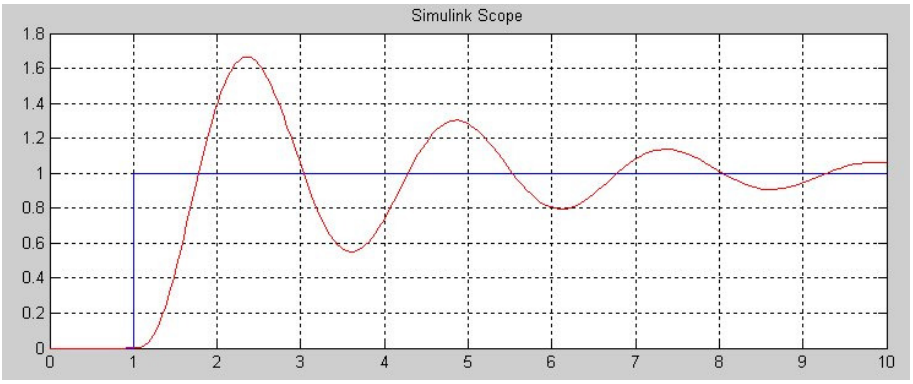


Abb. 11: Grafische Darstellung von Simulink Messdaten mit dem `plot`-Befehl

8. Grundlagen zum Reglerentwurf mit MATLAB

8.1 Bestimmung des Verstärkungsfaktors K_V mit Hilfe des BODE-Diagramm

Um den Verstärkungsfaktor K_V zu ermitteln wird anfangs von einem $K_V = 1$ ausgegangen und die Übertragungsfunktion G_0 des offenen Regelkreis als Serienschaltung der beiden Übertragungsglieder des Reglers G_R und der ermittelten Strecke G_S berechnet:

$$G_o = G_R \cdot G_S \qquad mit: \ |G_o(j\omega)|_{dB} = 20 \cdot log_{10}(|G_o(j\omega)|)$$

Das BODE-Diagramm besteht aus der Betragskennlinie und der Phasenkennlinie, die gemeinsam auch als Frequenzkennlinien bezeichnet werden. Dabei wird der Betrag $|G(j\omega)|$ als Amplitudengang in dB angegeben. Betrag $|G(j\omega)|_{dB}$ und Phase $\angle G(j\omega)$ werden jeweils als Funktion von $\log_{10}(\omega)$ aufgetragen. Mit MATLAB lässt sich das BODE-Diagramm von G_0 einfach grafisch erstellen.

Für die Beurteilung der Stabilitätsgüte eines Regelkreis mit Hilfe des BODE-Diagramms werden zwei verschiedene Parameter herangezogen, die Phasenreserve (auch Phasenrand) und der Verstärkungsrand (auch Amplitudenrand). Als Phasenreserve φ_{rand}, engl. „phase margin“, bezeichnet man an der Stelle $|G(j\omega)|_{dB} = 0$ dB (entspricht $|G(j\omega)| = 1$) die Differenz der Phase im Phasengang zu -180°. Der Amplitudenrand A_{rand}, engl. „gain margin“, stellt den Faktor dar mit dem $|G(j\omega)|$ an der Stelle $\varphi = -180°$ multipliziert werden müsste, damit der Amplitudengang durch den kritischen Punkt $|G(j\omega)|_{dB} = 0$ dB (entspricht $|G(j\omega)| = 1$) geht.

Ein Regelkreis befindet sich an der Stabilitätsgrenze wenn $A_{rand} = 1 \equiv 0$ dB und $\varphi_{rand} = 0°$ ist, d.h. für $A_{rand} > 1$ und $\varphi_{rand} > 0°$ ist er stabil. Regelkreise weisen einen ausreichend gedämpften und schnellen Regelverlauf – eine ausreichende Stabilitätsgüte – auf, wenn $2 < A_{rand} < 6$ und $30° < \varphi_{rand} < 75°$ ist.[3]

Für einen Reglerentwurf wird oft eine Phasenreserve von $\varphi_{rand} = 60°$ vorgegeben, der entsprechende Amplitudenrand ergibt sich dann aus dem BODE-Diagramm oder lässt sich mittels „`margin`“-Befehl (nähere Beschreibung in *Anhang 2*) leicht berechnen und grafisch darstellen (siehe *Abb. 12*). Der gesuchte Wert für K_V entspricht in diesem Fall bereits dem berechneten Wert für den Amplitudenrand.

ACHTUNG: ***Bei der grafischen Darstellung mit `margin` gibt MATLAB den Amplitudenrand Gm in [dB] an, bei der numerischen Berechnung wird Gm ohne Einheit angegeben.***

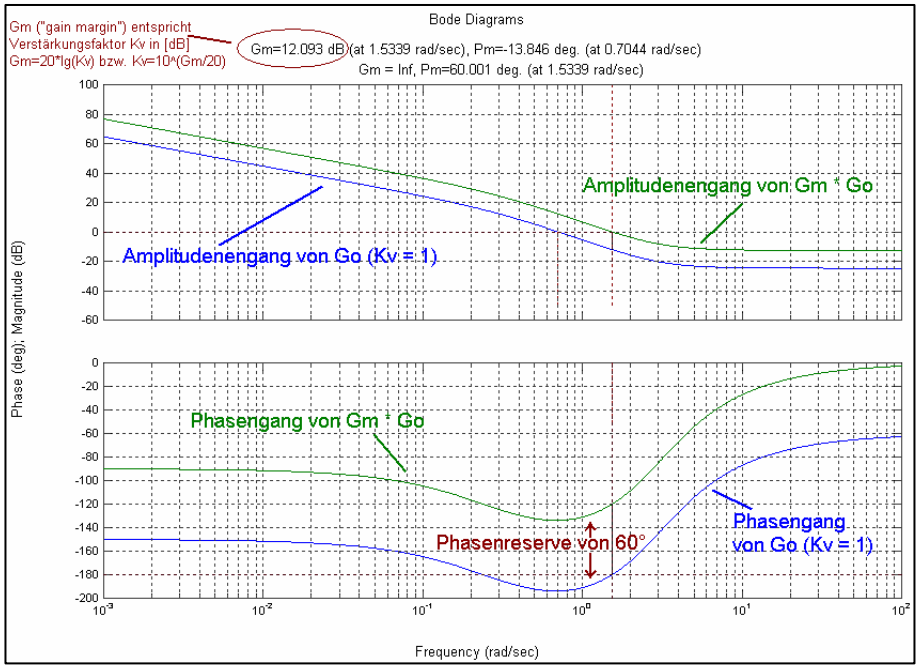


Abb. 12: Beispiel für zwei überlagerte BODE-Diagramme eines offenen Regelkreis mit markierter Phasenreserve bei –180° (erzeugt mit dem `margin`-Befehl).

[3] Aus: „Grundkurs der Regelungstechnik“, L.Merz / H.Jaschek, R.Oldenbourg Verlag, ISBN 3-486-21603-1.

8.2 Bestimmung des Verstärkungsfaktors K_V mit Hilfe der Wurzelortskurve (WOK)

Das Bode-Verfahren ist nicht immer anwendbar, da es oft nicht möglich ist, den Phasengang um den vorgegebenen Phasenrand zu verschieben, so dass die –180°-Linie geschnitten wird. In diesem Fall bietet sich die Bestimmung des Verstärkungsfaktors K_V mit Hilfe der Wurzelortskurve an.

Das Wurzelortsverfahren hat den Vorteil, dass in der grafischen Darstellung das Stabilitätsverhalten der Strecke anschaulich dargelegt wird und die quantitativen Anforderungen an den Regelkreis einfach ermittelt werden können.

Die Bestimmung des Verstärkungsfaktors K_V basiert dabei auf der Führungssprungantwort, die als repräsentativ für das Zeitverhalten des Regelkreis angesehen wird. Um den Grundanforderungen an eine gleichermaßen stabile wie schnelle Regelung zu genügen, werden die Überschwingweite und die Übergangszeit begrenzt. Daraus ergibt sich, dass ein konjugiert komplexes Polpaar, das eine Überschwingweite von höchstens 5% und eine Übergangszeit von max. 3 sec. verursacht, auf zwei vom Ursprung ausgehenden Strahlen liegt, die mit der negativ reellen Achse die Winkel ±45° bilden[4].

Die Dämpfung d, die sich daraus ergibt, kann bestimmt werden aus: $d = \cos(\varphi)$
Für den optimalen Winkel $\varphi = 45°$ ergibt sich deshalb eine Dämpfung (engl.: „damping“) d = 0.707.

MATLAB bietet mit der Toolbox `rltool` eine äußerst komfortable Möglichkeit zur Bestimmung des Verstärkungsfaktors K_V mit Hilfe der WOK. Nach Aufruf von `rltool` auf der MATLAB Oberfläche wird das Fenster „SISO Design Tool“ geöffnet. Alternativ lässt sich der Verstärkungsfaktor in ähnlicher Weise auch mittels der MATLAB-Befehle `rlocus` und `rlocfind` relativ einfach bestimmen (siehe *Anhang 2*).

8.3 „SISO[5] Design Tool“ zur Reglerbestimmung mittels Wurzelortskurve - `rltool`

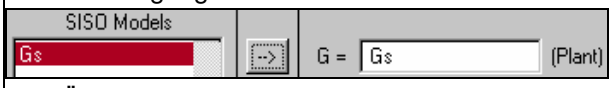
Folgende Eingaben sind notwendig, um eine erfolgreiche Reglerbestimmung durchzuführen:

1. Import der Übergangsfunktionen von Strecke Gs und Regler Gr (für Kr=1) aus MATLAB

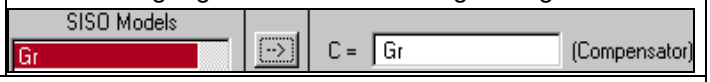
Anwählen von „File“ und dann „Import“. Alle auf der MATLAB Oberfläche verfügbaren Variablennamen von Übertragungsfunktionen werden im Unterfenster „SISO Models“ aufgelistet. Durch Markieren einer Funktion und Klicken auf den Pfeil-Button  jeweils links von der gewünschten Blockbezeichnung wird die Funktion an die jeweilige Stelle im dargestellten Blockschaltbild gesetzt.

→ **FILE**
→ **IMPORT**

- Übertragungsfunktion der Strecke:



- Übertragungsfunktion des bisherigen Reglers:



Unter „Compensators“, „Format“ und „Zero/pole/gain“ auswählen. Dies ist die gebräuchlichste Darstellungsform und verhindert Fehler beim Reglerentwurf des PI-Reglers.
Bevor die Reglerverstärkungen durch Verschieben der Pole auf der WOK ermitteln werden, sollte der Faktor vor dem Regler beachtet werden. Falls der Faktor ungleich 1 sein sollte, muss die ermittelte Reglerverstärkung durch diesen Faktor geteilt werden, damit die Reglereinstellungen stimmen.

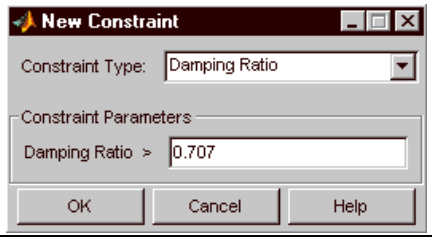
2. Eingabe der gewünschten Dämpfung d = cos (45°) ≈ 0.707

Unter „Edit“ lassen sich die Eigenschaften der Wurzelortskurve („Root Locus“) neu definieren:

- Gitternetzlinien („Grid“);
- neue oder bestehende Null- und Polstellen für einen Regler („Compensator“);
- Vorgaben für den Reglerentwurf („Design Constraints“), wie Übergangszeit („Settling Time“), Überschwingweite („Peak Overshot“) und Dämpfung („Damping Ratio“).

Die Dämpfung wird als Winkelhalbierende im 2. und 3. Quadranten der s-Ebene dargestellt.

→ **EDIT**
→ **ROOT LOCUS**
→ **DESIGN CONSTRAINTS**
→ **NEW**



[4] Aus „Regelungstechnik“ (8.Auflage), O.Föllinger, Hüthig Buch Verlag Heidelberg, ISBN 3-7785-2336-8.

[5] SISO = **S**ingle **I**nput – **S**ingle **O**utput

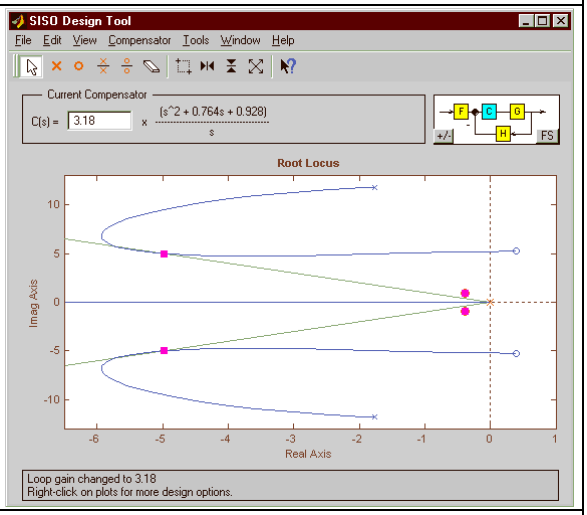
3. Bestimmung des Verstärkungsfaktors K_v

Standardmäßig werden die Wurzelorte für $K_V = 1$ durch pinkfarbene Zeichen auf der WOK markiert. Die runden Zeichen stehen dabei für die kompensierten Streckenpole.

Mit dem Cursor können die viereckigen Markierungen entlang der Wurzeläste verschoben werden, bis die gewünschte Dämpfung, repräsentiert durch die dargestellte Gerade, erreicht ist (siehe Abb. rechts).

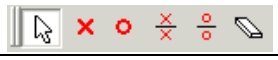
Im Fenster unter „Current Compensator“ kann der neue Verstärkungsfaktor jetzt abgelesen werden (im Bild: $K_V = 3.18$).

Falls die Kompensation der größten Zeitkonstanten kein befriedigendes Ergebnis liefert, können auch die kompensierenden Reglernullstellen beliebig verschoben werden, genauso wie auch die Polstelle des Reglers im Nullpunkt.



Zusätzliche Null- und/oder Polstellen können ebenfalls eingegeben oder vorhandene wieder gelöscht werden. Entweder über die Befehle in der Taskleiste oder grafisch durch beliebiges Platzieren, nachdem der gewünschte Pol bzw. die gewünschte Nullstelle ausgewählt wurde.

→ **COMPENSATORS**
→ **EDIT**
→ **C**

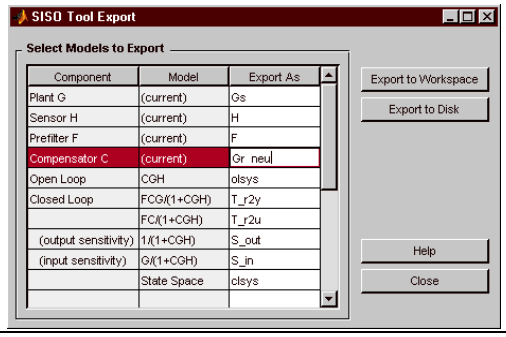


4. Exportieren der Ergebnisse auf die MATLAB Oberfläche

Zur weiteren Bearbeitung der Ergebnisse aus **`rltool`** kann die ermittelte Übertragungsfunktion des Reglers auf die MATLAB Oberfläche exportiert werden.

Mit „File“ und „Export“ wird nebenstehendes Fenster geöffnet. Normalerweise reicht es, nur den Regler („Compensator C“) zu markieren, sicherheitshalber umzubenennen, falls der Grundregler noch für weitere Reglerentwürfe verwendet werden soll, und mit „Export to Workspace“ auf die MATLAB Oberfläche zu exportieren.

→ **FILE**
→ **EXPORT**



5. Weitere Möglichkeiten von rltool

a) SIMULINK Diagramm des ermittelten Regelkreis

Mit „*Tools*“ und „*Draw Simulink Diagram*“ lässt sich automatisch eine einfache SIMULINK Simulation erstellen, die allerdings noch angepasst werden muss, bevor sie laufen kann.

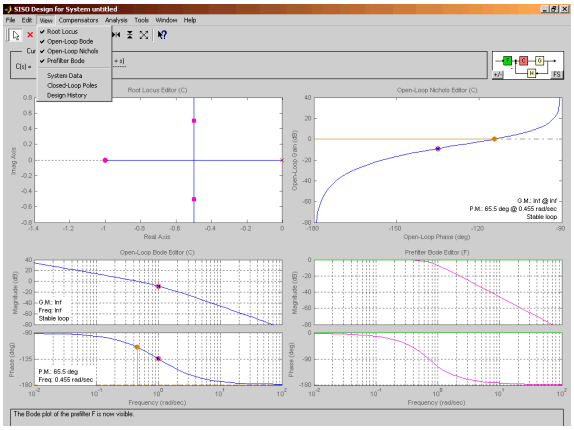
→ **TOOLS**
→ **DRAW SIMULINK DIAGRAM**

b) Weitere grafische Charakteristika des Regelkreis in der selben Grafik wie die WOK

Unter „View“ ist standardmäßig nur die Grafik „Root Locus“ ausgewählt. Zusätzlich lassen sich noch folgende weitere Grafiken auswählen, lassen sich folgende Charakteristika des aktuellen Regelkreis, Strecke mit jeweiligem Regler, darstellen (alle markierten Grafiken werden in einer Grafik dargestellt):

- Open-Loop Bode - Bode Diagramm für den offenen Regelkreis. Durch Verschieben der Amplitude kann der Verstärkungsfaktor des Reglers verändert werden, im Phasengang wird die zugehörige Phasenreserve angezeigt.
- Open-Loop Nichols - Nichols Diagramm. Durch Verschieben der Kurve kann der Verstärkungsfaktor geändert werden. Die zugehörige Phasen- und Amplitudenrandwerte werden angezeigt.
- Pre-Filter Bode – BODE-Diagramm des geschlossenen Regelkreis in Abhängigkeit vom Einfluss des Vorfilters F, sowie das BODE-Diagramm des Vorfilters. Wenn F = 1 ist, wird das Bode Diagramm des geschlossenen Regelkreis dargestellt.

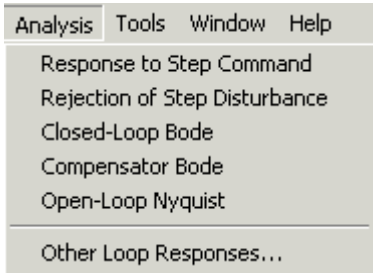
→ **VIEW**

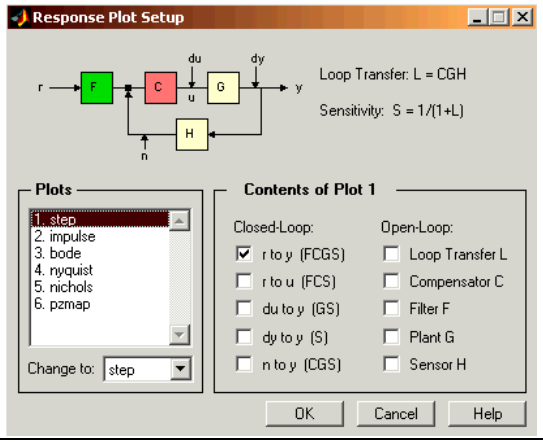


c) Weitere grafische Charakteristika des Regelkreis in separaten Grafiken (Analysis)

Unter „Analysis“ bieten sich weitere grafische Darstellungen der Charakteristika des Rekelkreis:

- „Response to Step Command“ – Sprungantwort
- „Rejection of Step Disturbance“ –
- „Closed-Loop Bode“ – BODE-Diagramm des geschlossenen Regelkreis
- „Compensator Bode“ – BODE-Diagramm des Reglers ohne Strecke
- „Open-Loop Nyquist“ – NYQUIST-Diagramm des offenen Regelkreis
- „Other Loop Responses …“:
  Insgesamt 6 verschiedene Grafiken können in beliebiger Reihenfolge ausgewählt und beliebig konfiguriert werden.
  Anhand der schematischen Darstellung des Regelkreis können für die einzelnen „*Plots*“ die zu durchlaufenden Zweige unter „*Contents of Plot x*“ ausgewählt werden.
  Weiteres bitte in der MATLAB-Hilfe nachlesen.





Anhang 1: Hilfethemen zu Matrizenoperationen: Erklärung der Operatoren und speziellen Zeichen

Die Hilfetexte zu den folgenden Matrixoperationen können mit Hilfe von `help` auf der MATLAB-Oberfläche aufgerufen werden, z.B.: **`help plus`** (oder **`help +`**)

Arithmetische Operatoren

Befehl	Beschreibung (in Engl.)	Zeichen	Befehl	Beschreibung (in Engl.)	Zeichen
plus	Plus	+	mpower	Matrix power	^
uplus	Unary plus	+	power	Array power	.^
minus	Minus	-	mldivide	Backslash or left matrix divide	\
uminus	Unary minus	-	mrdivide	Slash or right matrix divide	/
mtimes	Matrix multiply	*	ldivide	Left array divide	.\
times	Array multiply	.*	rdivide	Right array divide	./

Relationale Operatoren

Befehl	Beschreibung	Zeichen	Befehl	Beschreibung	Zeichen
eq	Equal	==	gt	Greater than	>
ne	Not equal	~=	le	Less than or equal	<=
lt	Less than	<	ge	Greater than or equal	>=

Logische Operatoren

Befehl	Beschreibung	Zeichen	Befehl	Beschreibung	Zeichen
	Short-circuit logical AND	&&	not	Logical NOT	~
	Short-circuit logical OR	||	xor	Logical EXCLUSIVE OR	
and	Element-wise logical AND	&	any	True if any element is nonzero	
or	Element-wise logical OR	|	all	True if all elements are nonzero	

Spezielle Zeichen

Befehl	Beschreibung	Zeichen	Befehl	Beschreibung	Zeichen
colon	Colon	:	punct	Comment	%
paren	Parentheses and subscripting	( )	punct	Invoke OS command	!
paren	Brackets	[ ]	punct	Assignment	=
paren	Braces and subscripting	{ }	punct	Quote	'
punct	Function handle creation	@	transpose	Transpose	.'
punct	Decimal point	.	ctranspose	Complex conjugate transpose	'
punct	Structure field access	.	horzcat	Horizontal concatenation	[,]
punct	Parent directory	..	vertcat	Vertical concatenation	[;]
punct	Continuation	...	subsasgn	Subscripted assignment	( ),{ },.
punct	Separator	,	subsref	Subscripted reference	( ),{ },.
punct	Semicolon	;	subsindex	Subscript index	

Bitweise Operatoren

Befehl	Beschreibung	Zeichen	Befehl	Beschreibung	Zeichen
bitand	Bit-wise AND.		bitxor	Bit-wise XOR.	
bitcmp	Complement bits.		bitset	Set bit.	
bitor	Bit-wise OR.		bitget	Get bit.	
bitmax	Max. floating point integer.		bitshift	Bit-wise shift.	

„Set“ Operatoren

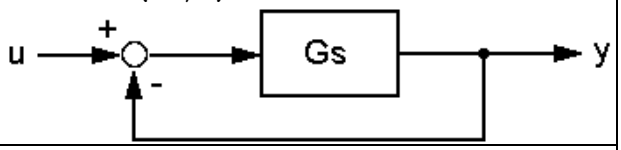
Befehl	Beschreibung	Zeichen	Befehl	Beschreibung	Zeichen
union	Set union.		setdiff	Set difference.	
unique	Set unique.		setxor	Set exclusive-or.	
intersect	Set intersection.		ismember	True for set member.	

Anhang 2: Zusammenstellung der in der Regelungstechnik wichtigsten MATLAB Befehle und Funktionen

ACHTUNG: Die vorliegende Übersichtstabelle ist keine vollständige Liste aller MATLAB-Befehle.

Ziel	Kommando	Erklärung
Unterdrücken der Bildschirmausgabe von Werten	***befehl ... ;***	Durch Beenden eine Befehlszeile mit Semikolon (**;**) werden Ergebnisse oder der Inhalt von Variablen nicht auf dem Bildschirm ausgegeben.
Hilfetexte zu einzelnen Befehlen abfragen	**help *befehl***	Mit **help** und dem gesuchten Befehls wird eine Kurzbeschreibung auf der MATLAB-Oberfläche ausgegeben. Genauere Informationen sind in den jeweiligen MATLAB User Guides zu finden.
Auflisten aller in MATLAB verfügbaren Parameter und Variablen	**whos** **who**	Mit der Eingabe von **who** können alle Variablen und Parameter, die im MATLAB Arbeitsbereich (Workspace) zur Verfügung stehen, aufgelistet werden. Mit **whos** werden Zusatzinformationen wie Dimension und Art der Variable aufgeführt.
Protokollieren der eingegebenen Befehle und von MATLAB ausgegebenen Ergebnisse	**diary** **diary('datei.txt')** **diary on** **diary off**	Mit **diary** wird die Protokollfunktion ein- oder ausgeschalten. Nach Eingabe von **diary** werden alle folgenden Befehle und Ausgaben auf der Matlab-Oberfläche in einer Textdatei protokolliert, bis zur erneuten Eingabe von **diary,** um das Protokoll zu beenden. Wird die Textdatei noch mal geöffnet, werden weitere Einträge am Ende angefügt. Ohne Angaben wird die Datei „*diary*" erzeugt. Es gibt keine Anzeige ob **diary** an oder aus ist!
Kommentare einfügen	**% Dies ist ein Kommentar**	Auf der MATLAB-Oberfläche oder in M-Files bzw. Funktionen können mit voranstehendem **%**-Zeichen Kommentare eingegeben werden
Formatieren der Ausgabe auf dem Bildschirm	**format compact** **format long** **format short**	Mit **format compact** werden Leerzeilen unterdrückt, z.B. das Protokoll (**diary**) nicht unnötig aufzublähen. Mit **format long** werden Zahlen bis auf 15 Stellen hinter dem Komma angezeigt, bei **format short** nur 5 Nachkommastellen; weitere Format-Befehle unter **help format**.
Speichern und Laden von Daten	**save DATEI VAR1 VAR2 ... VARn** **load DATEI**	In der Datei namens *DATEI* werden die Variablen VAR1, VAR2, ..., VARn gespeichert (Aneinanderreihung nur mit Leerzeichen). Durch **load *DATEI*** werden alle gespeicherten Variablen wieder auf die MATLAB-Oberfläche geholt.
Löschen von Variablen auf der MATLAB-Oberfläche	**clear VAR1 Var2 ...** **clear all**	Mit **clear** und den jeweiligen Variablennamen können einzelnen Variablen gelöscht werden, mit **clear all** werden wirklich alle Daten gelöscht.
MATLAB Befehlsfenster leeren	**clc**	Mit **clc** wird der Inhalt des Befehlseingabefenster (*Command Window*) gelöscht, die Variablen auf der MATLAB Oberfläche bleiben jedoch erhalten.
Ausgabe der Befehle eines M-Files auf der MATLAB-Oberfläche	**echo on** **echo off**	Mit **echo on** werden Befehle, die ein ablaufendes M-File ausführt auf der MATLAB-Oberfläche ausgeschrieben, nützlich z.B. für Berichte (Befehle und Ergebnisse in einem Bericht).
Rechnen mit einzelnen Spalten aus Matrizen	**matrix_name(:,i)** **matrix_name(:,i:k)**	**matrix_name**: Bezeichnung der Matrix; **i**: Bezeichnung der (ersten) gewünschten Spalte. **k**: Bez. der letzten gewünschten Spalte. Einzelne oder zusammenhängende Spalten einer Matrix können so dargestellt oder unter eigenem Namen abgespeichert werden.

Ziel	Kommando	Erklärung
Rechnen mit einzelnen Zeilen aus Matrizen	**matrix_name(i,:)** **matrix_name(i:k,:)**	**i**: Bez. der (ersten) gewünschten Zeile. **k**: Bez. der letzten gewünschten Zeile. Einzelne oder zusammenhängende Zeilen einer Matrix können so dargestellt oder unter eigenem Namen abgespeichert werden.
Übertragungs-funktion Gs berechnen aus Nenner und Zähler	**Gs=tf(zaehl,nenn)** **Gs=** **tf([a_n .. a_2 a_1 a_0],** **[b_n .. b_2 b_1 b_0])**	**Gs**: Übertragungsfunktion (Bez. beliebig) **zaehl**: Zähler **nenn**: Nenner Zähler und Nenner haben das Format eines Zeilenvektors bestehend aus den Koeffizienten in absteigender Reihenfolge, also [a_n ... a_3 a_2 a_1 a_0] bzw. [b_n ... b_3 b_2 b_1 b_0] für eine Übertragungs-funktion Gs mit: $Gs = \frac{a_n \cdot s^n + ... + a_2 \cdot s^2 + a_1 \cdot s + a_0}{b_n \cdot s^n + ... + b_2 \cdot s^2 + b_1 \cdot s + b_0}$ Falls a_1, a_2, ... $a_n = 0$ und $a_0 \neq 0$, dann können die eckigen Klammern um a_0 weggelassen werden.
Multiplikation von Polynomen z.B. bei der Eingabe einer Übertragungs-funktion	**conv([a_n .. a_2 a_1 a_0],** **[b_n .. b_2 b_1 b_0])**	Mit **conv([a_n .. a_1 a_0],[b_n .. b_1 b_0])** werden die Koeffizienten der Polynome entsprechend miteinander multipliziert: $(a_n \cdot s^n + ... + a_1 \cdot s + a_0) \cdot (b_n \cdot s^n + ... + b_1 \cdot s + b_0)$ Z.B.: `Gs=tf(5,conv([1 2],[3 4]))` ergibt: `5 / (s + 2) (3 s + 4) = 5 / 3 s^2 + 10 s + 8`
Übertragungs-funktion in Polform (faktorisiert)	**zpk(Gs)**	Mit **zpk(Gs)** wird die Übertragungsfunktion $Gs = \frac{a_n \cdot s^n + ... + a_2 \cdot s^2 + a_1 \cdot s + a_0}{b_n \cdot s^n + ... + b_2 \cdot s^2 + b_1 \cdot s + b_0}$ in der faktorisierten Form dargestellt, um Pole und Nullstellen direkt herauslesen zu können: $Gs = \frac{(1 + s \cdot n_1) \cdot (1 + s \cdot n_2) \cdot ... \cdot (1 + s \cdot n_n)}{(1 + s \cdot p_1) \cdot (1 + s \cdot p_2) \cdot ... \cdot (1 + s \cdot p_{n-1}) \cdot (r + q \cdot s + s^2)}$ mit $(p_n + s)$ bzw. $(n_n + s)$: einf. reelle Pole/Nullstellen und $(r + q \cdot s + s^2)$: komplex konjugiertes Polpaar
Eingabe von Totzeit T_T (Funktioniert erst ab MATLAB Version 5.3)	**set(Gs,'InputDelay',T_T)** **get(Gs)**	Mit dem Befehl **set(Gs,'Property',Wert)** werden die Eigenschaften der Übertragungs-funktion Gs verändert, die mit **get(Gs)** abgefragt werden. Die Totzeit eines Systems wird eingegeben mit: **set(Gs,'InputDelay',T_T)** und erscheint dann z.B. für $T_T = 30$ in folgender Form: `Gs = exp(-30*s) * 1.` Falls ein reines Totzeitglied definiert werden soll, muss Gs als Übertragungsfunktion eingegeben werden mit: `Gs=tf(1,1)`, nicht als `Gs=1.`
Parallelschaltung von mehreren Über-tragungsgliedern (z.B. PID-Regler)	**Gs=parallel(G1,G2)** **Gs=parallel(Gs,G3)** **Gs=G1+G2+G3**	Die Parallelschaltung von mehreren Übertragungs-gliedern kann mit dem Befehl **parallel**, aber auch als Summe berechnet werden. Mit **parallel** können jedoch immer nur zwei Übertragungsglieder auf einmal eingesetzt werden!
Reihen- oder Ketten-schaltung von mehreren Ü-Gliedern	**Gs=series(G1,G2,..)** **Gs=G1*G2*...**	Die Reihen- oder Kettenschaltung mehrerer Übertragungsglieder kann durch Multiplikation oder mit dem Befehl **series** berechnet werden.
Polstellen ausgeben	**pole(Gs)**	Alle Polstellen von **Gs** (auch konjugiert komplexe Polpaare) werden als Spaltenvektor ausgegeben.

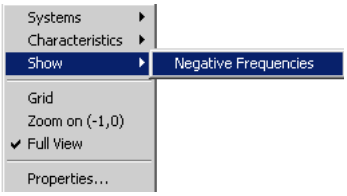
Ziel	Kommando	Erklärung
Betrag von Variablen (z.B. Polen) berechnen	**abs(*Variable*)**	Mit **abs(*Variable*)** kann der Betrag bzw. die Beträge, wenn *Variable* eine Matrix ist, berechnet werden, auch von komplexen Zahlen: **abs(X) = sqrt(real(X).^2 + imag(X).^2)**
Nullstellen ausgeben	**tzero(Gs)**	Alle Nullstellen von **Gs** (auch konjugiert komplexe Paare) werden als Spaltenvektor ausgegeben.
Graf. Darstellung der Nullstellen	**pzmap(Gs)**	Mit **pzmap(Gs)** werden die Pol- (**x**) und Nullstellen (**O**) von Gs in der s-Ebene graf. dargestellt.
Graf. Darstellung von x-y-Werten	**plot(x,y)** **plot(x1,y1,x2,y2)**	**x**: x-Achse, z.B. Zeit **y**: y-Achse, z.B. Messwerte über Zeit x Mehrere x-y-Paare in einem **plot** sind erlaubt.
Mehrere Grafiken in ein Bild, beliebig angeordnet	**subplot(m,n,z)** **subplot(3,1,1);** **step(Gs)** **subplot(3,1,2) ;** **bode(Gs)**	Mit **subplot(m,n,z)** wird ein Grafikfenster geöffnet, in dem Platz frei gehalten wird für **m** Reihen und **n** Spalten von Untergrafiken. Die Untergrafiken werden durchnummeriert von links nach rechts, dann von oben nach unten. Mit **z** wird die Platznummer angegeben.
Grafische Darstellung der Sprungantwort	**step(Gs)** **step(Gs,t_A:dt:t_E)** **step(Gs_1,Gs_2,..)**	**t_A**: Anfangszeit (optional, sonst „0“) **dt**: Zeitintervalle (optional) **t_E**: Stoppzeit (optional) Die Darstellung von mehreren Übertragungs-funktionen in einem Diagramm ist möglich.
Neues Grafikfenster öffnen	**figure**	Mit **figure** wird ein neues Grafikfenster geöffnet. Standardmäßig überschreibt MATLAB bestehende Grafiken im jeweils aktiven Fenster.
Mehrere Grafiken in einem Grafikfenster darstellen	**hold** **hold on** **hold off**	Mit **hold** oder **hold on** bleibt die aktuelle Grafik mit allen Achseneinstellungen, Gitternetzlinien, Titeln, Legenden und anderen Eigenschaften bestehen. Weitere Kurven mit dem **plot**-Befehl werden zu den bestehenden Kurven gezeichnet. Mit wiederholtem **hold**-Befehl bzw. **hold off** wird wieder zurück in den Standardmodus zurück gewechselt, so dass ein weiterer **plot**-Befehl die bestehende Grafik komplett ersetzt.
Gitternetzlinien in Grafiken einblenden	**grid**	Einfügen von Gitternetzlinien zur besseren Übersicht in Grafiken.
Führungsüber-tragungsfunktion des geschlossenen Regelkreis	**feedback(Go,1)**	Die Führungsübertragungsfunktion mit negativer Rückführung berechnet sich mit **feedback(Go,1)**. 
Definition eines bestimmten logarithmischen Wertebereich z.B. für die Frequenz ω	**w=logspace(d1,d2,n)**	Mit **logspace(d_1,d_2,n)** wird ein 1-reihiger Vektor definiert, der **n** logarithmisch gleichmäßig verteilte Werte zwischen 10^{d1} und 10^{d2} enthält. Wenn für n kein Wert eingegeben wird, gilt n = 50. Z.B. für ω = 0.1 bis 100, **w=logspace(-1,2)**.
Definition eines bestimmten linearen Wertebereichs	**X=linspace(x_1,x_2,n);**	Identisch zu **logspace**, nur das **x_1** den Anfangswert und **x_2** den Endwert eines Werte-bereichs darstellt, mit **n** linear verteilten Werten. Wenn für n kein Wert angegeben wird, gilt n = 100.

Ziel	Kommando	Erklärung

Berechnung von Polstellen für definierte Verstärkungsfaktoren k	`[p,k]=rlocus(Gs,k)`
	`[p,k]=rlocus(Gs)`
	`[p]=rlocus(Gs,k)`
		Mit `[p,k]=rlocus(Gs,k)` werden dem Vektor `p` die Polstellen (in komplexer Form) für definierte Verstärkungsfaktoren `k` zugeordnet. Mit `[p,k]=rlocus(Gs)` werden die Polstellen für einen vorgegebenen Bereich von `k` ausgegeben.

Grafische Darstellung der Wurzelortskurve	`rlocus(Go,K)`[6]
	`rlocus(Go,xa:dx:xe)`
	`rlocus(Go)`
		Mit `rlocus(Go,K)` wird die Wurzelortskurve für einen definierten Vektor `K` mit verschiedenen Werten für K_V grafisch ausgegeben, wobei alle Pol- und Nullstellen markiert sind. Für K = 1 erhält man das gleiche Ergebnis wie mit `pzmap(Go)`. Es kann auch ein Wertebereich mit Anfangswert x_a, Endwert x_e und Intervall dx definiert werden. Ohne Angabe von K, wird die WOK für beliebige K-Werte von [0, ∞[als Kurve angezeigt, wobei nur die Pole für K = 1 markiert werden.
		ACHTUNG: `rlocus` schließt den Regelkreis; nur Ü-Funktionen des offenen RK eingeben!

Identifizieren bestimmter Polstellen in der grafischen Darstellung	`[kv,p]=rlocfind(Gs)`	Mit `[k,p]=rlocfind` werden die Pol- und Nullstellen für den Bereich `k` grafisch dargestellt; mit Hilfe eines „Fadenkreuzes“ kann eine bestimmte Polstelle durch Anklicken gewählt werden, deren Daten (Verstärkungsfaktor k_V und der komplexe Wert der Polstelle) in MATLAB angezeigt werden.

Grafische Darstellung der Nyquist-Ortskurve	`nyquist(Gs)`
	`nyquist(Gs,w)`
	`[re,im,w]=`
	`nyquist(Gs)`
	`[re,im]=`
	`nyquist(Gs,w)`
	 
		Mit `nyquist(Gs)` wird die Nyquist-Ortskurve dargestellt. Mit `[re,im,w]=nyquist(Gs)` werden die Realanteile dem Vektor `re` und die Imaginäranteile dem Vektor `im` für die jeweiligen Frequenzwerte aus `w` zugeordnet. Der Frequenzbereich `w` wird vorgegeben mit `nyquist(Gs,w)` (Grafik) bzw. `[re,im]=nyquist(Gs,w)`.
		Rechtsklick mit der Maus auf eine freie Fläche neben der Nyquist-Kurve öffnet ein Auswahlfenster. Hier kann unter „Show“ die unerwünschte Darstellung des negativen Frequenzbereichs deaktiviert werden. Diese Funktion ist nicht mehr sichtbar, sobald die „Axes Properties“ aufgerufen wurden!

Grafische Darstellung des BODE-Diagramms	`bode(Gs)`
	`bode(Gs,w)`
	`[a,p,w]=bode(Gs)`
	`[a,p]=bode(Gs,w)`
		Mit `bode(Gs)` wird das BODE-Diagramm grafisch ausgegeben. Mit `[a,p,w]=bode(Gs)` werden die Amplitudenwerte dem Vektor `a` und die Phasenwerte dem Vektor `p` für die jeweiligen Frequenzwerte aus `w` zugeordnet. Der Frequenzbereich `w` wird vorgegeben mit `bode(Gs,w)` (Grafik) bzw. `[a,p]=bode(Gs,w)`.

Grafische Darstellung des Phasenrands im BODE-Diagramm	`margin(Gs)`	Mit `margin(Gs)` wird das BODE-Diagramm mit markiertem Phasen- und Amplitudenrand grafisch dargestellt. Der Amplitudenwert von Gs wird im Phasengang bei -180° markiert, der Phasenwert im Amplitudengang an der Stelle 1 ≡ 0 dB. Für Gm = 1 ≡ 0 dB und Pm = 0° wird angezeigt, dass der geschlossene Regelkreis instabil ist. ACHTUNG: Der angegebene Wert für Gm ist in [dB] und muss eventl. noch umgerechnet werden.

[6] Der Befehl `rlocus(Gs,K)` funktioniert nicht mehr bei Matlab 7.01, obwohl der Befehl auch hier in der Hilfe erklärt ist. Dieser Fehler wird hoffentlich bei zukünftigen Versionen korrigiert!

Ziel	Kommando	Erklärung
Berechnung von Kp mit Hilfe einer vorgegebenen Phasenreserve im BODE-Diagramm (Beide Befehle sind notwendig für die Berechnung)	`[a,p,w]=bode(Go,w);`	Für jeden Wert ω aus dem Wertebereich **w** (z.B. definiert über `logspace`-Befehl, s.o.), werden jeweils **a** ein Amplitudenwert und **p** ein Phasenwert zugeordnet (Bezeichnungen beliebig). Damit wird eine Wertetabelle für jeweils Amplitude und Phase bezogen auf ω erzeugt. *ACHTUNG: Der Wertebereich **w** sollte mittels BODE-Diagramm überprüft werden. Die interessierenden Phasenwerte, z.B. -180°+Phasenrand, müssen im Wertebereich enthalten sein, sonst ergibt der folgende `margin`-Befehl ein falsches Ergebnis, da dann Ergebnisse der äußersten Grenze des Wertebereichs eingesetzt werden.*
	`[Gm,Pm,wa,wp]= margin(a,p-prand,w)`	**prand**: Phasenrand **Gm**: Amplitudenwert („gain margin“) an der Stelle -(180°-Phasenrand) **Pm**: Phasenwert („phase margin“) an der Stelle, an der die Amplitude = 0 dB **wa**: Frequenz ω zu **Gm** (Amplitude) **wp**: Frequenz ω zu **Pm** (Phase) **a,p,w**: vorher zu berechnende Amplituden-, Phasen- und Frequenzwerte als Vektor. Der **margin**-Befehl sucht aus diesen Werten, z.B. als **Gm** den Amplitudenwert **a**, der zu dem Phasenwert **p** gehört, der am nächsten an -(180°-Phasenrand) herankommt. Je mehr Werte **a** und **p** in einem vorgegebenen Wertebereich berechnet wurden, desto exakter ist das Ergebnis. Der ausgegebene Amplitudenwert **Gm** entspricht dem gesuchten Verstärkungsfaktor Kp in [ ], da dies der Kehrwert des Betrags ist, um den der Amplitudengang verschoben werden muss, allerdings bereits von [dB] umgerechnet in [ ]. KEINE UMRECHNUNG MEHR ERFORDERLICH!
Platz für eigene Befehle:		

Anhang 3: Übungsaufgabe zum Reglerentwurf mit MATLAB

(Aufgabe bekannt aus Vorlesung Regelungstechnik I von Prof. Adermann, Hilfsblatt H 4 / 5a)

Für den gegebenen Regelkreis sind die Parameter für folgende Regler zu ermitteln:

1. **P-Regler**
2. **I-Regler**
3. **PI-Regler**
4. **PID-Regler**

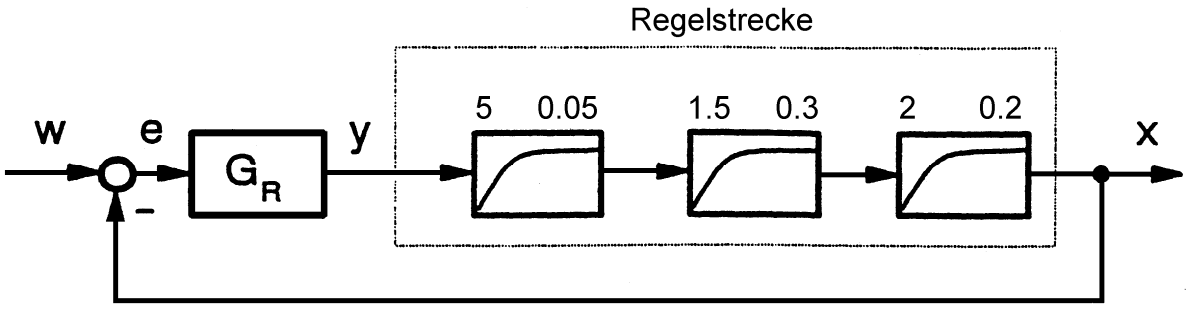


Abb. 13: Regelkreis: PT_3-Regelstrecke

Aufgabe 1: *Die Strecke* `Gs` *ist in MATLAB einzugeben und die Sprungantwort, die Wurzelortskurve (WOK), die NYQUIST-Ortskurve, sowie das BODE-Diagramm von* `Gs` *sind grafisch darzustellen.*

Tipp: *Strecke Gs als Reihe der Einzelstrecken berechnen lassen.*
MATLAB-Befehle: `step(Gs), rlocus(Gs), nyquist(Gs)` *und* `bode(Gs)`.

Aufgabe 2: *Die Reglerzeitkonstanten für den PI- und den PID-Regler sind durch Kompensation der größten Zeitkonstanten zu ermitteln.*

Tipp: *Annahme für Reglerentwurf: $K_R = 1$*

Aufgabe 3: *Die Reglerverstärkungen sind mit Hilfe des Frequenzkennlinienverfahrens (BODE-Diagramm des offenen Reglerkreis, vgl. Kap. 8.1) so abzulesen, dass die Phasenreserve jeweils 60° beträgt.*

Tipp: *MATLAB-Befehl* `margin` *verwenden (vgl. Anhang 2, letzte Seite). Nicht vergessen, vorher den Frequenzbereich w mit ausreichend vielen Werten (mind. 10000) zu definieren, so dass die Reglerverstärkung relativ genau berechnet werden kann. Die Berechnung der Reglerverstärkung mit Hilfe von MATLAB ist für alle Reglertypen gleich. Für jedes* `Go` *(Übertragungsfunktion des offenen Regelkreis aus Strecke und Regler) ist dieselbe Befehlsfolge auszuführen.*

Aufgabe 4: *Ein Vergleich der verschiedenen Regler wird anhand der Übergangsfunktionen der geschlossenen Regelkreise (Führungsübertragungsfunktionen) durchgeführt.*

Tipp: *Für jeden Regler jeweils Führungsübertragungsfunktion berechnen mit MATLAB-Befehl:* `Gw_x=feedback(Go_x,1)`.
Vergleichen der grafischen Darstellung der Führungssprungantwort aller vier Regler mit MATLAB-Befehl:
`step(Gw_1,Gw_2,Gw_3,Gw_4);grid;title(´Führungssprungantwort´)`

Lösungsansatz für die Übungsaufgabe

1. Strecke:

Die Strecke **Gs** besteht aus drei PT1-Gliedern, die Zahl links über dem Block gibt den Wert K_S an, die Zahl rechts den Wert T_S des einzelnen PT1-Glieds mit:

$$Gs_{Teilstrecke}(s) = \frac{K_S}{1 + s \cdot T_S}$$

2. Reglerentwurf:

P-Regler: $\boxed{Gr_1(s) = K_R}$

I-Regler: $\boxed{Gr_2(s) = \frac{1}{T_N \cdot s} = \frac{K_R}{s}}$ mit $K_R = \frac{1}{T_N}$

PI-Regler: $\boxed{Gr_3(s) = K_R' \cdot \left(1 + \frac{1}{T_N \cdot s}\right) = K_R' \cdot \frac{T_N \cdot s + 1}{T_N \cdot s} = K_R \cdot \frac{T_R \cdot s + 1}{s}}$ mit $K_R = \frac{K_R'}{T_N}$

PID-Regler: $Gr_4(s) = K_R' \cdot \left(1 + \frac{1}{T_N \cdot s} + T_V \cdot s\right) = K_R' \cdot \frac{T_N \cdot s + 1 + T_V \cdot T_N \cdot s^2}{T_N \cdot s}$

$$\boxed{Gr_4(s) = K_R \cdot \frac{(T_{R1} \cdot s + 1) \cdot (T_{R2} \cdot s + 1)}{s}}$$

mit $K_R = \frac{K_R'}{T_N}$ und aus Koeffizientenvergleich: $T_{R1} \cdot T_{R2} = T_V \cdot T_N$ und $T_{R1} + T_{R2} = T_N$

3. Reglerdimensionierung mit BODE-Diagramm

Wichtige Befehlsfolge für das Berechnen der optimierten Reglerverstärkung für beliebige Regler `Gr` (vgl. *Anhang 2* für nähere Erläuterung zu den Befehlen):

```
Go=Gr*Gs            % Berechnen der Übertragungsfunktion des offenen Regelkreis
bode(Go)                                    % BODE-Diagramm grafisch ausgeben,
          %um zu sehen, in welchem Frequenzbereich die Phase von -120° liegt
Prand=60;                              % Definieren der Phasenreserve Prand = 60°
w=logspace(-2,2,100000);                    % Definieren von Frequenzbereich w;
             % eventl. Anpassen an richtigen Bereich (vgl. dazu BODE-Diagramm.
             % Semikolon NICHT vergessen, sonst Auflistung von 100.000 Zeilen!
[a,p,w]=bode(Go_p,w);       % Zuordnen von Phasen- (p) und Amplitudenwerten (a)
[Kr,Pr,wa,wp]=margin(a,p-Prand,w)     % Berechnen des Verstärkungsparameters Kr
margin(Kr*Go);grid;title('Verschiebung von Go um Phasenrand von 60°')
                                        %Grafische Darstellung des Phasenrands
Gw =feedback(Go*Kr,1)      % geschlossener RK optimiertem Verstärkungsfaktor Kr
figure;step(Gs,Gw);grid;legend('Gs - Strecke im offenen RK','Gw -
geschlossener RK mit optimiertem Regler)         % Sprungantworten im Vergleich
Gw_p=Gw;      % eigener eindeutiger Name für jede Führungsübertragungsfunktion
```

4. Führungssprungantworten

Schnellster Regler:	PID-Regler
Langsamster Regler:	I-Regler
Regler mit bleibender Regeldifferenz:	P-Regler

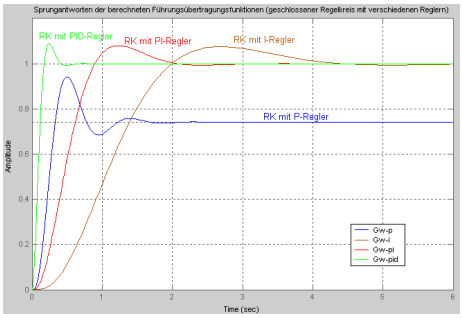


Abb. 14: Darstellung aller Regler in einer Grafik