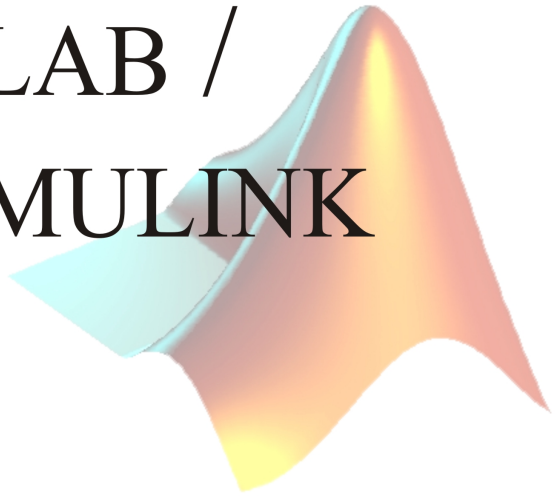
Gunter Diehm

Einführung in Matlab
für Studierende der Fakultät ETIT
Systemtheorie, Regelungstechnik

MATLAB / SIMULINK



Stand: 29. April 2013

Karlsruher Institut für Technologie

Institut für Regelungs- und Steuerungssysteme
Institutsleiter: Prof. Dr.-Ing. Sören Hohmann

Vorwort

> Der heutige Ingenieurberuf fordert nicht nur ein fundiertes theoretisches Verständnis des jeweiligen Fachgebietes, sondern ebenso Kenntnisse im Umgang mit entsprechenden Softwarewerkzeugen, die die Arbeit oftmals erleichtern. In der Regelungstechnik, wie auch in vielen verwandten Fachgebieten, hat sich heutzutage MATLAB/SIMULINK als das „State-of-the-art"-Instrument entwickelt.

Da sich MATLAB zusammen mit dem ergänzenden Simulationspaket SIMULINK insbesondere auch für die Modellierung, Simulation und Analyse von Regelkreisen sowie zum Reglerentwurf hervorragend eignet, wird im Rahmen der Vorlesung Systemdynamik und Regelungstechnik (SRT), welche vorwiegend von Studierenden der Elektro- und Informationstechnik (ETIT) am Karlsruher Institut für Technologie (KIT) besucht wird, sowie in den begleitenden Übungs- und Tutoriumsveranstaltungen auch der Umgang mit MATLAB/SIMULINK vermittelt.
Dieses Kompendium ist daher als Begleitmaterial zur SRT-Vorlesung zu verstehen. Darüber hinaus ist es für Studierende auch in weiterführenden ETIT-Veranstaltungen als Nachschlagewerk bei der Verwendung von MATLAB und SIMULINK in systemtheoretischen Fragestellungen geeignet. Aus diesem Grund werden nicht alle aufgeführten Befehle und Funktionen zwingend für SRT benötigt. Gleichzeitig erhebt die vorgestellte Auswahl keinesfalls den Anspruch der Vollständigkeit in Bezug auf den Einsatz von MATLAB in der Regelungstechnik. Weiterführende Informationen hierzu finden sich u.a. in [1], [2] und [3].

In diesem Kompendium werden dem Leser theoretische Kenntnisse in höherer Mathematik sowie in Signaltheorie und einfacher Programmierung unterstellt. Systemtheoretische Kenntnisse, die zum Verständnis nötig sind, werden im Laufe des Semesters vermittelt. Als Teil der Schriftenreihe *Einführung in Matlab für Studierende der Fakultät ETIT* baut dieses Dokument dabei explizit auf dem ersten Teil *Lineare elektrische Netze* [4] auf. Dort finden sich auch Hinweise zum Bezug der Software am KIT. Sofern nicht anders angegeben beziehen sich die Ausführungen in diesem Kompendium auf die Version MATLAB R2012b. Verwendete Toolboxen sind SIMULINK, sowie die *Control Systems Toolbox*.

Inhaltsverzeichnis

Teil I

MATLAB in der Regelungstechnik

1

Lineare Übertragungsglieder

Für lineare Übertragungsglieder gibt es in MATLAB spezielle Datentypen. Hiermit lassen sich eine Vielzahl von unterschiedlichen LTI-Gliedern als Objekte definieren, welche dann im Workspace wie Variablen vorliegen. Dabei sind sowohl zeitkontinuierliche Systeme (mit der komplexen Frequenzvariable s) als auch zeitdiskrete Systeme (mit der komplexen Frequenzvariable $z = e^{Ts}$ und der Abtastzeit T) erzeugbar. Einige der bekannten Operationen, wie beispielsweise die Addition, lassen sich ebenso auf solche LTI-Objekte anwenden. Darüber hinaus stellt die *Control Systems Toolbox* spezielle Befehle für LTI-Objekte bereit.

Die Definition der Übertragungsfunktion eines LTI-Systems kann in MATLAB mit Hilfe verschiedener Befehlen erfolgen. Je nach Anwendungsfall kann die eine oder andere Form vorteilhaft sein. Es ist ebenso möglich, die verschiedenen Darstellungen ineinander zu konvertieren.

1.1 Transfer-Function

Mithilfe des Befehls `tf` lassen sich Übertragungsfunktionen anhand ihrer Zähler- und Nennerpolynome definieren. So wird z.B. die Übertragungsfunktion

$$G(s) = \frac{2s+1}{s^2+3s+4} = \frac{B(s)}{A(s)}$$

mit den Zähler- und Nennerpolynomen $B(s)$ und $A(s)$ folgendermaßen definiert:

```
>> B = [2 1]; A = [1 3 4];
>> sys = tf(B,A)
```

oder direkt

```
>> sys = tf([2 1],[1 3 4])
```

Dabei müssen der Funktion `tf` zwei Vektoren übergeben werden, welche die Koeffizienten des Zählerpolynoms $[b_m, \cdots, b_0] \overset{hier}{=} [2, 1]$ bzw. des Nennerpolynoms $[a_n, \cdots, a_0] \overset{hier}{=} [1, 3, 4]$ in absteigender Reihenfolge enthalten. Durch den Befehl wird das `tf`-Objekt `sys` im Workspace angelegt. Als Ausgabe erscheint die Übertragungsfunktion als Term:

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

s^2 + 3 s + 4
```

Zeitdiskrete Übertragungsfunktionen

Soll mit `tf` eine zeitdiskrete Übertragungsfunktion definiert werden, so ist als drittes Argument die Abtastzeit anzugeben.

```
>> sys = tf([3 0 0],[0.2 -1],0.05)

Transfer function:
  3 z^2
---------

0.2 z - 1

Sampling time (seconds): 0.05
```

Hierbei sind die ersten beiden Argumente wieder die Koeffizienten der Polynome in z in *absteigender* Reihenfolge $[a_n, \cdots, a_0] \mathrel{\widehat{=}} a_n z^n + \cdots + a_0$. Möchte man hingegen die Polynome in z^{-1} eingeben, so ist als weiteres Argument die Option `'Variable','z^-1'` anzugeben. Es ist zu beachten, dass MATLAB dann die Koeffizienten der Polynome in *aufsteigender* Reihenfolge der Potenzen von z^{-1}, also $[\tilde{a}_0, \cdots, \tilde{a}_n] \mathrel{\widehat{=}} \tilde{a}_0 + \cdots \tilde{a}_n z^{-n}$ erwartet. Da bei der Umwandlung mit der höchsten Potenz von z gekürzt wird, kommt es bei ungleichem Grad von Zähler- und Nennerpolynom zu Koeffizientenverschiebungen, welche berücksichtigt werden müssen. Um z.B. die gleiche Übertragungsfunktion wie in obigem Beispiel in z^{-1} zu erhalten, muss also Folgendes eingegeben werden:

```
>> sys = tf([3],[0 0.2 -1],0.05,'variable','z^-1')

Transfer function:
       3
-----------------

0.2 z^-1 - z^-2

Sampling time (seconds): 0.05
```

Hingegen liefert folgende Definition (gleiche Koeffizientenvektoren wie in Standarddarstellung) eine andere Übertragungsfunktion:

```
>> sys = tf([3 0 0],[0.2 -1],0.05,'variable','z^-1')

Transfer function:
    3
----------

0.2 - z^-1

Sampling time (seconds): 0.05
```

Rückgewinnung von Parametern einer Transfer-Function

Mit Hilfe des Befehls `tfdata` können die zu einer bereits im Workspace vorliegenden Übertragungsfunktion gehörenden Koeffizientenvektoren sowie ggf. die Abtastzeit zurückgewonnen und in Variablen gespeichert werden:

```
>> G = tf([8 5],[2 3 2]);
>> [B,A] = tfdata(G,'v')

B =

     0     8     5

A =

     2     3     2

>> mysys = tf([1 1],[0.3 0 1],0.1);
>> [num,den,T]=tfdata(mysys,'v')

num =

     0     1     1

den =

     0.3000        0     1.000

T =

     0.1000
```

Das optionale Argument `'v'` erzwingt dabei für Eingrößensysteme die Ausgabe der Koeffizienten als Vektoren. Ohne diese Option, bzw. im Mehrgrößenfall, liefert `tfdata` die Koeffizientenvektoren in einem *cell array* verpackt:

```
>> G = tf([8 5],[2 3 2]);
>> [B,A] = tfdata(G)

B =

    [1x3 double]


A =

    [1x3 double]

>> B{1,1}

ans =
     0     8     5
```

```
>> A{1,1}

ans =
     2     3     2
```

1.2 Zero-Pole-Gain

Neben `tf` besteht mit dem Befehl `zpk` eine weitere Möglichkeit, Übertragungsfunktionen zu definieren. Hierbei werden statt den Koeffizienten der Polynome direkt die Lage der Nullstellen und Pole sowie der Verstärkungsfaktor des Systems angegeben. Somit ist der Befehl insbesondere dann geeignet, wenn das zu definierende System bereits in faktorisierter Darstellung vorliegt.
Z.B. wird das System

$$G(s) = \frac{2s+3}{s^3+6s^2+8s} = 2\frac{s+1.5}{s(s+2)(s+4)},$$

welches offensichtlich eine Nullstelle bei -1.5 und drei Pole bei 0, -2 und -4 aufweißt, mit `zpk` folgendermaßen eingegeben:

```
>> G = zpk([-1.5],[0 -2 -4],2)
```

Das erste Argument bildet ein Vektor, welcher die Nullstellen auflistet, gefolgt von einem Vektor, der die Polstellen angibt und dem Verstärkungsfaktor. Weißt ein System keine Pole oder Nullstellen auf, so ist der entsprechende Vektor leer (`[]`) zu übergeben.
Als Ausgabe erscheint das System in faktorisierter Darstellung:

```
Zero/pole/gain:
  2 (s+1.5)
-------------
s (s+2) (s+4)
```

Wie man sieht, erzeugt der Befehl für reelle Nullstellen und Pole stets Faktoren der Form $(s-\tilde{s})$ in Zähler und Nenner, in denen die Nullstelle bzw. der Pol $\tilde{s}$ direkt abgelesen werden können. Für konjugiert komplexe Pol- oder Nullstellenpaare wird hingegen eine gemeinsamer Term ausgegeben.

```
>> G = zpk([],[-2+i*3 -2-i*3],5)

Zero/pole/gain:
       5
---------------
(s^2 + 4s + 13)
```

Es ist auch möglich, das System in Faktoren der Form $(1-Ts)$ auszugeben, in welchen statt der Lage der reellen Pole die zugehörigen Zeitkonstanten T bequem abgelesen werden können. Hierzu dient das optionale Argument `'DisplayFormat','time constant'`.

```
>> G = zpk([-1.5],[0 -2 -4],2,'DisplayFormat','time constant')

Zero/pole/gain:
 0.375 (1+0.6667s)
-----------------------
s (1+0.5s) (1+0.25s)
```

Dabei ist zu beachten, dass der angezeigte Verstärkungsfaktor (im Beispiel 0.375) durch die Umrechnung im Allgemeinen nicht mehr dem Wert entspricht, welcher in `zpk` eingegeben wurde (hier 2).

Ebenso wie bei `tf` können mit `zpk` auch zeitdiskrete Übertragungsfunktionen definiert werden, indem als viertes Argument die Abtastzeit angegeben wird.

```
>> G = zpk([1.25],[0.8],1.6,0.01)

Zero/pole/gain:
1.6 (z-1.25)
------------
  (z-0.8)

Sampling time (seconds): 0.01
```

Weiterhin können auch für Zero-Pole-Gain-Objekte bei bereits im Workspace vorhandenen Systemen die Parameter zurückgewonnen werden.

```
>> sys = zpk([1 -2],[-4+i*2 -4-i*2 -1],3);
>> [z, p, k] = zpkdata(sys,'v');

z =

     1
    -2

p =

  -4.0000 + 2.0000i
  -4.0000 - 2.0000i
  -1.0000

k =

     3
```

Für das optionale Argument `'v'` gelten die gleichen Aussagen wie unter Abschnitt 1.1.

1.3 State-Space

Eine weitere Darstellungsform von LTI-Systemen bildet die Beschreibung im Zustandsraum. Zustandsraummodelle (ZRM) sind nicht Bestandteil von SRT, finden jedoch in der

Regelungstechnik häufig Anwendung.

Ein ZRM der Form

$$\underline{\dot{x}} = \underline{A} \cdot \underline{x} + \underline{B} \cdot \underline{u}$$
$$\underline{y} = \underline{C} \cdot \underline{x} + \underline{D} \cdot \underline{u}$$

mit den Zustandsgrößen $\underline{x}$ und den Ein- und Ausgängen $\underline{u}$ bzw. $\underline{y}$ wird mit dem Befehl `ss` definiert. Als Argumente werden hierfür die Systemmatrix $\underline{A}$, die Eingangsmatrix $\underline{B}$, die Ausgangsmatrix $\underline{C}$ und die Durchgriffsmatrix $\underline{D}$ in den passenden Dimensionen übergeben.

```
>> A = [1 2;0 -3]; B = [2;5]; C = [0.5 1.25]; D = 0;
>> mysys = ss(A,B,C,D)

a =
       x1  x2
   x1   1   2
   x2   0  -3

b =
       u1
   x1   2
   x2   5

c =
         x1    x2
   y1   0.5  1.25

d =
       u1
   y1   0

Continuous-time state-space model.
```

Wiederum können durch Angabe der Abtastzeit als weiteres Argument auch zeitdiskrete Zustandsraummodelle realisiert werden. Alle Parameter des State-Space-Objekts können mittels `ssdata` zurückgewonnen werden.

```
>> A = [1 2 0;0 -3 -1; 0 1 2]; B = [2 0;0 5;-2 0]; C = [0 -1 2]; D = [0 0];
>> sys = ss(A,B,C,D,0.1);
>> [A B C D Ts] = ssdata(sys)

A =

     1     2     0
     0    -3    -1
     0     1     2

B =
```

```
     2     0
     0     5
    -2     0

C =

     0    -1     2

D =

     0     0

Ts =

    0.1000
```

1.4 Frequency-Response

Neben den bereits vorgestellten Methoden zur Definition von LTI-Systemen, existiert mit dem Befehl `frd` auch die Möglichkeit, ein System anhand seines Freqeunzganges zu definieren.

$$G(j\omega) = A(\omega) \cdot e^{j\varphi(\omega)}$$

Diese Darstellung bietet zwar weniger Einblicke in die Eigenschaften des Systems, kann aber trotzdem hin und wieder nützlich sein, z.B. wenn der Frequenzgang eines unbekannten Systems gemessen werden kann.
Um ein `frd`-Objekt zu erstellen, muss zunächst der Vektor der Stützpunkte angegeben werden, also jener Frequenzen ω, an denen der Frequenzgang eingegeben bzw. ausgewertet wird. Wie von Bode-Diagrammen bekannt, bietet es sich hierbei an, eine logarithmische Einteilung zu wählen. Dies ist jedoch nicht zwingend erforderlich. Der Frequenzgang selbst wird als Vektor gleicher Länge mit komplexen Zahlen definiert. So beschreibt er für jeden Frequenzpunkt den komplexen Verstärkungsfaktor des Systems.

```
>> freq = logspace(1,2);
>> resp = 0.05*freq.*exp(i*2*freq);
>> sys = frd(resp,freq)
```

Als Ausgabe erscheint eine tabellarische Auflistung aller Stützpunkte des Frequenzganges:

```
 Frequency(rad/s)          Response
 ________________          ________

      10.0000          0.2040 + 0.4565i
      10.4811         -0.2703 + 0.4490i
      10.9854         -0.5492 + 0.0112i
      11.5140         -0.2930 - 0.4955i
      12.0679          0.3276 - 0.5067i
```

...

```
       91.0298          4.4985 - 0.6925i
       95.4095         -3.2613 + 3.4816i
      100.0000          2.4359 - 4.3665i

Continuous-time frequency response.
```

Wie bei den übrigen LTI-Objekten können auch `frd`-Objekte für zeitdiskrete Systeme durch Angabe der Abtastzeit als weiterem Argument erstellt werden. Auch die Extraktion von Stützfrequenzen, Frequenzgang und ggf. Abtastzeit durch den Befehl `frdata` erfolgt analog zu den vorhergehenden Abschnitten.

1.5 Erweiterte Eigenschaften linearer Übertragungsglieder

1.5.1 Mehrgrößen-Systeme

Die bisher vorgestellten Methoden zur Definition von LTI-Systemen lassen sich auch auf Mehrgrößensysteme anwenden, d.h. Systeme mit mehreren Ein- und/oder Ausgängen (MIMO).
Während MIMO-Systeme in Zustandsraumdarstellung durch die entsprechenden Dimensionen der B- bzw. C-Matrix automatisch entstehen, können Frequency-Response-Models für Mehrgrößensysteme definiert werden, wenn für alle Übertragungspfade der gleiche Frequenzvektor verwendet wird. Das erste Argument des `frd`-Befehls wird dann als multidimenionaler Array der entsprechenden Ein-Ausgangs-Dimensionen übergeben:

```
freq = logspace(1,2);
resp(1,1,:) = 0.05*freq.*exp(i*2*freq);
resp(1,2,:) = 0*freq+1;
resp(1,3,:) = -0.1*freq.*exp(-i*4*freq);
sys = frd(resp,freq); % erzeugt ein Frequency-Response-Model mit 3
                      % Eingängen und einem Ausgang (1x3)
```

Für Übertragungsfunktionen (`tf`, `zpk`) gibt es zur Definition von Mehrgrößensystemen generell zwei Herangehensweisen.
Zunächst können mehrere SISO-Übertragungsfunktionen durch einfache Konkatenation zu Übertragungs-Matrizen erweitert werden.

```
>> G1 = tf(1,[1 2]); G2 = tf([1 1],[1 0]);
>> sys = [G1; G2] % erzeugt ein 2x1-MIMO-System

Transfer function from input 1 to output...
        1
 #1:  -----
      s + 2

      s + 1
 #2:  -----
        s
```

Daneben besteht die Möglichkeit direkt Übertragungsmatrizen zu definieren. Hierbei werden die Vektoren der Zähler- und Nennerkoeffizienten (bei `tf`) bzw. der Pole und Null-

stellen (bei `zpk`) in cell arrays entsprechender Ein-Ausgangs-Dimensionen (Anzahl der Ausgänge x Anzahl der Eingänge) angeordnet und an die Befehle übergeben. Die Verstärkungsfaktoren bei `zpk` werden dagegen als Matrix übergeben.

```
num = {2, 1, [1 0]; [1 1], 0.5, [1 2]}
den = {[1 0], [1 1] [1 2 1]; [1 0 0], [2 3], [1 3 4]};
G = tf(num,den); % definiert 2x3-Übertragungsmatrix G(s)

z = {[];[-0.8]};
p = {[-1,0.75];[1.2 0.3]};
k = [2.5; 3.6];
R = zpk(z,p,k,0.1); % zeitdiskrete faktorisierte 2x1-Übertragungsmatrix
```

Mehrgrößensysteme sind übrigens kein Bestandteil von SRT.

1.5.2 Totzeit-Systeme

Matlab bietet für alle oben genannten Typen von LTI-Systemen auch die Möglichkeit, Totzeiten zu berücksichtigen. Dies kann mit Hilfe der Eigenschaften `'InputDelay'`, `'OutputDelay'` und `'ioDelay'` und der nachfolgenden Angabe der Totzeit(en) erfolgen. Für Eingrößensysteme als Übertragungsfunktion (`tf`, `zpk`) sowie als Frequenzgang (`frd`) sind dies Skalare und es spielt keine Rolle ob die Totzeit am Ein- oder Ausgang des Systems berücksichtigt wird.

```
>> sys1 = tf(1,[2 5],'InputDelay',1.8)

Transfer function:
                1
exp(-1.8*s) * -------
              2 s + 5

>> sys2 = tf(1,[2 5],'OutputDelay',1.8)

Transfer function:
                1
exp(-1.8*s) * -------
              2 s + 5
```

Bei Zustandsraummodellen muss hingegen darauf geachtet werden, ob eine Totzeit am Ein- oder Ausgang des Systems wirkt.

Für MIMO-Systeme sind die Totzeiten der einzelnen Ein- bzw. Ausgänge nach `'InputDelay'` und `'OutputDelay'` als Vektoren passender Dimension zu übergeben.

```
>> G1 = tf({1,[3 -8]},{[2 5],[1 2 0]},'InputDelay',[1; 0.8])

Transfer function from input 1 to output:
               1
exp(-1*s) * -------
            2 s + 5
```

```
Transfer function from input 2 to output:
               3 s - 8
exp(-0.8*s) * ----------
              s^2 + 2 s
```

Unterschiedliche Totzeiten für jeden Übertragungspfad sind auf diese Weise im Allgemeinen jedoch nicht realisierbar. Dies kann stattdessen kann mit `'ioDelay'`, gefolgt von einer Matrix, welche die Totezeiten aller Übertrgungspfade enthält, erreicht werden.

```
>> G2 = zpk({[],[],[1];[-2],[0],[]},{[-3 -4],[0],[-1];[0],[-1 -5],[-2]},[2 ...
    -4 0.5;1 10 -5],'ioDelay',[0 5 0.3;0.8 2.1 1.7])

Zero/pole/gain from input 1 to output...
           2
 #1:  ------------
      (s+3) (s+4)

                      (s+2)
 #2:  exp(-0.8*s) * ------
                        s

Zero/pole/gain from input 2 to output...
                   -4
 #1:  exp(-5*s) * --
                   s

                        10 s
 #2:  exp(-2.1*s) * ------------
                    (s+1) (s+5)

Zero/pole/gain from input 3 to output...
                    0.5 (s-1)
 #1:  exp(-0.3*s) * ----------
                      (s+1)

                     -5
 #2:  exp(-1.7*s) * ------
                    (s+2)
```

1.6 Nützliche Funktionen im Umgang mit LTI-Objekten

1.6.1 Umwandlung zwischen den unterschiedlichen Darstellungen

Wie bereits ewähnt können die verschiedenen Typen von LTI-Objekten ineinander umgewandelt werden. Hierzu werden die gleichen Befehle verwendet, die zur Definition der Objekte bereits vorgestellt wurden. Als Argument wird den Befehlen dann jedoch lediglich das in einer anderen Form existierende System übergeben.

```
>> sys_tf = tf([2 -4],[1 -3 2]); % System als tf definieren
```

```
>> sys_zpk = zpk(sys_tf)                    % umwandeln in zpk

Zero/pole/gain:
  2(s-2)
----------
(s+1)(s+2)

>> sys_ss = ss(sys_zpk)                     % umwandeln in ss

a =
        x1   x2
   x1    1    1
   x2    0    2

b =
        u1
   x1    0
   x2    2

c =
        x1   x2
   y1   -1    1

d =
        u1
   y1    0

Continuous-time state-space model.

>> sys_frd = frd(sys_ss,logspace(1,2)) % umwandeln in frd

  Frequency(rad/s)            Response
  ----------------            --------
       10.0000        -1.980e-002 - 0.1980i
       10.4811        -1.804e-002 - 0.1891i
       10.9854        -1.644e-002 - 0.1806i
  ...
```

Bei der Konvertierung in ein `frd`-Objekt muss als zusätzliches Argument natürlich noch der Frequenzvektor der Stützstellen angegeben werden, da diese in den übrigen Sytembeschreibungen nicht definiert sind. Eine Umwandlung von `frd`-Objekten in andere LTI-Systeme ist hingegen nicht möglich.

1.6.2 Minimalrealisierung

Bei der Multipikation von Übertragungsfunktionen in Matlab werden Pole und Nullstellen aufgrund numerischer Fehler manchmal zunächst nicht vollständig gekürzt, sodass die resultierenden Übertragungsfunktionen unnötig kompliziert erscheinen und eine größere Anzahl an Polen und Nullstellen vortäuschen, als tatsächlich vorhanden sind. Eine sogenannte Minimalrealisierung, bei der sehr nahe beieinander liegende Pole und Nullstellen gekürzt werden, lässt sich mit Hilfe des Befehls `minreal` erzeugen.

```
>> G = zpk([2],[-0.1 -2 -5],0.1);
>> R = tf([1 7 10],[1 0]);
>> Fo = G*R

Zero/pole/gain:
0.1 (s-2) (s+2) (s+5)
---------------------
s (s+0.1) (s+2) (s+5)
>> Fo_min = minreal(Fo)

Zero/pole/gain:
0.1 (s-2)
---------
s (s+0.1)
```

1.6.3 Wechsel zwischen zeitkontinuierlicher und zeitdiskreter Darstellung

In der Praxis ist gelegentlich der Übergang von zeitkontinuierlichen auf zeitdiskrete Systeme nötig, z.B. bei der Beschreibung von Abtastregelkreisen. Dies ist in MATLAB ebenso einfach möglich, wie der umgekehrte Weg.
Mit Hilfe des Befehls `c2d` und der Angabe der Abtastzeit kann ein zeitkontinuierliches System diskretisiert werden.

```
>> mysys = tf(5,[1 2 9]);
>> mysys_d = c2d(mysys,0.02)

Transfer function:
0.0009865 z + 0.0009734
-----------------------
z^2 - 1.957 z + 0.9608

Sampling time (seconds): 0.02
```

Hierbei stehen verscheidene Diskretisierungsmethoden zur Verfügung. Neben der Default-Einstellung `'zoh'`, welche einem echten Abtastvorgang (mthematisch beschreibbar über die $\mathfrak{z}$-Transformation) entspricht, können z.B. `'foh'` oder `'tustin'` für eine Abtatung erster Ordnung oder die Tustin-Approximation gewählt werden. Je nach Ausgangssystem können dadurch leicht unterschiedliche zeitdiskrete Systeme entstehen. So ergibt die Wahl von `'tustin'` für obiges Beispiel

```
>> mysys_d2 = c2d(mysys,0.02,'tustin')

Transfer function:
0.0004898 z^2 + 0.0009795 z + 0.0004898
---------------------------------------
        z^2 - 1.957 z + 0.9608

Sampling time (seconds): 0.02
```

Der umgekehrte Weg (von zeitdiskreter Übertragungsfunktion zu zeitkontinuierlichem Pendant) kann mit dem Befehl `d2c` erreicht werden.

```
>> sys = tf(0.15,[1 -0.6],0.1)

Transfer function:
 0.15
-------
z - 0.6

Sampling time (seconds): 0.1
>> sys_k = d2c(sys)

Transfer function:
  1.916
---------
s + 5.108
```

Hierbei sollte jedoch beachtet werden, dass die resultierenden kontinuierlichen Systeme einerseits aufgrund des Abtasttheorems nach *Shannon* nicht eineindeutig sind und sich andererseits die Modellordnung unter gewissen Umständen verändern kann.

2

Analyse und graphische Ausgaben mit der *Control Systems Toolbox*

2.1 Betrachtung im Zeitbereich

2.1.1 Ermittlung von Sprungantworten

Der Zeitverlauf von Sprungantworten wird in MATLAB mit der Funktion `step` ermittelt und grafisch dargestellt. Der Funktion muss ein LTI-System übergeben werden. Beispielsweise erzeugen folgende 2 Zeilen die Sprungantwort aus Abbildung 2.1:

```
>> sys = tf([3],[1 2 5]);
>> step(sys)
```

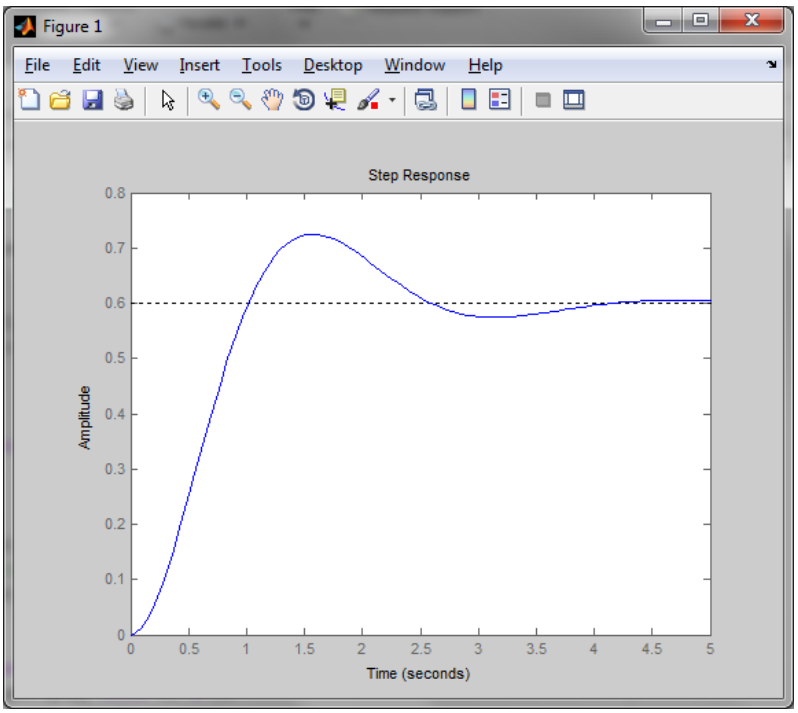


Abbildung 2.1. Anwendung des Befehls `step`

Zusätzlich kann eine Simulationsdauer mit `step(sys,t)` festgelegt werden. Der Funktion können auch mehrere LTI-Systeme als Eingabeparameter übergeben werden, deren Sprungantworten dann alle im selben Koordinatensystem abgebildet werden. Der Befehl

```
>> [y,t] = step(sys)
```

übergibt dem Vektor y Werte für die Sprungantwort zu den Zeitpunkten die im Vektor t abgelegt werden. Bei obigem Aufruf der Funktion wird die Sprungantwort jedoch nicht mehr geplottet, sondern nur die Werte für `y` und `t` ausgegeben,

2.1.2 Ermittlung von Impulsantworten

Impulsantworten werden in MATLAB sehr ähnlich ermittelt und dargestellt. Hierzu muss lediglich der Befehl `impulse` verwendet werden. Die Anwendung des Befehls erfolgt analog zu `step`.

2.1.3 Ermittlung von Antworten auf beliebige Eingangssignale

Mit Hilfe der Funktion `lsim` können beliebige Signale an den Eingang des LTI-Systems gelegt werden. Die Antwort des Systems wird geplottet. Der Funktion müssen neben dem System `sys` ein Vektor `t` mit beliebigen auszuwertenden Zeitpunkten sowie ein Vektor `u` mit den Werten des Eingangssignals zu den angegebenen Zeitpunkten übergeben werden. Im zeitdiskreten Fall muüssen die Abtastzeitpunkte des Systems mit denen des vorgegebenen Zeitvektors `t` übereinstimmen. Folgende Zeilen erzeugen die Darstellung nach Abbildung 2.2:

```
>> sys = tf([3],[1 2 5]);
>> t = linspace(0,20,200);
>> u = [-0.5*ones(size(t(1:50))), 0.5*ones(size(t(51:100))), ...
   -cos(t(101:end))];
>> lsim(sys,u,t)
```

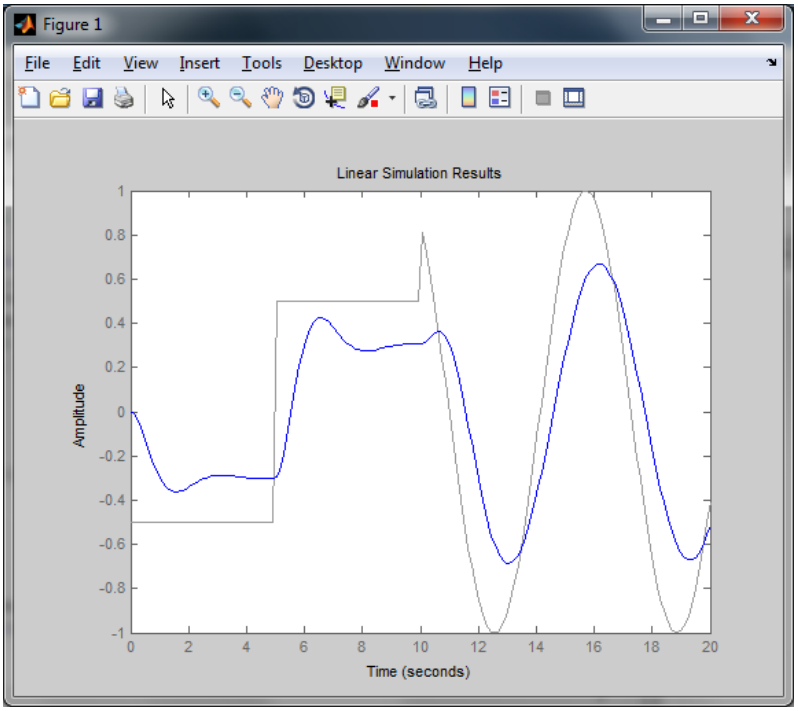


Abbildung 2.2. Anwendung des Befehls `lsim`

2.1.4 Anfangsauslenkung eines Zustandsraummodells

Ein Zustandsraummodell, das von außen zwar keine Anregung erhält, aber intern mit Anfangswerten belegt ist, kann einen von Null verschiedenen Verlauf der Zustandsgrößen und des Ausgangssignals aufweisen. Um deren Verlauf ermitteln zu können, bietet Matlab die Funktion `initial` an, der bei ihrem Aufruf sowohl das Zustandsraummodell `sys` als auch die Anfangswerte `x0` übergeben werden müssen. Beispielsweise erzeugt das Zustandsraummodell

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -0.5572 & -0.7814 \\ 0.7814 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$
$$y = \begin{bmatrix} 1.9691 & 6.4493 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

mit dem Anfangswert $\underline{x}(0) = [1, 0]^T$, nach Eingabe in Matlab

```
>> A = [-0.5572 -0.7814;0.7814 0]; C = [1.9691 6.4493]; x0 = [1;0];
>> sys = ss(A,[],C,[]);
>> initial(sys,x0)
```

den Verlauf nach Abbildung 2.3.

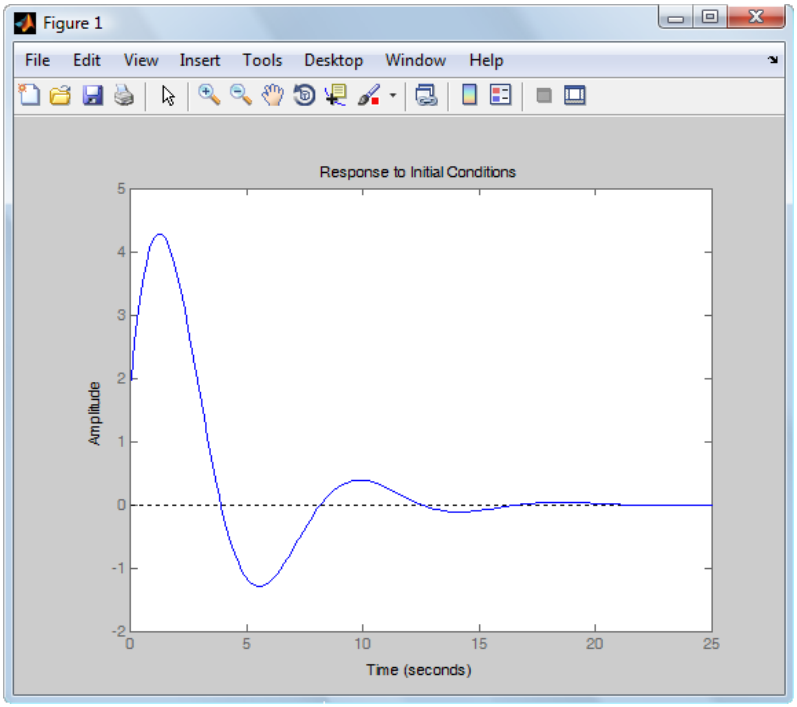


Abbildung 2.3. Anwendung des Befehls `initial`

2.2 Betrachtung im Frequenzbereich

Für die Analyse und Synthese von LTI-Systemen stehen im Frequenzbereich einige Funktionen zur Verfügung:

2.2.1 Pole und Nullstellen

Die Funktion

```
>> pzmap(sys)
```

erzeugt einen Pol-Nullstellen-Plan des Systems. Die Pole werden als Kreuze und die Nullstellen als Kreise in der komplexen s-Ebene dargestellt. Durch die linksseitige Angabe von Variablen, werden die Pole und Nullstellen statt der graphischen Darstellung in diesen Variablen gespeichert:

```
[p,z]=pzmap(sys)
```

Folgendes Beispiel zeigt die Berechnung der Pole und Nullstellen einer zeitkontinuierlichen Übertragungsfunktion

```
>> sys = tf([2 1],[1 3 4]);
>>[p,z] = pzmap(sys)

p =
```

```
  -1.5000 + 1.3229i
  -1.5000 - 1.3229i


z =

   -0.5000

>>pzmap(sys)
```

Der letzte Befehl erzeugt die Darstellung aus Abbildung 2.4.

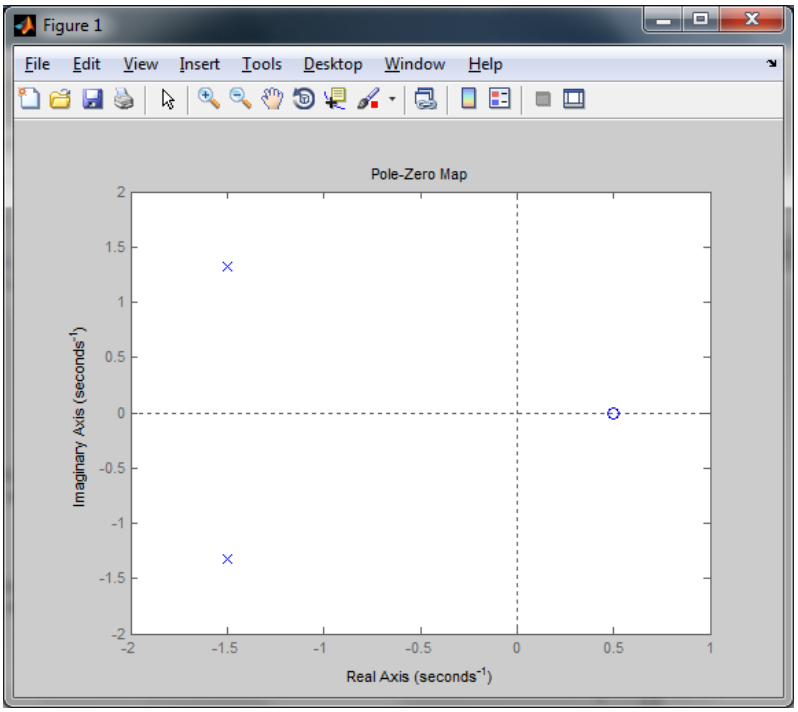


Abbildung 2.4. Anwendung des Befehls `pzmap`

Die Pole der Übertragungsfunktion können auch direkt über den Aufruf des Befehls

```
pole(sys)
```

ausgegeben werden. Ebenso können auch die Nullstelen der Übertragungsfunktion `sys` über den Befehl

```
zero(sys)
```

berechnet werden.

2.2.2 Stationäre Verstärkung

Beim Aufruf der Funktion

```
k = dcgain(sys)
```

wird der Wert der stationären Verstärkung k des LTI-Modells `sys` berechnet. Bei einer zeitkontinuierlichen Übertragungsfunktion entspricht dieser Wert dem Funktionswert bei $s = 0$.

2.2.3 Frequenzgang und Nyquist-Ortskurve

Der komplexe Freqeunzgang eines dynamischen Systems kann durch

```
>> G_f = evalfr(sys,f)
```

ausgewertet werden. Zurückgegeben wird der Wert des Frequenzgangs des Systems `sys` bei der Frequenz `f` als komplexe Zahl.
Interessiert der Frequenzgang für einen ganzen Frequenzbereich, hilft der Befehl `freqresp` weiter. Zurückgegeben werden die Werte des Frequenzgangs von `sys` als Vektor an den Frequenzen, die im Vektor `f_in` festgelegt werden, sowie die Werte von `f_in` selbst.

```
>> [resp,f_out] = freqresp(sys,f_in)
```

Der Frequenzgang lässt sich mittels `nyquist` auch graphisch ausgeben. An die Funktion können dabei ein beziehungsweise mehrere LTI-Systeme übergeben werden, z.B.:

```
>> sys1 = tf(3,[1,5]);
>> sys2 = tf([1 2],[1 4 3 1]);
>> nyquist(sys1, sys2)
```

Es wird die Nyquist-Ortskurve (Ortskurve des Frequenzgangs) berechnet und geplottet. Zusätzlich erscheint der kritische Punkt -1 als rotes Kreuz. Soll nur die halbe Ortskurve für ausschließlich positive Frequenzen angezeigt werden (wie in SRT üblich), so kann dies mittels Rechtsklick im Zeichnungsbereich und anschlieSSendem Abwählen der Option *Show>Negative Frequencies* erreicht werden (siehe Abbildung 2.5).

Zusätzlich kann der Parameter `w` übergeben werden, welcher die Grenzen des Frequenzbereichs $\omega = [\omega_{\min},\ \omega_{\max}]$ festlegt, über die geplottet werden soll. Statt der graphischen Ausgabe können mit diesem Befehl auch Real- und Imaginärteil des Frequenzgangs zu bestimmten Frequenzpunkten ausgegeben werden, wenn die Funktion mit linksseitigen Argumenten verwendet wird:

```
>> [re,im,w] = nyquist(sys)
```

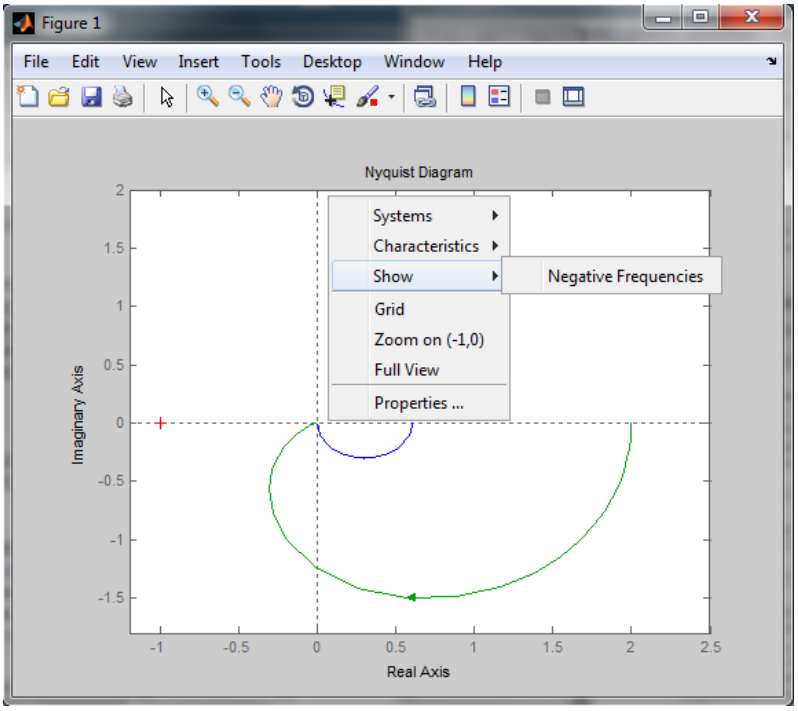


Abbildung 2.5. Anwendung des Befehls `nyquist`

2.2.4 Bodediagramm

Als Alternative zur Nyquist-Ortskurve kann der Frequenzgang auch getrennt in Amplitude und Phase geplottet werden. Dargestellt wird das Bode-Diagramm des übergebenen Systems `sys` nach Eingabe von:

```
>> bode(sys)
```

Abbildung 2.6 zeigt die erzeugte Darstellung für das System `sys2` aus dem vorhergehenden Beispiel.

Auch hier können Amplitude und Phase zu bestimmten Frequenzpunkten ausgegeben werden, wenn linksseitige Argumente angegeben werden.

```
>> [mag,phase,w] = bode(sys)
```

Die Funktion `margin(sys)` erzeugt ebenfalls das Bodediagramm des LTI-Systems, wobei die Amplitudenreserve `Gm` und die Phasenreserve `Pm` zusätzlich eingezeichnet und angegeben sind. Ebenfalls können Amplituden- und Phasenreserve des Systems sowie die zugehörigen Durchtrittsfrequenzen `Wcg` und `Wcp` als Vektoren ausgegeben werden.

```
>> [Gm,Pm,Wcg,Wcp] = margin(sys)
```

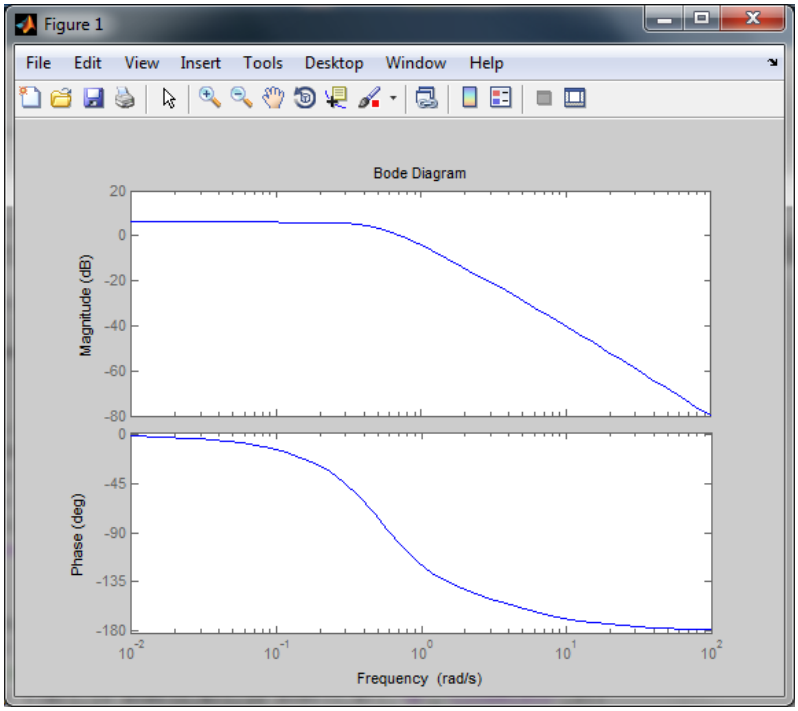


Abbildung 2.6. Anwendung des Befehls `bode`

2.2.5 Wurzelortskurven

Soll in MATLAB die Wurzelortskurve eines LTI-Systems berechnet und dargestellt werden, kann dies mit Hilfe des Befehls `rlocus(sys)` vorgenommen werden. Die Wurzelortskurve (WOK) erscheint in einem figure. Bei Klick mit der linken Maustaste erscheint ein Cursor, mit dem die WOK beliebig abgefahren werden kann. Zu jedem beliebigen Punkt können Verstärkung, Frequenz und aktuelle Position abgelesen werden.
Als Beispiel wird die Wurzelortskurve eines Systems mit einer Nullstelle bei -1 und drei Polstellen bei -1, -2, und -3 durch folgende Zeilen erzeugt (s. Abbildung 2.7):

```
>> sys = zpk([-1],[1 -2 -3],0.1);
>> rlocus(sys)
```

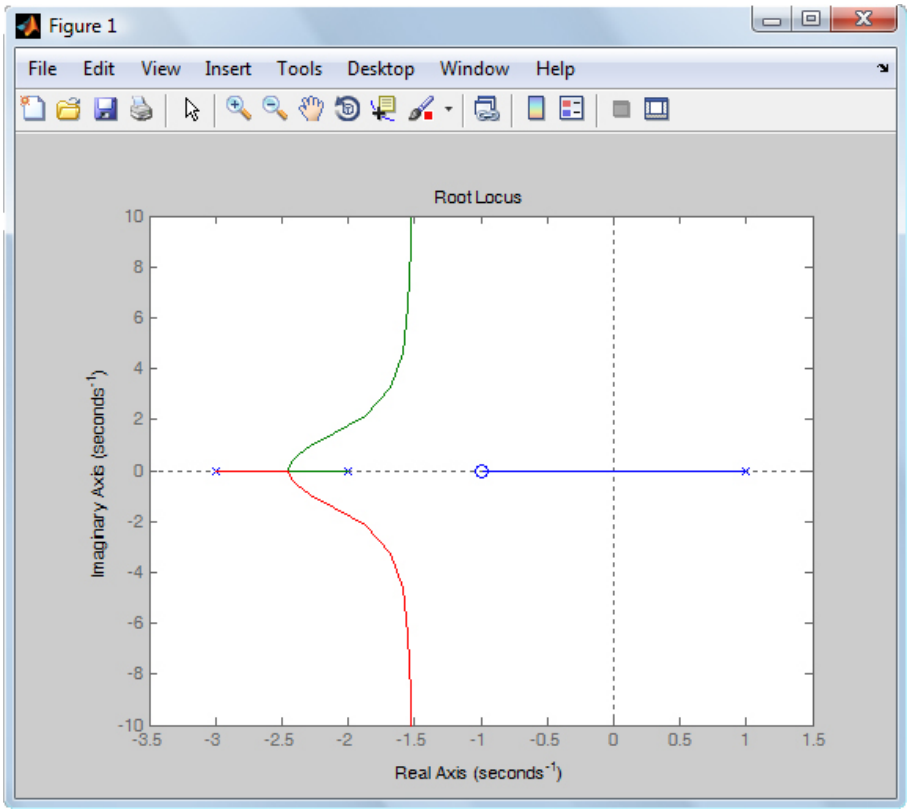


Abbildung 2.7. Wurzelortskurve

2.3 LTI-Viewer

Eine äußerst praktische Möglichkeit zur Darstellung und Analyse von LTI-Systemen bietet der LTI Viewer, welcher vielfältige Betrachtungen im Zeit- und Frequenzbereich zulässt. Dieser kann mit dem Befehl

```
>> ltiview
```

aufgerufen werden, woraufhin sich ein Fenster entsprechend Abbildung 2.8 öffnet. Über File>Import können beliebige Systeme in den LTI Viewer importiert werden. Unter Edit>Plot Configurations kann eine umfassende Analyse der Systeme im Frequenz- und Zeitbereich erfolgen. Abbildung 2.9 zeigt das Auswahlfenster, in dem die verschiedenen Plots ausgewählt werden können. Hierbei stehen unter anderem `'step'`, `'impulse'`, `'bode'` und `'nyquist'` zur Auswahl.

Außerdem bietet MATLAB zahlreiche Hilfestellungen bei der Analyse von LTI-Systemen im Zeit- und Frequenzbereich. Unter *control* findet sich in der MATLAB-Hilfe eine genaue Auflistung aller wichtigen Befehle im Zusammenhang mit LTI-Systemen. Genauere Informationen zu einem bestimmten Modelltyp (`tf`, `zpk`,...) findet sich bei Eingabe des Befehls `ltimodels('model')` beziehungsweise bei Eingabe des Befehls `ltiprops model`.

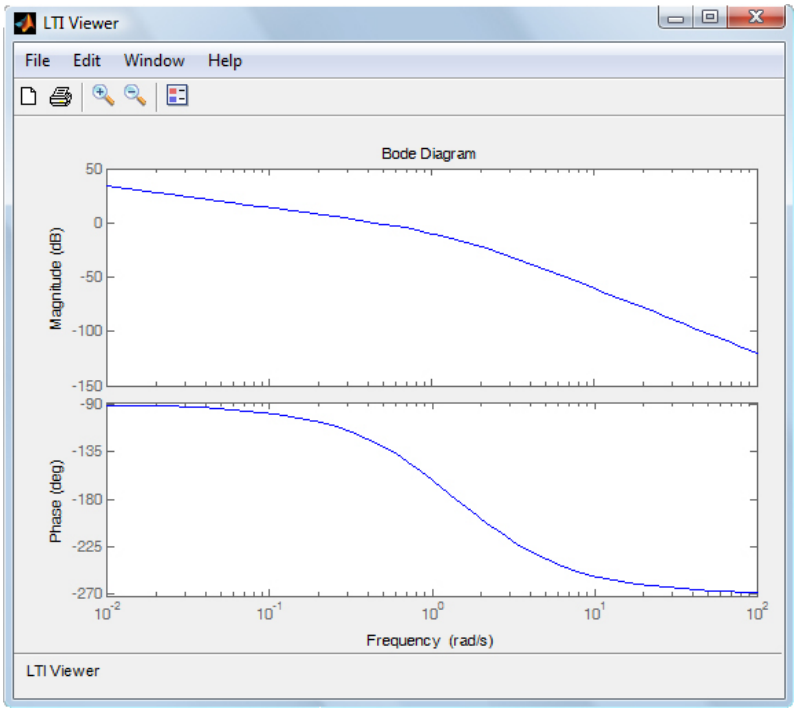


Abbildung 2.8. LTI Viewer

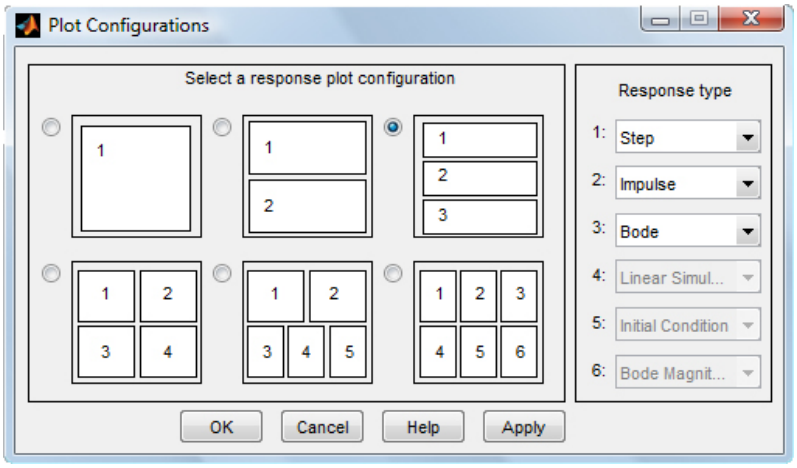


Abbildung 2.9. LTI Viewer

2.4 Anpassung von graphischen Ausgaben

Neben der in [4] beschriebenen Möglichkeit, graphische Ausgaben über den Plot-Browser anzupassen, können auch Befehle hierzu verwendet werden. Dies ist insbesondere dann hilfreich, wenn mehrere Graphiken mit identischen Einstellungen erstellt, oder eine graphische Ausgabe zu einem späteren Zeitpunkt erneut mit den gleichen Einstellungen erzeugt werden soll, da die Befehlsfolge dann als Code gespeichert werden kann.
Die graphischen Elemente in MATLAB sind streng objektorientiert aufgebaut und werden auch als *Handle Graphics* bezeichnet. Ein sogenanntes *handle* lässt sich dabei als ein

Zeiger auf ein konkretes Grafikobjekt verstehen. Die Beziehungen der einzelnen Objekte zueinander können folgender Grafik entnommen werden. Das Objekt *root* liegt nur ein-

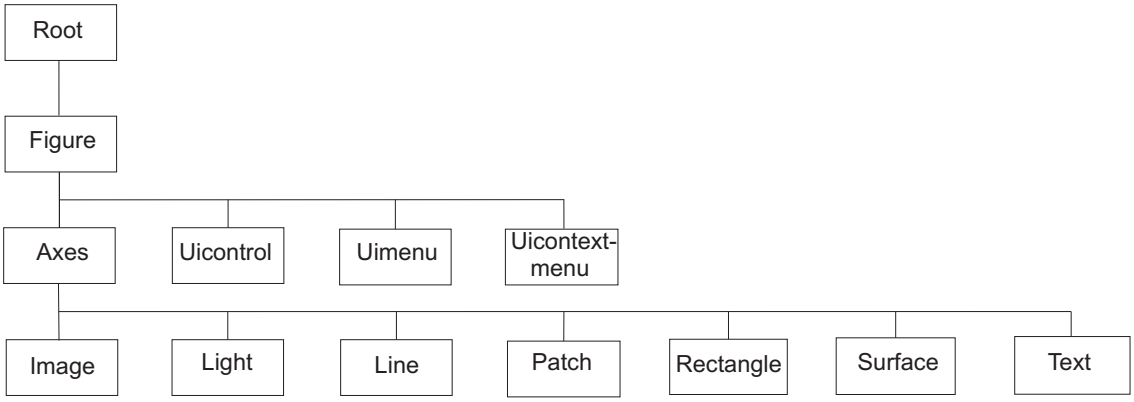


Abbildung 2.10. Struktur der Objektorientierung in MATLAB

fach vor und wird beim Start von MATLAB automatisch generiert. Untergeordnet sind die *figure*-Objekte, die gleichzeitig den Rahmen für alle grafischen Ausgaben in MATLAB darstellen. Generiert wird ein figure-Objekt mit dem Befehl `figure`. Dabei wird automatisch ein Index, das handle, zugeteilt. Wie alle Grafikobjekte hat somit auch jedes figure ein eindeutiges handle. Die handles der vorhandenen Grafikobjekte werden zunächst jedoch nicht im workspace angezeigt. Mit dem Befehl `gcf` wird das handle des aktuellen figure ausgegeben, mit `gca` das des aktuellen Achsensystems. Viele Funktionen, wie zum Beispiel `plot()`, geben das handle des erzeugten grafischen Objekts zurück, wodurch bei Zuweisung zu einer Variablen der spätere Zugriff ermöglicht wird. Das heißt, bei Eingabe der Befehlszeile

```
>> h=plot(x,y);
```

wird die Variable h angelegt, welche wie das handle auf das figure-Objekt zeigt, in dem geplottet wird. Der Befehl *plot* kann auch ohne explizite Zuweisung verwendet werden, d.h.

```
>> plot(x,y)
```

Hierbei wird das handle trotzdem erzeugt, aber keine Variable im workspace abgelegt, welche auf das Objekt verweist. Dies erschwert später den Zugriff auf das Objekt. Mit dem Befehl `figure(handle)` kann ein beliebiges figure aufgerufen werden. Zum Löschen eines figure kann der Befehl `clf` verwendet werden. Ein figure kann mit `close(handle)` geschlossen werden. Zum Schließen aller geöffneten figure verwendet man den Befehl `close all`. Alle Grafikbefehle beziehen sich immer auf das zuletzt erzeugte beziehungsweise angesprochene figure. In ein figure lässt sich nun beispielsweise ein einfacher Graph mittels `plot` darstellen oder z.B. auch eine spezielle graphische Ausgabe der Control Systems Toolbox (s. Abschnitt 2.1 und 2.2). Daneben gibt es zahlreiche weitere plot-Befehle, z.B. `semilogy`, `semilogx`, `loglog`, `fill`, `quiver`, `plot3`, `mesh`, etc. Weitere Informationen finden sich z.B. in [1] oder unter [5] und [6].

Jedem figure sind weitere Objekte als Kinder untergeordnet. Die Objekete UIcontrol, UImenu und UIcontextmenu dienen der Erstellung graphischer Benutzeroberflächen und sollen hier nicht weiter behandelt werden. Die Objekte *axes* dienen der Unterteilung eines figure in mehrere Regionen, die dann wiederum mit den Kinder-Objekten *image*, *light*, *line*, *patch*, *rectangle*, *surface* und *text* graphisch gestaltet werden können. Die Unterteilung des figure kann mit dem Befehl `subplot(N,M,k)` vorgenommen werden. Das figure wird in $N \times M$ Teilplots unterteilt. Die Indizierung der subplots erfolgt über den Index k. Mit dem Befehl `gca` kann der Index des aktuellen subfigures abgerufen werden.

Wie auch in anderen objektorientierten Programmiersprachen können Objekteigenschaften mit den Befehlen `get()` und `set()` gezielt abgerufen beziehungsweise verändert werden:
Mit

```
>>get(handle)
```

bzw.

```
>>get(gcf)
```

kann das aktuelle figure - oder das mit dem angegebenen handle - sowie dessen Eigenschaften abgerufen werden.

```
>>get(gca)
```

ruft dagegen das aktuelle Achsensystem sowie dessen Eigenschaften auf.

```
>>get(handle,'eigenschaft')
```

liefert nur die einzelne angegebene Objekteigenschaft, während mit

```
>>set(handle,'eigenschaft',wert)
```

einzelne Objekteigenschaften verändert werden können. Durch

```
>>reset(a)
```

werden die Eigenschaften des Objektes mit dem handle `a` zurückgesetzt und durch

```
>>delete(b)
```

wird das Objekt mit handle `b` gelöscht.

Daneben bietet MATLAB die Möglichkeit, Objekteigenschaften im *Property Editor* direkt zu verändern. Der Aufruf des Property Editors erfolgt mit dem Befehl *plottools*. Es stehen diverse Werkzeuge zur Verfügung. Unter anderem lassen sich mit Hilfe des Buttons *New Subplot* neue subplots in das bestehende figure integrieren. Die subplots können außerdem per Mausklick angewählt und die Größe und Position variiert werden. Zusätzlich sind die

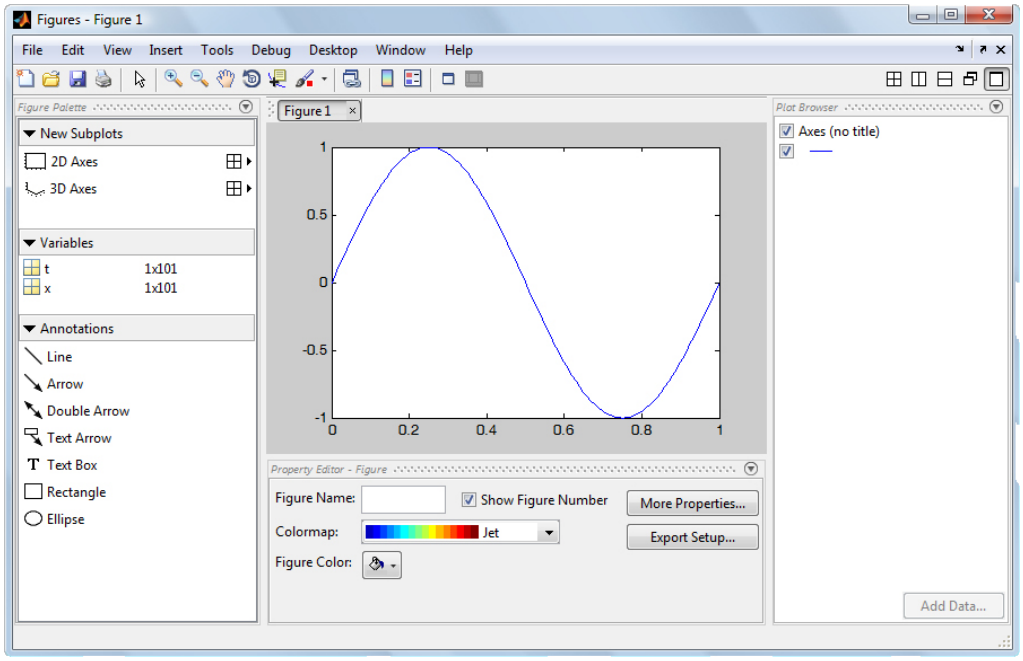


Abbildung 2.11. Property Editor

Achsen der subplots unten im *Property Editor* beliebig veränderbar. Durch einen Klick mit der rechten Maustaste auf den subplot, kann dieser über das Kontextmenü einfach wieder gelöscht werden. In der oberen Befehlsleiste stehen noch viele weitere Werkzeuge zur Verfügung: Zoomen und Verschieben des Bildausschnitts, Abfahren der Funktion mit einem *Data Cursor* zum Ablesen der präzisen Funktionswerte, Anpassen von Farbe und Stil der Graphen und Einfügen von Legenden. Mit Hilfe von File>Generate M-File kann das erstellte figure in Matlab-Quellcode umgewandelt werden, sodass eine Reproduktion der eingestellten Eigenschaften möglich ist.

Häufig ist es jedoch sehr aufwändig, sowohl die Namen der einzelnen Eigenschaften, die verändert werden sollen, als auch deren zulässigen Werte herauszusuchen und dann gezielt zu verändern. Eine einfache Methode, um auf schnellem Weg eine Reihe von Eigenschaften zu ändern, bietet die Funktion `uiinspect`, welche zwar nicht standardmäßig in Matlab enthalten ist, jedoch im *Central File Exchange* von Mathworks [7] heruntergeladen werden kann. Befindet sich die zugehörige Datei *uiinspect.m* im aktuell in Matlab geöffneten Arbeitsverzeichnis (oder in einem Ordner im allgemeinen Matlab-Such-Pfad) kann die Funktion für beliebige Figures verwednet werden.

Nach dem Aufrufen des Befehls

```
>> uiinspect(gcf);
```

öffent sich das Fenster aus Abbildung 2.12, in dem alle zur Verfügung stehenden Eigenschaften der geöffneten Plotts aufgelistet werden. Auf der linken Seite sind in einer Baumstruktur alle derzeit geöffneten Grafiken sowie die einzelnen Unterobjekte mit ihren jeweiligen handles erkennbar, während auf der rechten Seite die einzelnen Eigenschaf-

ten des jeweils ausgewählten Objektes angezeigt werden. Hierzu zählen beispielsweise die Farbe, die Linienstärke oder die Form von Markern.

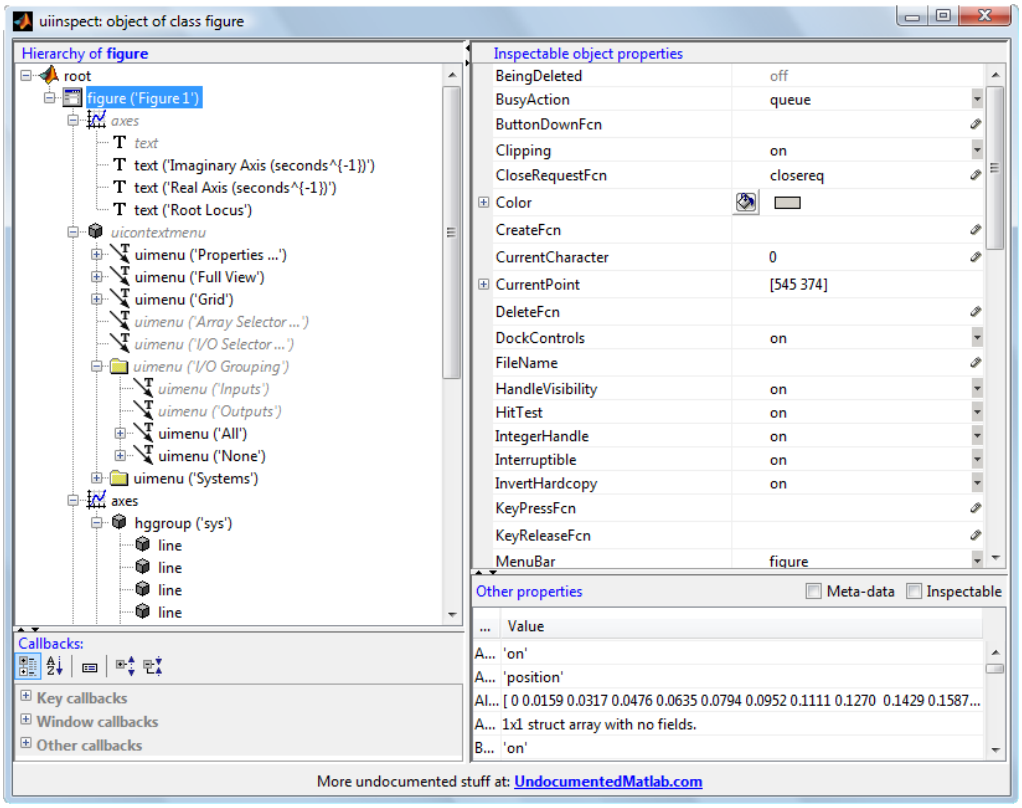


Abbildung 2.12. uiinspect(gcf)

3

Reglerentwurf mit der *Control Systems Toolbox*

3.1 SISO-Tool

Das SISO Design Tool dient zur Synthese komplexer Regelstrukturen mit Hilfe von MATLAB. Aufgerufen wird das Programm durch Eingabe von `sisotool`. Dabei stehen vier verschiedene Methoden zur Synthese zur Verfügung: das Wurzelortskurvenverfahren, der Entwurf im Frequenzbereich, der Entwurf mit Hilfe des Nichols-Diagramms und eine automatische Regeleinstellung. Nach Eingabe des Befehls `sisotool` öffnen sich zwei Fenster: der *Control and Estimation Tools Manager* und das *SISO Design for SISO Task* (siehe Abbildungen 3.1 und 3.2).

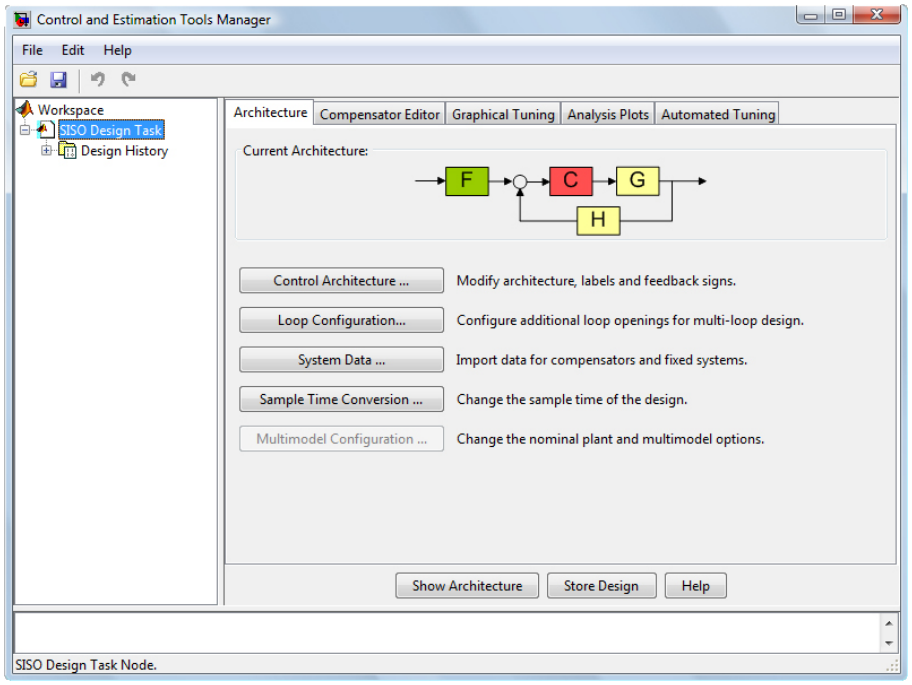


Abbildung 3.1. Control and Estimation Tools Manager

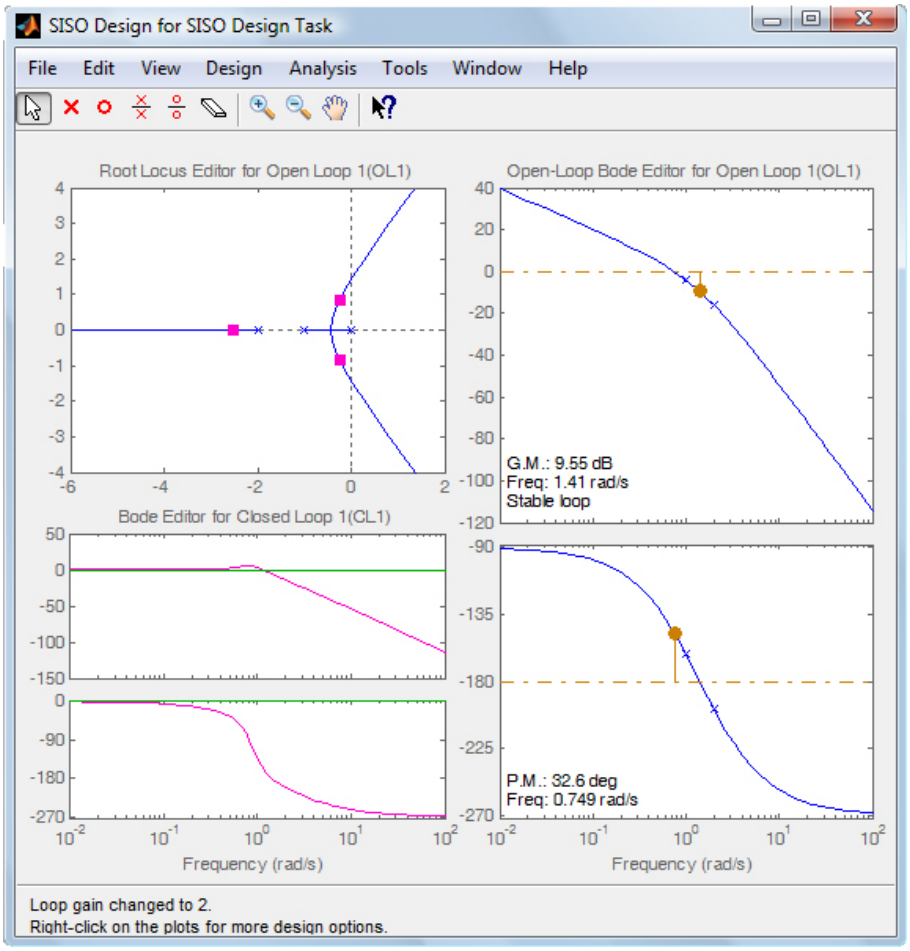


Abbildung 3.2. SISO Design for SISO Design Task

Der *Control and Estimation Tools Manager* zeigt die aktuelle Regelkreisstruktur samt der zugehörigen LTI-Objekte auf: der Strecke G, dem Regler C, der Rückführung H und des Vorfilters F. Diese LTI-Objekte müssen sich im workspace befinden und können auf zwei verschiedene Möglichkeiten an das SISO-Tool übergeben werden. Zunächst kann die Übergabe von Regler und Strecke direkt beim Aufruf des SISO-Tools mit Hilfe des Befehls `sisotool(G,C)` erfolgen. Des Weiteren kann die Übergabe jedoch auch im SISO-Tool direkt stattfinden. Über den Button *System Data* lässt sich ein neues Fenster öffnen. Hier kann mittels *Browse* für eines der anzugebenden LTI-Modelle ein LTI-Objekt aus dem Workspace ausgewählt werden. Allgemein dient der Bereich *Architecture* im *Control and Estimation Tools Manager* zur Einstellung der Regelkreisstruktur, im Bereich *Graphical Tuning* kann die Reglereinstellung grafisch vorgenommen werden. Mit Hilfe des *Compensator Editor* kann eine Einstellung des Reglers erfolgen. Per Rechtsklick im Fenster Dynamics können neue Pole und Nullstellen zur Übertragungsfunktion des Reglers hinzugefügt werden. Zusätzlich stehen die Funktion *Analysis Plot* zur Wahl der Regelkreis- und Eingangsgrößen und das *Automated Tuning*, sprich die automatische Reglersynthese, zur Verfügung. Innerhalb von *Architecture* können Regelstruktur und offener Regelkreis variiert werden, mit Hilfe von *Sample Time Conversion* lässt sich die Abtastperiode verändern.

Im *SISO Design for SISO Design Task*-Fenster werden zu dem parametrisierten Regelkreis die Wurzelortskurve sowie das Bode-Diagramme für offenen und geschlossenen Regelkreis dargestellt.
Soll der geschlossenen Kreise auf Stabilität untersucht werden, sind hierbei neben der Wurzelortskurve auch die im rechten Bodediagramm dargestellte Amplituden- und Phasenreserve sehr hilfreich. Bewegt man den Mauszeiger über eine der Kurven im Amplitudengang, so erscheint ein Handsymbol, mit dem man die Kurve nach oben und unten verschieben kann. Dies stellt die Veränderung der eingestellten Verstärkung ein, wodurch sich die Amplituden- und Phasenreserve entsprechend einstellen lassen. Ähnliches gilt auch für die Wurzelortskurve, in der sich die einzelnen Pole mit der Maus auf den vorgegebenen Kurven verschieben lassen. Den aktuell eingestellten Verstärkungfaktor kann ganz unten in Abbildung 3.2 abgelesen. Standardmäßig beträgt der Verstärkungsfaktor 1, bei Variation der Verstärkung wird dann z.B. *Loop gain changed to 2* angezeigt.

3.2 weitere Reglerentwurfsmethoden

Die *Control Systems Toolbox* stellt weitere spezielle Befehle zum Reglerentwurf zur Verfügung. Auf diese soll hier nicht näher eingegenagen werden. Näheres zu diesen weiterführenden Methoden findet sich z.B. in der MATLAB-Hilfe

Teil II

Simulink

4

Einführung in SIMULINK

SIMULINK ist ein Simulationstool innerhalb von MATLAB. Systemmodelle können signalflussbildähnlich in Blockform aufgebaut und simuliert werden. In der Bibliothek sind bereits viele Blöcke vordefiniert, die dann übernommen und parametrisiert werden können. SIMULINK wird z.B. durch Eingabe des Befehls `simulink` in MATLAB gestartet. Zunächst öffnet sich der SIMULINK Library Browser (siehe Abbildung 4.1). Es wird ersichtlich, dass

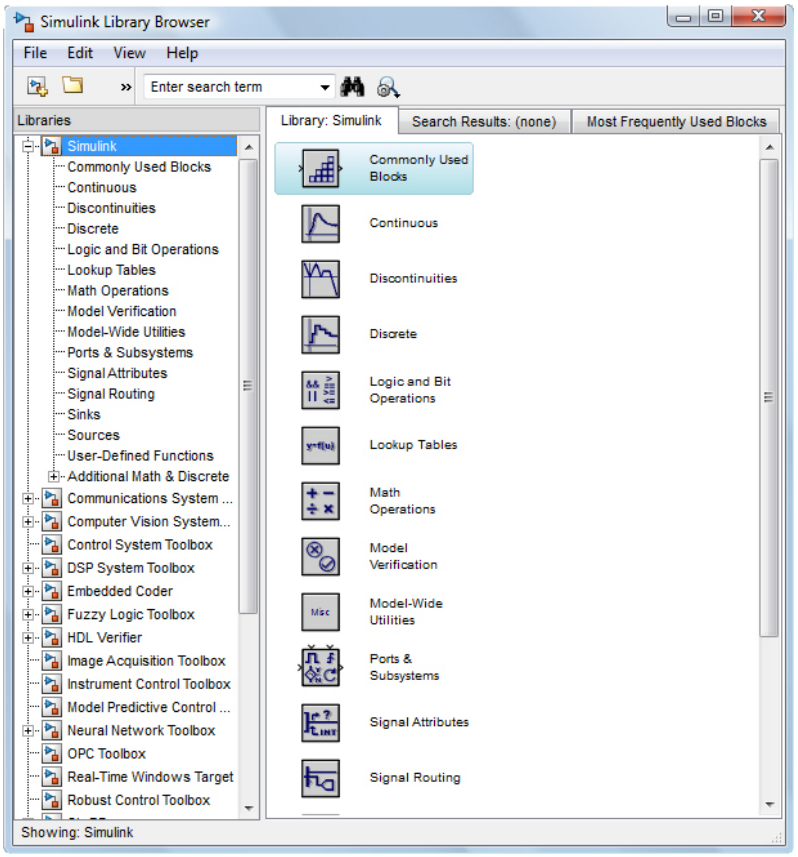


Abbildung 4.1. SIMULINK Library Browser

sich die Bibliothek in zahlreiche logische Unterbibliotheken unterteilt. Unter *File>New*

öffnet sich ein neues Simulationsfenster. Ab der MATLAB-Version 2012 werden die erstellten SIMULINK-Modelle mit der Dateiendung *.slx* abgespeichert, bei älteren Versionen mit *.mdl*. Die älteren Dateien können jedoch nach wie vor in MATLAB geöffnet werden und SIMULINK bietet auch die Möglichkeit, neu erstellte Modelle unter der alten Dateiendung abzuspeichern, da MATLAB-Versionen vor 2012 die neue Dateiendung nicht öffnen können.

Blöcke aus der Bibliothek können einfach ins Simulationsfenster hinübergezogen werden. Mit Hilfe der Shift-Taste lassen sich auch mehrere Blöcke gleichzeitig verschieben. Das Verbinden zweier Blöcke erfolgt mit gedrückter linker Maustaste und Nachfahren des gewünschten Verbindungsweges. Signalabzweigungen lassen sich erstellen, indem man von einem Signalzweig ausgehend mit gehaltener rechter Maustaste den Signalweg nachfährt. Durch Verschieben eines Blocks mit gehaltener rechter Maustaste wird dieser dupliziert. Mittels *Format>Flip Block* beziehungsweise *Format>Rotate Block* lassen sich Blöcke drehen bzw. spiegeln. Die Blöcke können nun per Doppelklick auf den jeweiligen Block parametrisiert werden. Je nach Block sind dabei unterschiedliche Parameter einstellbar. Neben Zahlenwerten können dabei für Parameter auch Variablen verwendet werden, welche zuvor in MATLAB definiert wurden und sich im workspace befinden. Hilfe zur Parametrisierung kann per Klick auf *help* angefordert werden. Testsignale finden sich ebenfalls in der Bibliothek unter *Simulink>Sources*. Um die Simulationsergebnisse anschauen beziehungsweise auswerten zu können, stehen zahlreiche Blöcke im Ordner *Simulink>Sinks* zur Verfügung. Der am häufigsten verwendete Block in diesem Zusammenhang ist *Scope*. Hier werden die verbundenen Signale über der Zeit aufgezeichnet und bei Doppelklick auf das Scope in einem neuen Fenster dargestellt (s. Abb. 4.2). Bei längeren Simulati-

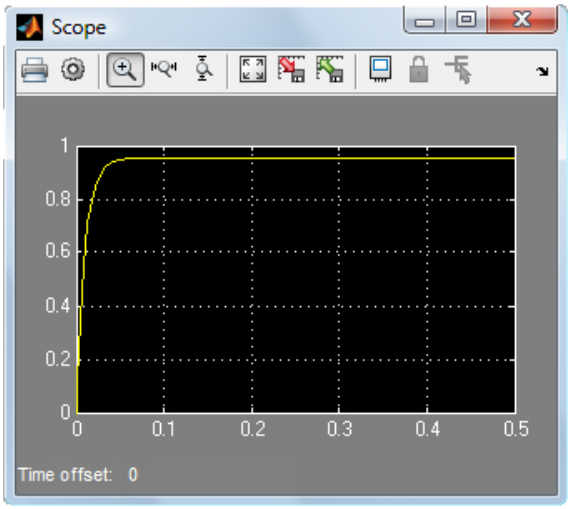


Abbildung 4.2. Das Scope-Fenster

onsdauern begrenzt SIMULINK die Anzeige automatisch auf die letzten 5000 Werte. Soll diese Voreinstellung deaktiviert werden, kann dies im Scopefenster unter dem Menüpunkt *Scope Parameters>History* erreicht werden.
Mit den Blöcken *To File* und *To Workspace* können die simulierten Daten in eine Datei beziehungsweise in den Workspace übernommen werden und stehen somit zur weiteren Verarbeitung zur Verfügung. Hierbei kann in den Eigenschaften des Blocks ausgewählt

werden, in welchem Datenformat die Simulationswerte an den Workspace ausgegeben werden. Die Standardeinstellung ist das Format Timeseries, das sowohl die diskreten Simulationszeitpunkte als auch die Werte des zugehörigen Ausgangs enthält.
Die Simulationsdauer kann man im Simulationsfenster oben rechts beliebig einstellen. Die Simulation wird per Klick auf die Play-Taste gestartet. Außerdem kann als Simulationsdauer auch `inf` für unendlich eingegeben werden, wobei die Simulation in diesem Fall erst bei Klick auf die Stopp-Taste endet. Allgemein können die Simulationsparameter über *Simulation>Configuration Parameters* eingesehen und verändert werden.
Abbildung 4.3 zeigt als Beispiel den Aufbau einer Standard-Regelkreisstruktur in Simulink.

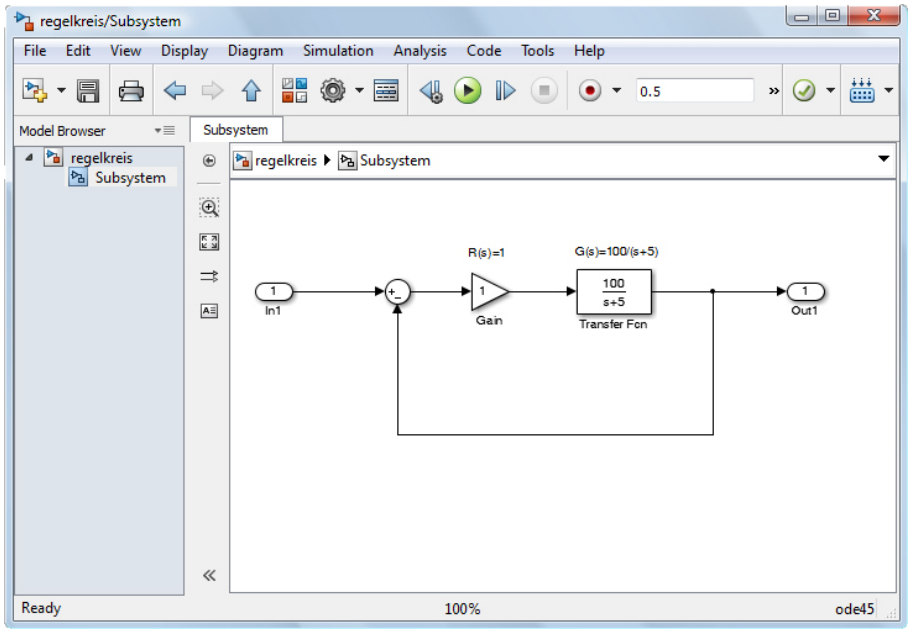


Abbildung 4.3. Einfacher Regelkreis in Simulink

Bei komplexeren Modellen kann es sinnvoll sein, Untersysteme einzuführen. Existiert bereits ein Modell, das zu einem Subsystem zusammengefasst werden soll, so kann einfach der entsprechende Ausschnitt der Schaltung markiert werden und dann über *Diagram>Subsystem & Model Reference>Create Subsystem from Selection* zu einem einzigen Block verbunden werden. Die Leitungen, die die markierte Schaltung verlassen, werden automatisch als *Input*- und *Output*-Ports deklariert. Seit der Version 2012 werden die Subsysteme eines Modells direkt im Model Browser auf der linken Seite des geöffneten Fensters angezeigt (s. Abbildung 4.3). Soll ein leeres Subsystem eingefügt werden, ist dies über den Library Browser unter dem Ordner *Simulink>Ports & Subsystems>Subsystem* möglich. Der Block lässt sich wie gewohnt ins Simulationsfenster ziehen. Per Doppelklick kann der Block gefüllt werden. Damit das Untersystem mit dem Gesamtsystem kommunizieren kann, müssen sogenannte *in*- und *out*-Blöcke verwendet werden. Diese finden sich im selben Ordner wie die Untersysteme.
Eine nützliche Funktion im Zusammenhang mit größeren Modellen, die Nichtlinearitäten enthalten, ist die Autolinearisierung. Diese kann in Simulink über *Analysis>Control Design>Linear Analysis* gestartet werden. In den Versionen vor 2012 ist das entsprechende

Tool unter *Tools>Control Design>Linear Analysis* zu finden. Es öffnet sich der Control and Estimation Tools Manger (s. Abbildung 4.4), in dem sich Arbeitspunkte berechnen lassen und das System um diese Arbeitspunkte linearisiert werden kann. Das resultierende System kann schließlich (z.B. als Übertragungsfunktion) in den Workspace exportiert und dort weiter verwendet werden.

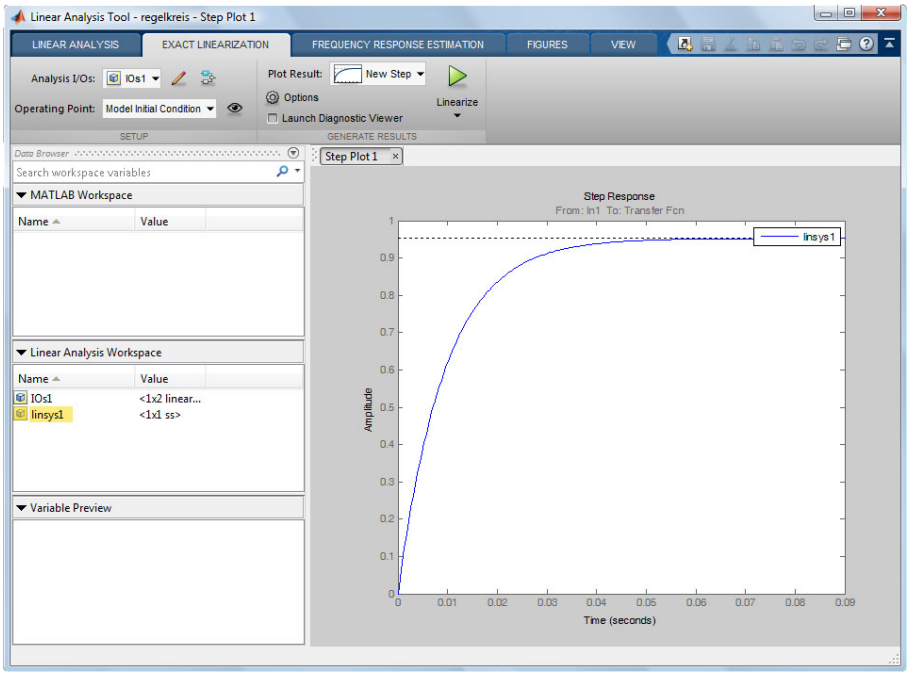


Abbildung 4.4. Control and Estimation Tools Manager der Autolinearisierung

Wird Hilfe bei der Anwendung von SIMULINK benötigt, kann diese unter dem Menüpunkt *Help>Using Simulink* aufgerufen werden. Anwendungsbeispiele finden sich unter *Help>Demos*.

Literatur

[1] Angermann, Anne et al.: *Matlab-Simulink-Stateflow: Grundlagen, Toolboxen, Beispiele.*

[2] Bode, Helmut: *Systeme der Regelungstechnik mit Matlab und Simulink: Analyse und Simulation.*

[3] Lutz, Holger und Wolfgang Wendt: *Taschenbuch der Regelungstechnik: mit Matlab und Simulink.*

[4] Krüger, Martin und Walther Schulze: *Einführung in Matlab für Studierende der Fakultät ETIT - Teil1: Lineare elektrische Netze.* Für Studierende des KIT verfügbar unter `https://ilias.studium.kit.edu` in der Rubrik Fakultät für Elektrotechnik und Informationstechnik > Institutsübergreifende Veranstaltungen > Einführung in Matlab für Studierende der Fakultät ETIT.

[5] Mathworks: *Internetseite des Herstellers.* `http://www.mathworks.com/products/matlab/`.

[6] Gomatlab: *Deutschsparachiges Forum rund um Matlab, Hilfe bei Problemen mit der Software.* `http://www.gomatlab.de/`.

[7] Mathworks: *UIINSPECT - display methods-properties-callbacks of an object.* `http://www.mathworks.com/matlabcentral/fileexchange/17935-uiinspect-display-methods-properties-callbacks-of-an-object`, 2013.