Übung 2 – MATLAB Control System Toolbox

1. Beschreibung linearer, zeitinvarianter Systeme (LTI):

- Single-Input/Single-Output (SISO) und Multiple-Input/Multiple-Output (MIMO)

1.1. Parametrische Beschreibung

- Übertragungsfunktion / Transfer Function (TF)

$$G(s) = \frac{num(s)}{den(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s^1 + b_0}{ab_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s^1 + a_0}$$

Zählerpolynom *num* mit Ordnung *m* und Nennerpolynom *den* mit Ordnung *n*

Befehl `tf(num, den)`: *num* und *den* als Vektoren mit Koeffizienten von s in absteigender Reihenfolge.:

```
>> tf([1 2],[1 0 10])
ans =
   s + 2
  --------
  s^2 + 10
Continuous-time transfer function.
>> help tf
```

- Nullstellen – Polstellen – Darstellung / Zero-Pole-Gain (ZPK)

$$G(s) = k \cdot \frac{(s - z_1) \cdot \ldots \cdot (s - z_{m-1}) \cdot (s - z_m)}{(s - p_1) \cdot \ldots \cdot (s - p_{n-1}) \cdot (s - p_n)}$$

k Verstärkungsfaktor (reell)

$z_1, \ldots, z_m$ Zaehler – Nullstellen (reell / konj. komplex)

$p_1, \ldots, p_n$ Nenner – Polstellen (reell / konj. komplex)

Befehl `zpk(z,p,k)`: Nullstellenvektor *z*, Polstellenvektor *p* und Verstärkungsfaktor *k*.

```
>> zpk([-6 1 1],[-5 1],3)
ans =
  3 (s+6) (s-1)^2
  ---------------
    (s+5) (s-1)
Continuous-time zero/pole/gain model.
>> help zpk
```

- Zustandsdarstellung / State – Space (SS)

  DGL 1. Ordnung für jedes Speicherelement (Integrator) → *n* DGLs 1. Ordnung statt eine DGL *n*–ter Ordnung

<table>
<tr>
<td>

Zustands – DGL:

$$\dot{x} = Ax + Bu$$

Ausgangsgleichung:

$$y = Cx + Du$$

</td>
<td>

x = Zustandsvektor $(N\,x \times 1)$
u = Eingangsvektor $(N\,u \times 1)$
y = Ausgangsvektor $(N\,y \times 1)$
A = Zustandsmatrix $(N\,x \times N\,x)$
B = Eingangsmatrix $(N\,x \times N\,u)$
C = Ausgangsmatrix $(N\,y \times N\,x)$
D = Durchgangsmatrix $(N\,y \times N\,u)$

</td>
</tr>
</table>

Befehl `ss(A,B,C,D)`

```
>> A=[1 2; 3 4]; B=[1 1; 0 1]; >> C=[0 1; 1 2; 3 1]; D=[0 0; 0 0; 0 0];
>> ss(A,B,C,D)
ans =
  A =
       x1  x2
   x1   1   2
   x2   3   4
  B =
       u1  u2
   x1   1   1
   x2   0   1
  C =
       x1  x2
   y1   0   1
   y2   1   2
   y3   3   1
  D =
       u1  u2
   y1   0   0
   y2   0   0
   y3   0   0

Continuous-time state-space model.
>> help ss
```

1.2. Nichtparametrische Beschreibung

- Frequenzgang – Daten – Modelle / Frequency Response Data (FRD)

  Frequenz – Daten aus Messung oder Simulation

  Sinus – Anregung: $y(t) = |G(\omega)| \cdot \sin(\omega t + \varphi(\omega))$

  Frequenzgangfunktion: $F(j\omega) = |F(j\omega)| \cdot e^{j\varphi(\omega)}$

  Betrag: $|F(j\omega)| = \sqrt{Re\{F(j\omega)\}^2 + Im\{F(j\omega)\}^2}$ Phase : $\arctan \dfrac{\mathrm{Im}\{F(j\omega)\}}{\mathrm{Re}\{F(j\omega)\}}$

Befehl `frd(ant; freq; eh)`:

`ant` = Vektor mit den komplexen Frequenzantworten
`freq` = Vektor mit gespeicherten Frequenzen
`eh` = Einheit der Frequenz ('rad/s' oder 'Hz')

```
>> freq = [0.01 0.1 1 10 100 1000 10000] ;
>> ant = (1-j*freq) ./(1+freq.^2);
>> sysfrd = frd(ant,freq,'Units','rad/s')

ant =

  0.9999 - 0.0100i   0.9901 - 0.0990i   0.5000 - 0.5000i   0.0099 - 0.0990i   0.0001 - 0.0100i   0.0000 - 0.0010i   0.0000 - 0.0001i

sysfrd =

    Frequency(rad/s)           Response
    ----------------           --------
           0.0100       9.999e-01 - 9.999e-03i
           0.1000       9.901e-01 - 9.901e-02i
           1.0000       5.000e-01 - 5.000e-01i
          10.0000       9.901e-03 - 9.901e-02i
         100.0000       9.999e-05 - 9.999e-03i
        1000.0000       1.000e-06 - 1.000e-03i
       10000.0000       1.000e-08 - 1.000e-04i
Continuous-time frequency response.

help frd
```

2. Umwandeln und Bearbeiten der Systeme

Vorrangliste: `FRD` → `SS` → `ZPK` → `TF`

Umwandeln: vorher: `sys = systf + tf(sysss)`

nachher: `sys = tf(systf + sysss)`

Explizite Umwandlung: Übergabe des Systems an **Befehle** `tf(sys)`, `ss(sys)`, `zpk(sys)` oder `frd(sys,freq)`

Umwandlung mit MATLAB – Befehlen:

```
zp2tf(z,p,k)          tf2zp(num,den)        tf2ss(num,den)
ss2tf(A,B,C,D,iu)     ss2zp(A,B,C,D,i)      zp2ss(z,p,k)
```

Arithmetische Operatoren:

Addition und Subtraktion: Parallelschaltung

$y = G_1 \cdot u + G_2 \cdot u$ ↔ `sys = sys1 + sys2`

Multiplikation: Reihenschaltung

$y = G_1 \cdot v = G_1 \cdot (G_2 \cdot u)$ ↔ `sys = sys1 * sys2`

Matrix-Inversion:

$u = G^{-1} \cdot y$ ↔ `sys = inv(sys1)`

links- und rechtsseitige Matrix – Division:

$G_1^{-1} \cdot G_2$ ↔ `sys1 \ sys2` $G_1 \cdot G_2^{-1}$ ↔ `sys1 / sys2`

Ansprechen der Ein- und Ausgänge mit `sys(i,j)`

Teilsystem:	extrahieren: `subsys = systf(1,2),`
	ändern: `systf(2,1) = subsys`
Ein-/Ausgänge:	löschen: `systf(:,1) = [ ]`
hinzufügen:	`systf = [systf,sys]` (Typ nach Rang),
	`systf(:,2) = sys` (Typ systf)
Horizontal:	`sys = [ sys1 , sys2 ]`
Vertikal:	`sys = [ sys1 ; sys2 ]`
Diagonal:	`sys = append(sys1,sys2)`
Parallel und seriell:	`sys = parallel(sys1,sys2,in1,in2,out1,out2)`
	`sys = series(sys1,sys2,outputs1,inputs2)`
Rückkopplung:	`sys = feedback(sys1,sys2)`

3. Analysieren der Systemeigenschaften

3.1. Allgemeine Eigenschaften

- Überprüfung und Abfrage von durch MATLAB bestimmte Systemeigenschaften
- Nützlich für die Programmierung komplexer Skripts und Funktionen:
  - Auswerteroutinen
  - Komplexe Plots erstellen
- Systemeigenschaften oder Boolesche Werte (ja/nein) als Rückgabewerte

• Modelltyp:	– ausgeben:	`class(sys)`
	– prüfen:	`isa(sys,'classname')`
• Zeitdaten:	– zeitkontinuierlich:	`isct(sys)`
	– zeitdiskret:	`isdt(sys)`
	– Verzögerung:	`hasdelay(sys)`
• Strukeur:	– Ein- & Ausgänge:	`isempty(sys)`
	– Ordnung:	`isproper(sys)`
	– SISO – Modell:	`issiso(sys)`
• Größe:		`size(sys)`

3.2. Modell – Dynamik

- Stationäre (Gleich –) Verstärkung: `dcgain(sys)`
  - – Frequenz ist $s = 0$ bzw. $z = 1$
  - – Reine Integratoren: Verstärkung ∞
- Natürliche Frequenzen und Dämpfungen: `damp(sys)`

Pole	Kreisfrequenz	Dämpfungsgrad
$p_i = a + j\,b$	$\omega_n = \sqrt{a^2 + b^2}$	$\xi_n = \frac{\lvert a \rvert}{\sqrt{a^2 + b^2}}$

- Nullstellen: `damp(sys)`
- Polstelen: – Eigenwerte der Matrix A: `pole(sys)`
  – Wurzeln des Polynoms c: `roots (sys)`
- Sortieren: – zeitkontinuierlich: [*s*, *ndx*] = `esort(p)`
  – zeitdiskret: [*s*, *ndx*] = `dsort(p)`
- Null – Polstellen – Verteilung: [p, *z*] = `pzmap(p)`
  Linien gleicher Dämpfung und natürlicher Frequenz: `sgrid`

3.3. Systemantwort im Zeitbereich

- Befehlsaufruf: `[y,t,x] = befehl(sys,par)`

  Zeitvektor: `t = 0: dt: Tf`
  Ausgang *y*: SIMO: `length(t)` × *Ny*; MIMO: `length(t)` × *Ny* × *Nu*
- Freie Bewegung: `[y,t,x] = inital(sys,x0,t)`
  Eingänge zu Null gesetzt, Anfangswerte des Zustandsvektors x_0
- Impulsantwort: `[y,t,x] = impulse(sys,t)`
- Sprungsantwort: `[y,t,x] = step(sys,t)`
- Systemantwort: `[y,t,x] = lsim(sys,u,t,x0)`
- Testsignal: `[u,t] = gensig(typs,tau)`

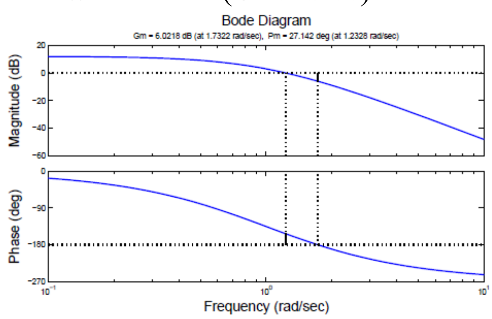
  typ: ‘`sin`‘ = Sinus — `tau` = Periodendauer
  ‘`square`‘ = Rechteck
  ‘`pulse`‘ = periodic pulse

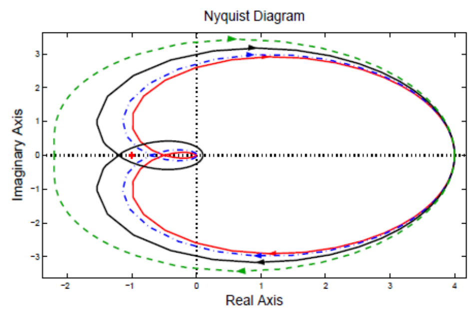
3.4. Systemantwort im Frequenzbereich

Frequenzgang: Komplexe Antwort auf Sinusanregung im eingeschwungenen Zustand
Voraussetzung: System asymptotisch stabil, d.h. Realteile aller Eigenwerte < 0

- Frequenzantwort berechnen:

  Einzelne Frequenz *f*: `frsp = evalfr(sys,f)`
  Frquenzvektor ω: `H = freqresp(sys,W)`
- Bode – Diagramm: `[mag,phase,W] = bode(sys)`
- Nyquist – Diagramm: `[re,im,W] = nyquist(sys)`
  Real- und Imaginärteil der Übertragungsfunktion des offenen Regelkreises von $\omega = 0$ bis ∞ (Ortskurve).





4. Entwurf und Optimierung von Reglern

4.1 Wurzelortskurvenverfahren

- Wurzelortskurve: Verhalten der Pole des geschlossenen Regelkreises in Abhängigkeit des Rückführverstärkungsfaktors k in der komplexen Null – Polstellen – Ebene.
- Übertragungsfunktion offener Regelkreis

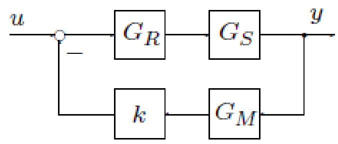
$$-G_0 = k \cdot G_R \cdot G_S \cdot G_M = \quad k \cdot \frac{n_R \cdot n_S \cdot n_M}{d_R \cdot d_S \cdot d_M}$$

- Pole von G_0 = Wurzeln des Nennerpolynoms von G

$$d_0 + k{\cdot}n_0 = d_R{\cdot}d_S{\cdot}d_M + k{\cdot}n_R{\cdot}n_S{\cdot}n_M = 0$$

- Wurzelortskurve:

```
rlocus(sys[,k])
[r,k] = rlocus(sys)
r = rlocus(sys,k)
```



- Verstärkungsfaktoren interaktiv auslesen:

```
[k,r] = rlocfind(sys)
[k,r] = rlocfind(sys,p)
```

4.2 Interaktiver Reglerentwurf mit dem SISO Design Tool

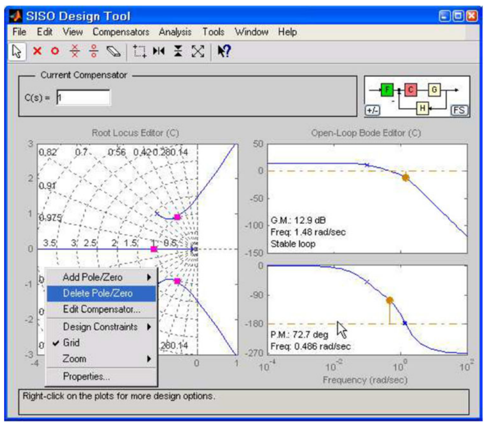
- SISO Design Tool

Reglerentwurf mit:

- Bode – Diagramm
- WOK – Verfahren

Start mit:

```
sisotool (sys)
```



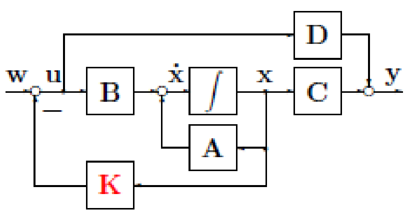
4.3 Polplazierung in Verbindung mit Zustandsrückführung

- Vollständige Zustandsrückführung mit Rückfährmatrix K

Streck:

$$\dot{x} = A\,x + B\,u$$

$$y = C\,x + D\,u$$



Regelgesetz (w = o): $\quad u = -K\,x$

Geschlossener Regelkreis: $\quad \dot{x} = (A - B\,K) \cdot x$

- Polplazierung: Rückfährmatrix K so berechnen, dass Pole des geschlossenen Regelkreises den Polen eines vorgegebenen Wunschpolynoms entsprechen.
- Zustandsregler – Rückfährvektor k/Rückfährmatrix K

```
k = acker(A,b,p)
K = place(A,B,p)
[K,prec,message] = place(A,B,p)
```

4.4 Linear – quadratisch optimale Regelung

- Minimierung eines quadratischen Gütekriteriums:
  (Q gewichtet Zustände, R gewichtet Eingänge)

$$J(\mathbf{x},\mathbf{u}) = \int_{t=0}^{\infty} \left(\mathbf{x}^T \mathbf{Q}\,\mathbf{x} + \mathbf{u}^T \mathbf{R}\,\mathbf{u} + 2\mathbf{x}^T \mathbf{N}\,\mathbf{u} \right) dt$$

- Algebraische Matrix – Riccati – Gleichung lösen

$$\mathbf{0} = \mathbf{A}^T \mathbf{S} + \mathbf{S}\mathbf{A} - (\mathbf{S}\mathbf{B} + \mathbf{N})\,\mathbf{R}^{-1}(\mathbf{B}^T \mathbf{S} + \mathbf{N}^T) + \mathbf{Q}$$

- Rückführmatrix K:

$$\mathbf{K} = \mathbf{R}^{-1}(\mathbf{B}^T \mathbf{S} + \mathbf{N}^T)$$

- LQ – optimierte Regler – Rückführmatrix K

```
[K,S,e] = lqr(A,B,Q,R,N])
[K,S,e] = lqry(sys,Q,R,N])
```

5. Bewertung der LTI – Modelle

- Zustandsdarstellung (SS)
  - Grundsätzlich am besten geeignet!
  - Algorithmen oft für SS – LTI – Modelle implementiert
- Übertragungsfunktion (TF)
  - Nur für Systeme niedriger Ordnung (< 10)
  - Oft schlecht konditioniert
- Nullstellen – Polstellen – Darstellung (ZPK)
  - Meist besser als TF – LTI – Modell
  - Probleme: mehrfache Polstellen/Polstellen bei Null
- Modelle möglichst als SS – LTI – Modell beschreiben.
  Hierbei möglichst eine normierte bzw. austarierte Beschreibung bei verwenden.
- Konvertierungen zwischen Modelltypen vermeiden.
- Ergebnisse auf ihre Verlässlichkeit und Realitätsnähe Überprüfen.

Wichtigste Ingenieuraufgabe!

Übung 2 – MATLAB Control System Toolbox (Anwendungen)

SISO Funktion `tf` bildet aus dem Zähler und Nenner die Übertragungsfunktion:

```
sys = tf (Z, N)

Z=[1 2 2];
N=[1 2 2 1];
s=tf(Z,N)

s =
      s^2 + 2 s + 2
  ---------------------
  s^3 + 2 s^2 + 2 s + 1
```

```
sys = tf (Z, N, 'variable', 'p')

Z1=[1 -1];
N1=[1 3 2 1];
s1=tf(Z1,N1,'variable','p')

s1 =
          p - 1
  ---------------------
  p^3 + 3 p^2 + 2 p + 1
```

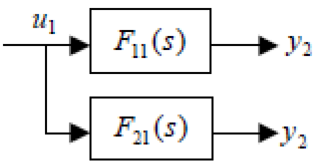
SISO Funktion `tfdada` Gibt den Zähler und Nenner der Übertragungsfunktion zurück:

```
sys = tfdata (sys, 'v')
```

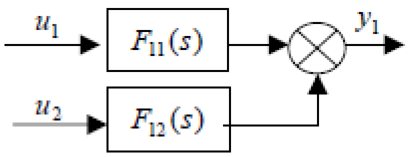
```
[Z, N] = tfdata(s, 'v')

Z =
     0     1     2     2
N =
     1     2     2     1
```

```
[Z, N] = tfdata(s1,'v')

Z =
     0     0     1    -1
N =
     1     3     2     1
```

MIMO Funktion `tf`:




```
nums={2 ;[-1 2] };
denums={[4 4 1];[1 2 1]};
G21=tf(nums, denums)

G21 =
  From input to output...
             2
   1:  ---------------
       4 s^2 + 4 s + 1

          -s + 2
   2:  -------------
       s^2 + 2 s + 1
```




```
nums1={2 [-1 2] };
denums1={[2 2 1] [1 2 1]};
G12=tf(nums1, denums1)
G12 =
  From input 1 to output:
         2
  ---------------
  2 s^2 + 2 s + 1

  From input 2 to output:
     -s + 2
  -------------
  s^2 + 2 s + 1
```

Funktion `zpk` bildet die Übertragungsfunktion in der Null-Pol-Form:

```
sys = zpk (Z, P, K)
Z=[]; P=[-1 -0.5 -2]; K=[2];
sz=zpk(Z,P,K)
```

```
Z1=[-0,5]; P1=[-1+i, -1-i, -0,5];
K1=[10];   sz1=zpk(Z1,P1,K1)
```

```
sz =
           2
  -------------------
  (s+1) (s+0.5) (s+2)
```

```
sz1 =
        10 s (s-5)
  ---------------------
  s (s-5) (s^2 + 2s + 2)
```

Funktion `zpkdata`:

```
[Z,N] = zpkdata (sys, 'v')
```

```
[Z,P,K]=zpkdata(sz,'v')

Z =
  0×1 empty double column vector
P =
   -1.0000
   -0.5000
   -2.0000
K =
     2
```

```
[Z1,P1,K1]=zpkdata(sz1,'v')

Z1 =
     0
     5
P1 =
  -1.0000 + 1.0000i
  -1.0000 - 1.0000i
   0.0000 + 0.0000i
   5.0000 + 0.0000i
K1 =
    10
```

Funktion `ss` bildet ein LTI Modell in der Zustandsform:

Aufgabe

Für eine Übertragungsfunktion

$$F(s)=\frac{2}{s^3+3s^2+s+1}$$

Bilden Sie ein LTI Objekt in der Form einer Zustandsgleichung.

DGL: $y'''=-3y''-y'-y+2u$

Zustandsgröße: $x_1=y \qquad \dot{x}_1=y'=x_2,$

$x_2=y' \rightarrow \quad \dot{x}_2=y''=x_3,$

$x_3=y'' \qquad \dot{x}_3=y'''=-3x_3-x_2-x_1+2u,$

Zustandsgleichung

$$\dot{\mathbf{x}}=\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix}=\begin{bmatrix}0&1&0\\0&0&1\\-1&-1&-3\end{bmatrix}\cdot\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}+\begin{bmatrix}0\\0\\2\end{bmatrix}\cdot u=\mathbf{A}\cdot\mathbf{x}+\mathbf{B}\cdot u$$

Ausgang:

$$y=[1\;0\;0]\cdot\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}+0\rightarrow y=\mathbf{C}\mathbf{x}.$$

```
A=[0 1 0; 0 0 1; -1 -1 -3];  B=[0;0;2]; C=[1 0 0];D=0;
```

```
ss1=ss(A,B,C,D)

ss1 =
  A =
        x1  x2  x3
    x1   0   1   0
    x2   0   0   1
    x3  -1  -1  -3

  B =
        u1
    x1   0
    x2   0
    x3   2

  C =
        x1  x2  x3
    y1   1   0   0

  D =
        u1
    y1   0
Continuous-time state-
space model.
```

Funktion `ssdata` gibt die Zustandsform eines LTI Modells zurück

```
[A,B,C,D] = ssdata(sys, 'v')
```

Modelltransformationen:

Alle Modellformen können gegenseitig überführt werden. Dazu dienen Funktionen für die Modell- Transformation.

- In die Übertragungsfunktion:

`sys = tf(sysf)` `sys` = Übertragungsfunktion
`sysf` = Pol- Nullstellen-Modell oder Zustandsmodell

```
szpk=tf(sz)

szpk =
              2
  -------------------------
  s^3 + 3.5 s^2 + 3.5 s + 1
```

```
sss1=tf(ss1)

sss1 =
          2
  -------------------
  s^3 + 3 s^2 + s + 1
```

- In die Pol- Nullstelle- Form:

`sys = zpk(sysf)` `sys` = in der Pol- Nullstelle- Form
`sysf` = Übertragungsfunktion oder Zustandsmodell

```
szpk1=zpk(s)

szpk1 =
    (s^2 + 2s + 2)
  -------------------
  (s+1) (s^2 + s + 1)
```

```
szpk2=zpk(ss1)

szpk2 =
                  2
  ----------------------------------
  (s+2.769) (s^2 + 0.2307s + 0.3611)
```

- In das Zustandsmodell:

`sys = ss(sysf)` `sys` = Zustandsmodell
`sysf` = Übertragungsfunktion oder in der Pol- Nullstelle- Form

```
ss1s=ss(s)

ss1s =

  A =
       x1  x2  x3
   x1  -2  -2  -1
   x2   1   0   0
   x3   0   1   0

  B =
       u1
   x1   2
   x2   0
   x3   0

  C =
        x1   x2   x3
   y1  0.5    1    1

  D =
       u1
   y1   0
Continuous-time state-space model.
```

```
ss2s=ss(sz)

ss2s =

  A =
         x1    x2    x3
   x1  -0.5     1     0
   x2     0    -1     1
   x3     0     0    -2

  B =
       u1
   x1   0
   x2   0
   x3   2

  C =
       x1  x2  x3
   y1   1   0   0

  D =
       u1
   y1   0
Continuous-time state-space model.
```

Zeitverhalten der LTI – Systeme:

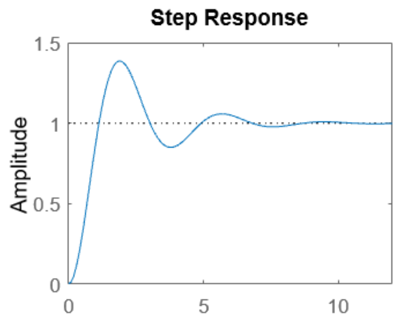
Der Zeitverlauf der Übergangsfunktion wird im MATLAB mit Hilfe der Funktion `step` berechnet und graphisch dargestellt.

Funktion `step` berechnet und stellt den Zeitverlauf der Übergangsfunktion dar:

```
step(sys),
step(sys,t),
step(sys1,sys2,…,sysN),
[y,t]=step(sys)
```

Vektor y = Ordinatenwerte, Vektor t = Zeit.

```
step(tf([3],[1 1 3]))
```

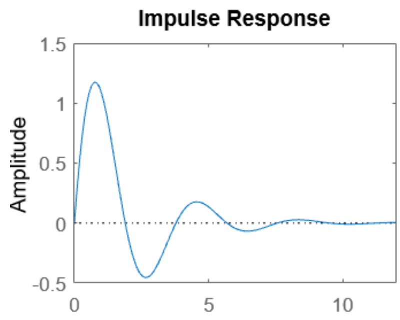


Funktion `impulse` berechnet und stellt den Zeitverlauf der Impulsfunktion dar:

```
impulse(sys),
impulse(sys,t),
impulse(sys1,sys2,…,sysN),
[y,t]= impulse (sys)
```

Vektor y = Ordinatenwerte, Vektor t = Zeit.

```
impulse (tf([3],[1 1 3]))
```



FREQUENZVERHALTEN DER LTI – SYSTEME:

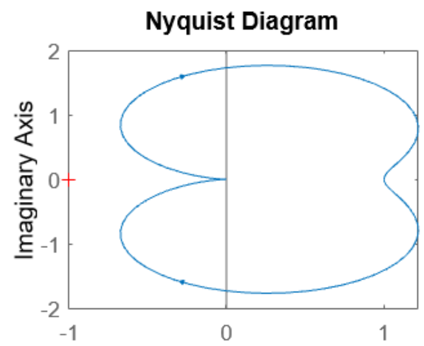
Für Analyse und Synthese in den Frequenzbereich kann man in MATLAB folgende Anweisungen benutzen: `nyquist, bode, evalfr, freqresp, margin.`

Funktion `nyquist` Frequenzgangdarstellung und -berechnung:

```
nyquist(sys)
nyquist(sys1,…,sysN)
nyquist(sys,w)
[re,im,w]=nyquist(sys)
```

`re,im,w` = Vektore

```
nyquist(tf([3],[1 1 3]))
```



Funktion `freqresp` Berechnung des Frequenzganges eines LTI Systems:

```
w=[0.1 0.5 1];
H=freqresp(tf([3],[1 1 3]),w)
  H(:,:,1) =
       1.0022 - 0.0335i
H(:,:,2) =
     1.0560 - 0.1920i
H(:,:,3) =
     1.2000 - 0.6000i
```

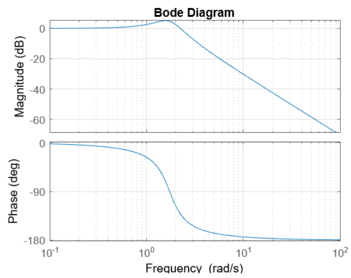
Funktion `evalfr` Berechnung des Frequenzganges für a komplex Zahl:

```
f=2+i;
evalfr(tf([3],[1 1 3]),f)
```

```
ans =

   0.4800 - 0.3600i
```

Funktion `bode` Berechnung der logarithmischen Amplituden- und Phasengänge:

```
bode (sys)
bode (sys1,…,sysN)
bode (sys,w)
[mag,phase,w] =bode (sys)
bode(tf([3],[1 1 3]));grid
```



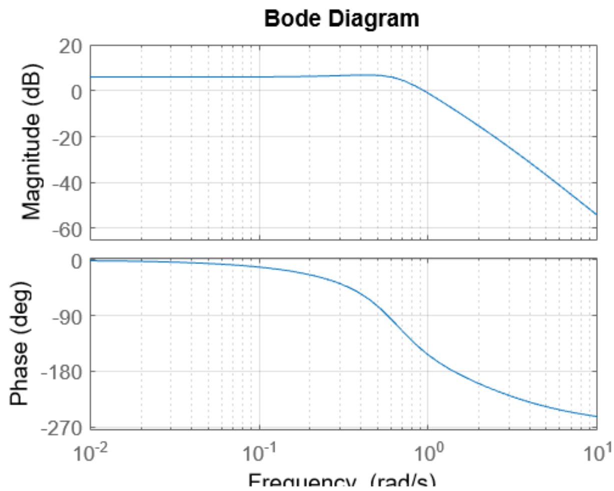
Funktion `margin` Berechnung der Amplituden- und Phasenrand:

```
[Gm,Pm,Wcg,Wcp] = margin (sys)
    sys… ein LTI system ist,
    Wcg… Durchtrittfrequenz des Amplitudenrandes
    Wcp… Durchtrittfrequenz des Phasenrandes
    Pm…  Phasenrand
    Gm…  Amplitudenrand
```

Berechnen Sie den Amplituden und – Phasenrand für $F(s) = \dfrac{2}{s^3 + 3s^2 + 2s + 1}$

```
N1=[1 3 2 1];
Z1=[2];
figure(1)
F1=tf(N1,Z1)
[Gm1,Pm1,Wcg1,Wcp1]=margin(F1)
bode(F1)
```

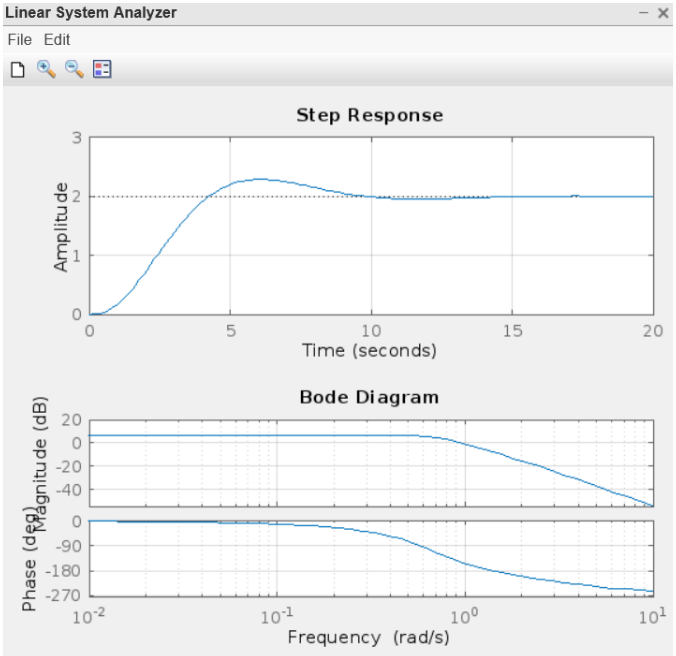
```
Gm1 =     2.5015
Pm1 =    31.4246
Wcg1 =     1.4146
Wcp1 =     0.9498
```



Funktion `ltiview` Berechnung und stellt den Zeit- oder Frequenzverlauf eines LTI Systems dar:

```
ltiview
ltiview (plottype,sys)
ltiview (plottype,sys1,sys2, … ,sysN)

    plottype = 'step' / 'impulse' / 'bode' / 'nyquist'
```

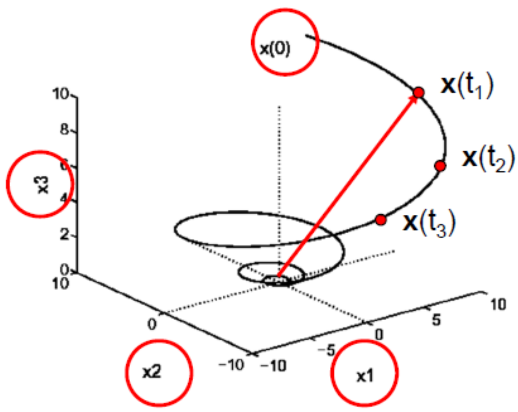
```
ltiview({'step';'bode'},F1);
```

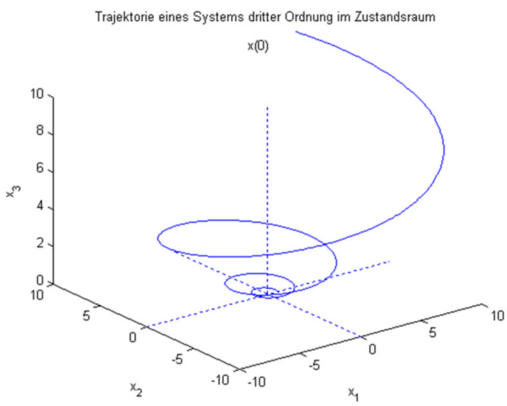


Zustandstrajektorie

```
% Script Trajektorie.m
echo off
clear
close all;
A = [-0.5 0 0; 0 -0.3 2; 0 -2 -0.3];
B = [1; 1; 1]; C = [1 1 1]; D = 0;
System=ss(A,B,C,D);
x0 = [10, 10, 10];
echo on
%
% Eigenbewegung eines Systems dritter Ordnung
% dreidimensionale Darstellung im Zustandsraum
%
echo off
figure(1);
T=0:0.01:10;
[Y, T, X]=initial(System, x0, T);
plot3(X(:, 2), X(:, 3), X(:, 1));
hold on
plot3([-10 10], [0 0], [0 0], ':');
plot3([0 0], [-10 10], [0 0], ':');
plot3([0 0], [0 0], [10 0], ':');
xlabel('x_1'); ylabel('x_2');
zlabel('x_3'); grid('off');
title('Trajektorie eines Systems dritter
Ordnung im Zustandsraum');
text(6, 10, 10, 'x(0)')
axis([-10 10 -10 10 0 10]);
```

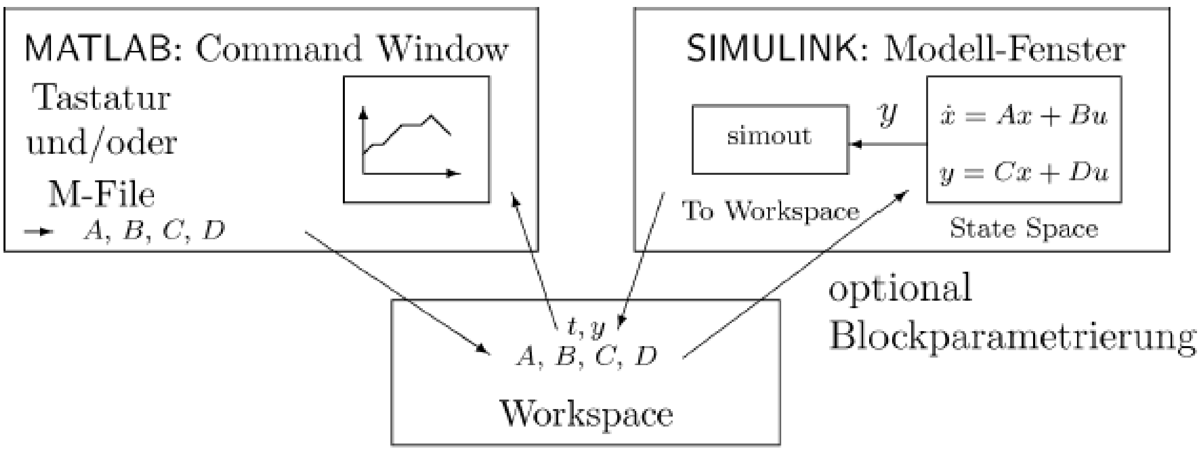
Der durch die Koordinaten von $x(t)$ beschriebene Punkt verändert sich mit der Zeit und beschreibt eine Kurve im Zustandsraum, die als Zustandskurve oder Trajektorie des Systems bezeichnet wird.





Modelltransformationen zwischen MATLAT und Simulink:

Der Datentransfer aus dem Simulink-Modell in den Workspace erfolgt u. a. über den `TO Workspace Block` in `Array-` oder `Structure`-Format, z. B. wie in Bild für die Matrix `simout` oder den `Outport Block` wie in Bild. Damit können die Daten in MATLAB u. a. zur grafischen Darstellung weiterverarbeitet werden. Eine weitere Möglichkeit zur Übergabe der Simulationsdaten in den Workspace kann im Scope-Fenster durch aktivieren der `Array-` oder `Structure` - Übergabe erreicht werden.




```
>> [A,B,C,D]=linmod('modell')  % erzeugt die Matrizen für die lineare
                               % Zustands raum-Darstellung des kompletten Modells
>> [zaehler,nenner]=ss2tf(A,B,C,D) % berechnet aus den Matrizen A,..D den
                                   % Zähler und Nenner für die entsprechende Tf-Funktion
>> [A,B,C,D] = tf2ss(zaehler,nenner) % berechnet die Matrizen des linearen
                                     % Zustandsraums aus den Zähler- und Nenner-Polynomen
```

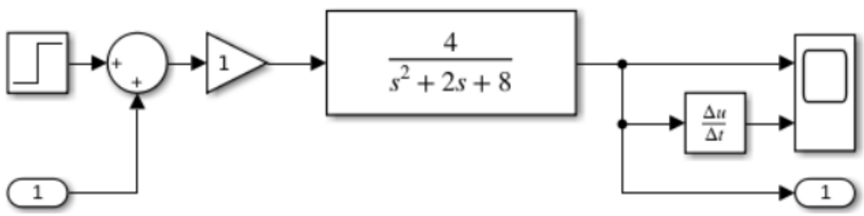
Beispiel:

Für ein simulink-Modell einer Übertragungsfunktion

$$G(s) = \frac{4}{s^2 + 2s + 8}$$

Dateiname:

`Beispiel_1a.xls`



Bilden Sie ein LTI System in der Form einer Zustandsgleichung.

Musterlösung:

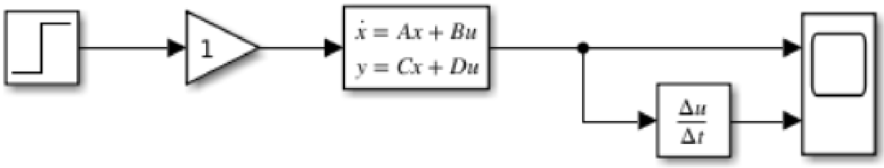
```
[A1,B1,C1,D1]=linmod('Beispiel_1a')
  A1 = -2    -8
        1     0
  B1 =  1
        0
  C1 =  0     4
  D1 =  0
  [A1,B1,C1,D1] = tf2ss(zaehler,nenner)
```

```
[zaehler,nenner]=ss2tf(A1,B1,C1,D1)
 zaehler =

     0     0     4

 nenner =

    1.0000    2.0000    8.0000
```

A1, B1, C1, D1

Dateiname:

Beispiel_1b.xls




```
plot(out.ScopeData.time,out.ScopeData.signals(1,1).values,...
     out.ScopeData.time,out.ScopeData.signals(1,2).values)
```