Introduction to the Control System Toolbox (CST)

Riccardo Antonello (riccardo.antonello@unipd.it)

Giulia Michieletto (giulia.michieletto@unipd.it)

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Università degli Studi di Padova

4 Marzo 2024

Outline

- Time domain analysis of LTI models
  - Impulse, step, natural responses
  - Forced response to generic input
- Frequency domain analysis of LTI models
  - Bode & Nyquist plots
- Stability analysis of LTI models
  - Poles & zeros
  - Root locus
  - Stability margins
- Model discretization
- Control design by state-space methods

Time domain analysis of LTI models

CST routines for time response analysis:

↳ **`impulse`** : impulse response.

↳ **`step`** : unit step response.

↳ **`initial`** : natural response to specified ICs.

↳ **`lsim`** : response to generic input and ICs.

Note: when invoked without specifying any return parameter, they show the results on an *interactive plot*.

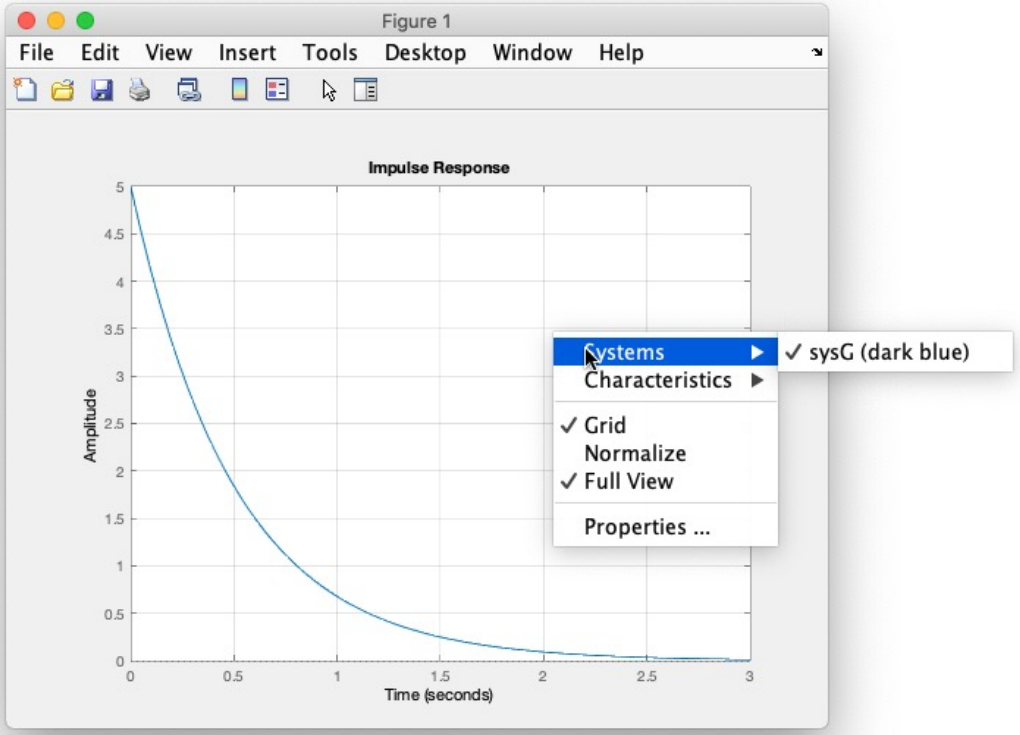
Time domain analysis of LTI models

$$G(s) = \frac{Y(s)}{U(s)} = \frac{5}{s+2}$$

Impulse response

$$g(t) = 5e^{-2t}$$

```
% display the impulse response
% on an interactive plot
figure;
impulse(sysG);
grid on;
```



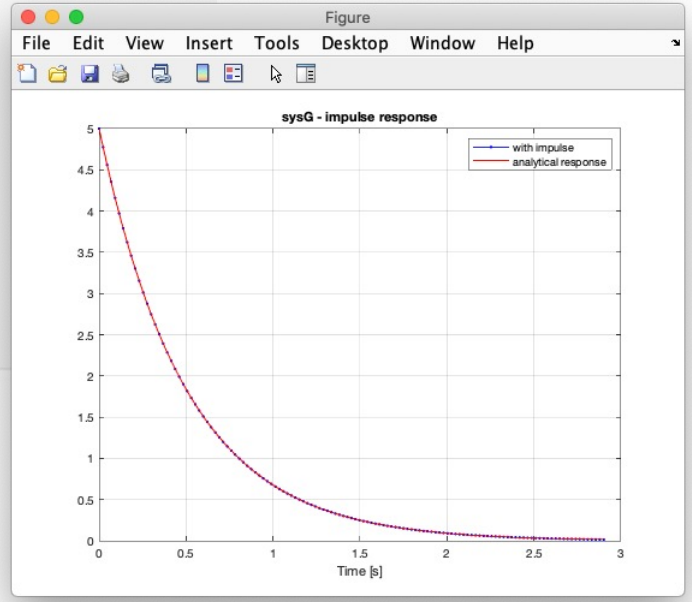
The plot generated by **`impulse`** is an interactive plot (*right-click* on it to access useful information related to the response).

Time domain analysis of LTI models

Impulse response

$$G(s) = \frac{Y(s)}{U(s)} = \frac{5}{s+2} \quad \Rightarrow \quad g(t) = 5e^{-2t}$$

```
% returns the impulse response g evaluated at the time instants t
% (automatically determined by impulse; can be specified as 2nd argument)
[g, t] = impulse(sysG);

% compare result with expected analytical response
ga =  5 * exp(-2*t);

figure;
plot(t, g, 'b.-');
hold on;
plot(t, ga, 'r');
grid on;
xlabel('Time [s]');
title('sysG - impulse response');
legend('with impulse', 'analytical response');
```

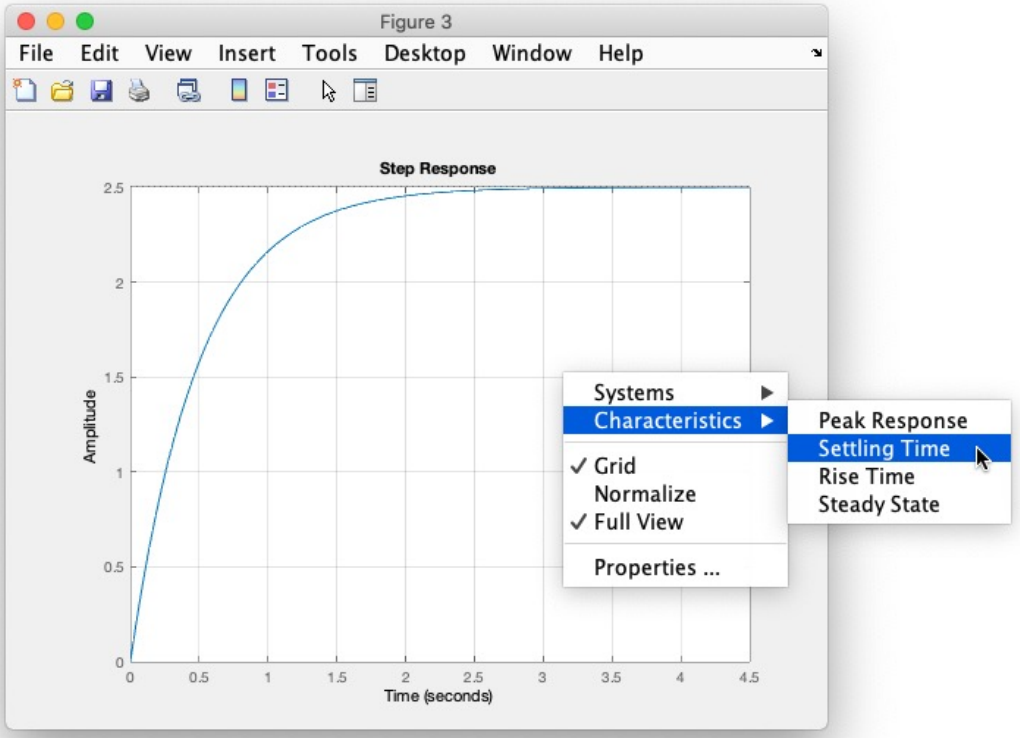


Time domain analysis of LTI models

Step response

$$G(s) = \frac{Y(s)}{U(s)} = \frac{5}{s+2} \quad \Rightarrow \quad y(t) = \frac{5}{2}\left(1 - e^{-2t}\right)$$

```
% display the step response
% on an interactive plot
figure;
step(sysG);
grid on;
```



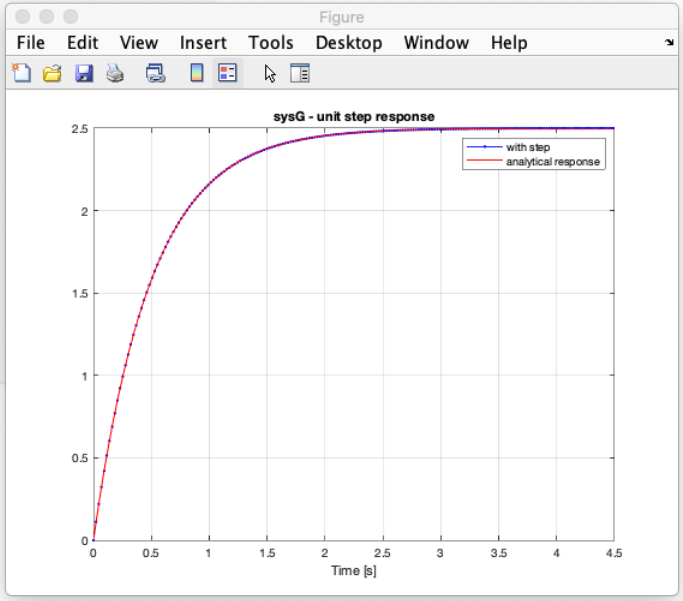
The plot generated by **`step`** is an interactive plot (*right-click* on it to access useful information related to the response).

Time domain analysis of LTI models

Step response

$$G(s) = \frac{Y(s)}{U(s)} = \frac{5}{s+2} \quad \Rightarrow \quad y(t) = \frac{5}{2}\left(1 - e^{-2t}\right)$$

```
% returns the impulse response y evaluated at the time instants t
% (automatically determined by step; can be specified as 2nd argument)
[y, t] = step(sysG);

% compare result with expected analytical response
ya = (5/2)*(1-exp(-2*t));

figure;
plot(t, y, 'b.-');
hold on;
plot(t, ya, 'r');
xlabel('Time [s]');
title('sysG - unit step response');
legend('with step', 'analytical response');
grid on;
```



Time domain analysis of LTI models

$$G(s) = \frac{Y(s)}{U(s)} = \frac{5}{s+2}$$

Step response

$$y(t) = \frac{5}{2}\left(1 - e^{-2t}\right)$$

Use **dcgain** to get the DC gain of an LTI model:

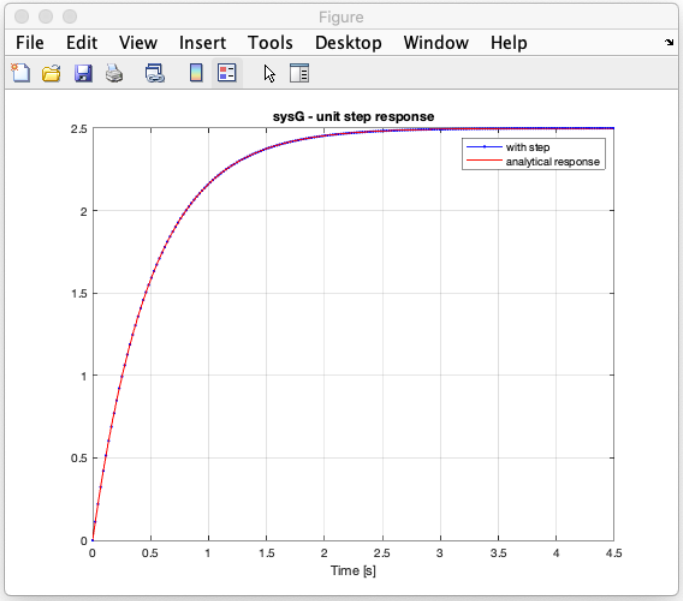
```
dcgain(sysG)
```

```
ans = 2.5000
```

Use **stepinfo** to get other relevant information regarding the step response :

```
stepinfo(sysG)
```

```
ans = struct with fields:
        RiseTime: 1.0985
    SettlingTime: 1.9560
     SettlingMin: 2.2613
     SettlingMax: 2.4999
       Overshoot: 0
      Undershoot: 0
            Peak: 2.4999
        PeakTime: 5.2729
```

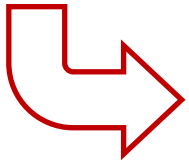


Use **dcgain** and **stepinfo** to obtain step response information in a programmatic way.

Time domain analysis of LTI models

For the evaluation of the *natural response*, the model must be converted into state-space form.

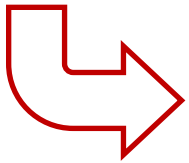
$$G(s) = \frac{Y(s)}{U(s)} = \frac{5}{s+2}$$



Ordinary differential equation (ODE)

$$\frac{dy(t)}{dt} + 2y(t) = 5u(t)$$

State-space model



$$\begin{cases} \dfrac{dx(t)}{dt} = -2x(t) + 5u(t) \\ y(t) = x(t) \end{cases}$$

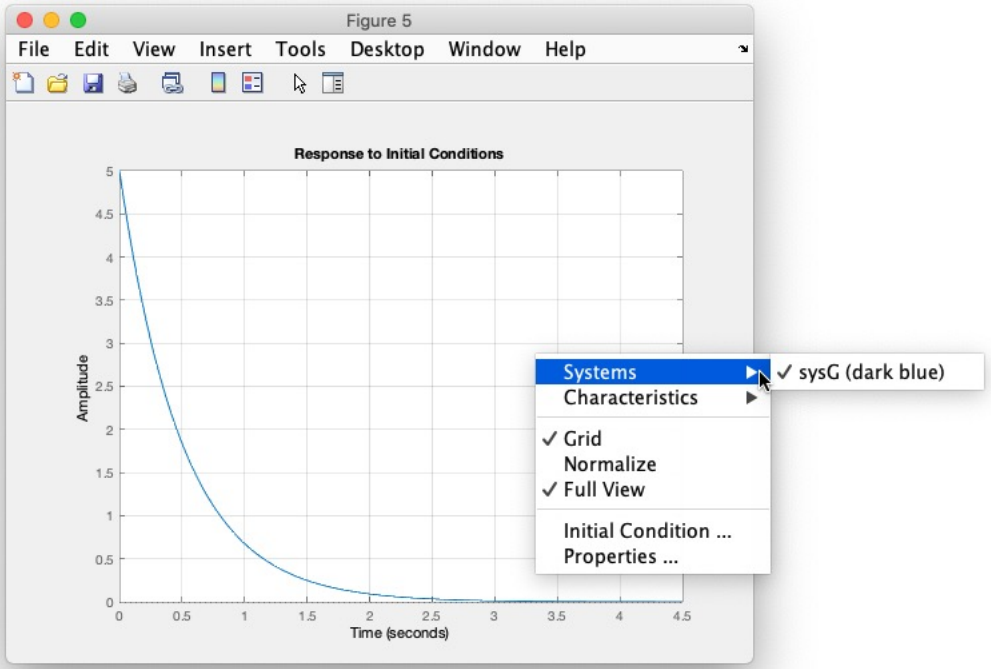
Time domain analysis of LTI models

$$\begin{cases} \dfrac{dx(t)}{dt} = -2x(t) + 5u(t) \\ y(t) = x(t) \end{cases}$$

Natural response

$$y(t) = e^{-2t}x(0)$$

```
% ss realization
A = -2; B = 5; C = 1; D = 0;
sysG = ss(A,B,C,D);

% display the natural ouput response
% on an interative plot
x0 = 5;  % initial state
figure;
initial(sysG, x0);
grid on;
```



The plot generated by **`initial`** is an interactive plot (*right-click* on it to access useful information related to the response).

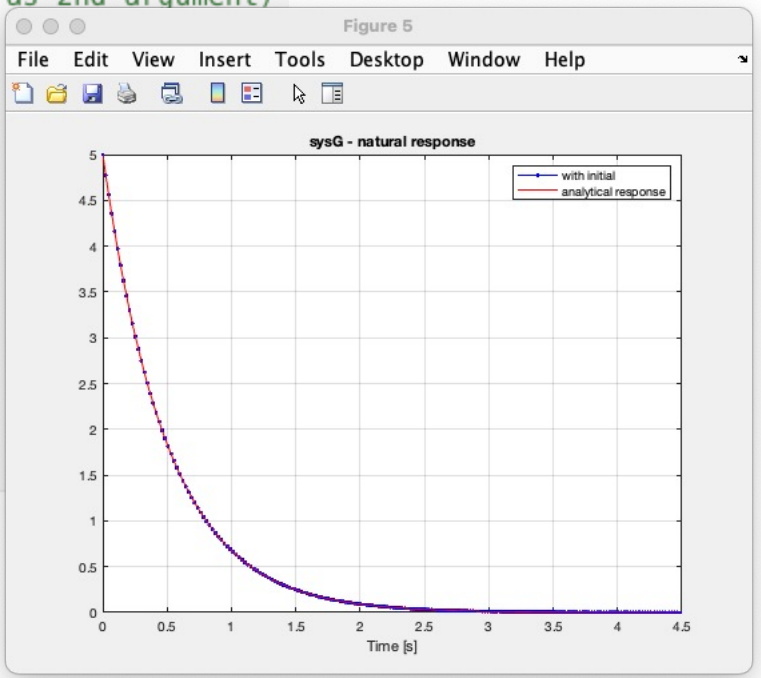
Time domain analysis of LTI models

$$\begin{cases} \dfrac{dx(t)}{dt} = -2x(t) + 5u(t) \\ y(t) = x(t) \end{cases}$$

Natural response

$$y(t) = e^{-2t}x(0)$$

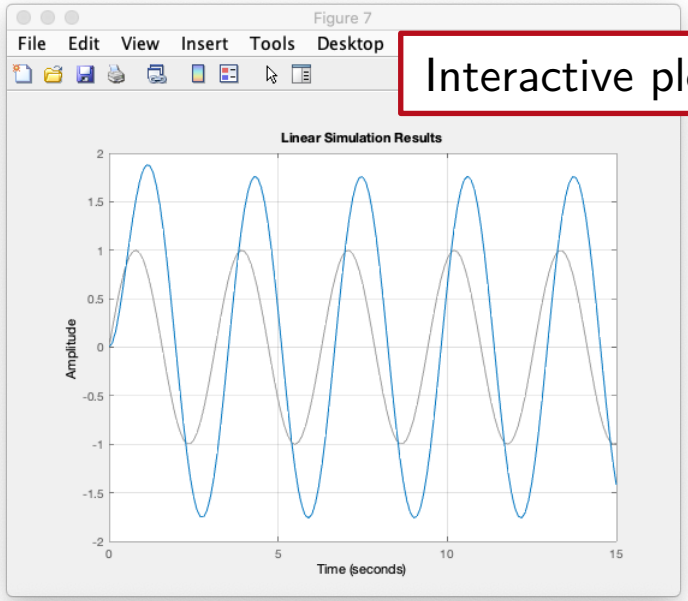
```
% returns the natural response yn evaluated at the time instants t
% (automatically determined by initial; can be specified as 2nd argument)
[yn, t] = initial(sysG, x0);

% compare result with expected analytical response
yna = C*exp(-2*t)*x0;

figure;
plot(t, yn, 'b.-');
hold on;
plot(t, yna, 'r');
xlabel('Time [s]');
title('sysG - natural response');
legend('with initial', 'analytical response');
grid on;
```



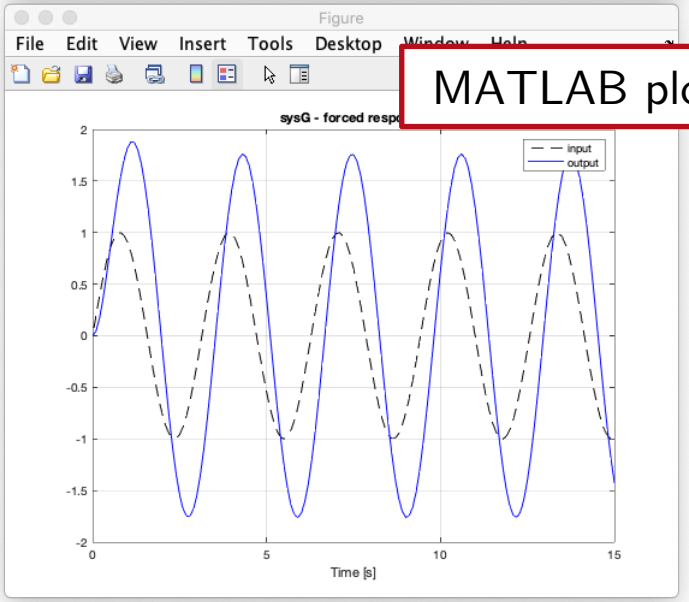
Time domain analysis of LTI models

Use **`lsim`** to obtain the response to a generic input $u(t)$ and IC $x(0)$.

```
t = 0:0.1:15;    % time instants
u = sin(2*t);    % input signal (alt.: use "gensig")
x0 = 0;          % init state

% display the forced response
% on an interactive plot
figure;
lsim(sysG, u, t, x0);
grid on;

% returns the forced response y
% evaluated at the time instants t
y = lsim(sysG, u, t, x0);

figure;
plot(t, u, 'k--');
hold on;
plot(t, y, 'b');
xlabel('Time [s]');
title('sysG - forced response');
legend('input', 'output');
grid on;
```



Interactive plot



MATLAB plot

Frequency domain analysis of LTI models

CST routines for frequency response analysis:

- **`freqresp`** : frequency response value at specified frequency points.
- **`bode`** : Bode plot.
- **`nyquist`** : Nyquist plot.

Note: when invoked without specifying any return parameter, the `bode` and `nyquist` routines show the results on an *interactive plot*.

Frequency domain analysis of LTI models

$$G(s) = \frac{Y(s)}{U(s)} = \frac{5}{s+2} \quad \Rightarrow \quad G(j\omega) = \frac{5}{j\omega+2}$$

Frequency response

```
% freq points (uniformly spaced in log scale)
w = logspace(-1, 1, 100);

%  evaluate the frequency response (alternative: use evalfr)
frG = freqresp(sysG, w);
```

```
whos frG
```

Name	Size	Bytes	Class	Attribute
frG	1x1x100	1600	double	complex

```
frG = squeeze(frG);
whos frG
```

Name	Size	Bytes	Class	Attributes
frG	100x1	1600	double	complex

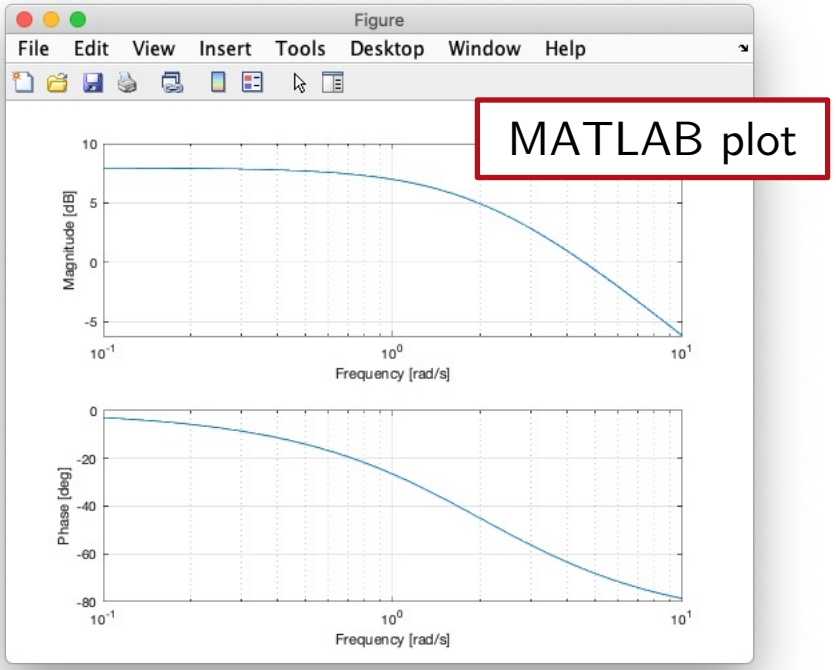
`frG` is a *multidimensional array* (*num of inputs* $\times$ *outputs* $\times$ *length of* `w`) containing the (complex) values of the frequency response, at the frequency points specified in `w` (in [rad/s]).

For SISO models, use **`squeeze`** to get the single non-empty dimension of `frG`.

Frequency domain analysis of LTI models

The *Bode plot* can be obtained by plotting the frequency response `frG` vs frequency `w` in *semi-logarithmic scale*:

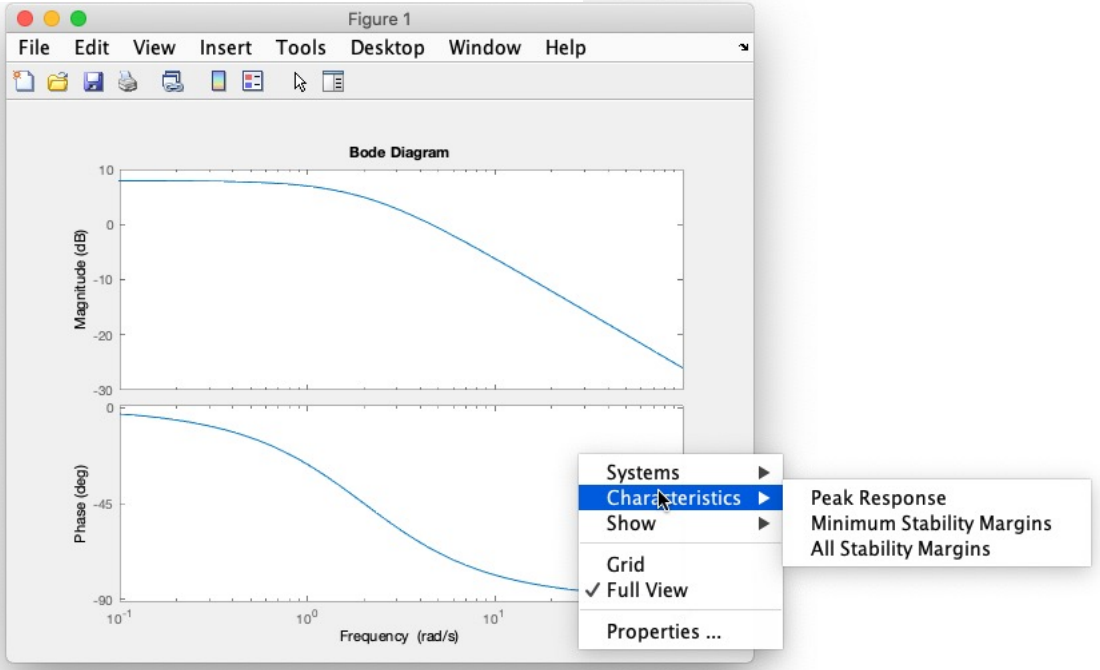
```
%  get magnitude and phase
magG = abs(frG);
phaG = angle(frG);

%  convert to dB and deg units
magG_dB = 20*log10(magG);
phaG_deg = phaG * 180/pi;

%  Bode plot
figure;
subplot(2,1,1);
semilogx(w, magG_dB);
ylabel('Magnitude [dB]');
xlabel('Frequency [rad/s]');
grid on;

subplot(2,1,2);
semilogx(w, phaG_deg);
ylabel('Phase [deg]');
xlabel('Frequency [rad/s]');
grid on;
```



Frequency domain analysis of LTI models

More immediate method: use **bode** routine.

```
% display the Bode diagram on an interactive plot
figure;
bode(sysG);

% get the frequency response (mag [absolute value] + phase [deg])
% evaluated at the freq points w (in [rad/s])
[magG, phaG_deg] = bode(sysG, w);
```



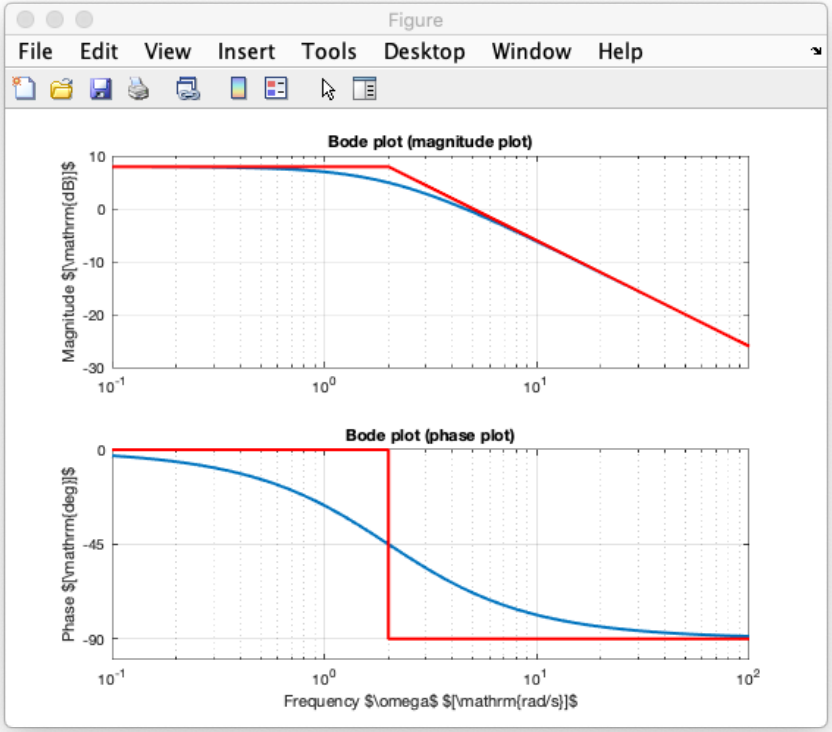
The plot generated by **bode** is an interactive plot (*right-click* on it to access useful information related to the frequency response).

Frequency domain analysis of LTI models

No routines are provided for sketching the *asymptotic* Bode plot – it is necessary to resort to 3rd party contributions (1).

`bodeasymp` routine

```
%   asymptotic Bode plot with "bodeasymp" routine
figure;
bodeasymp(sysG);
```



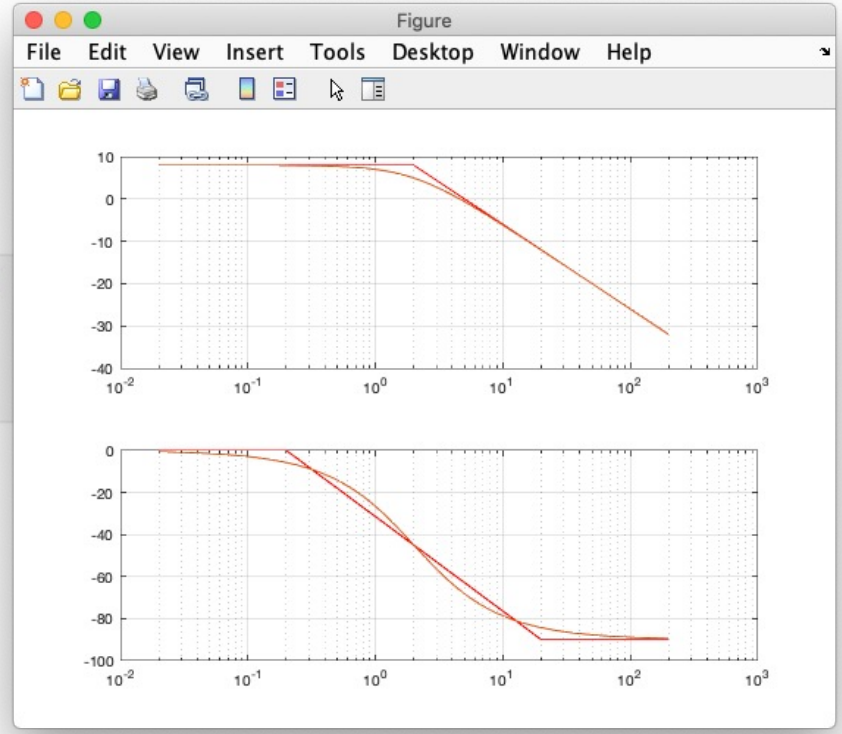
(1) Consult www.mathworks.com/matlabcentral/

Frequency domain analysis of LTI models

No routines are provided for sketching the *asymptotic* Bode plot – it is necessary to resort to 3rd party contributions (1).

`bode_asymptotic` routine

```
%   asymptotic Bode plot with "bode_asymptotic" routine
[numG, denG] = tfdata(sysG, 'v');
figure;
bode_asymptotic(numG, denG);
```

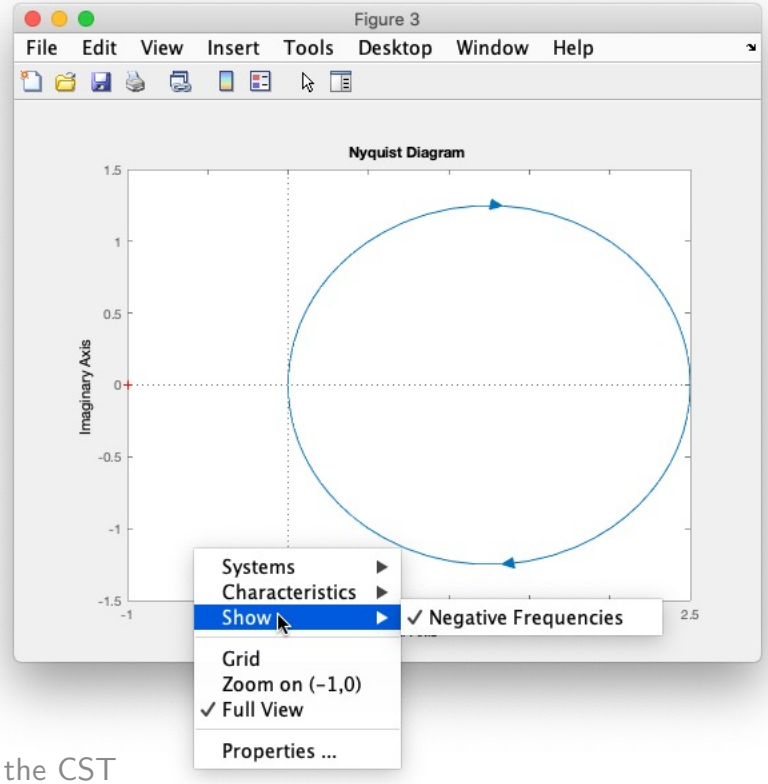


(1) Consult www.mathworks.com/matlabcentral/

Frequency domain analysis of LTI models

For Nyquist plot, use **`nyquist`** routine:

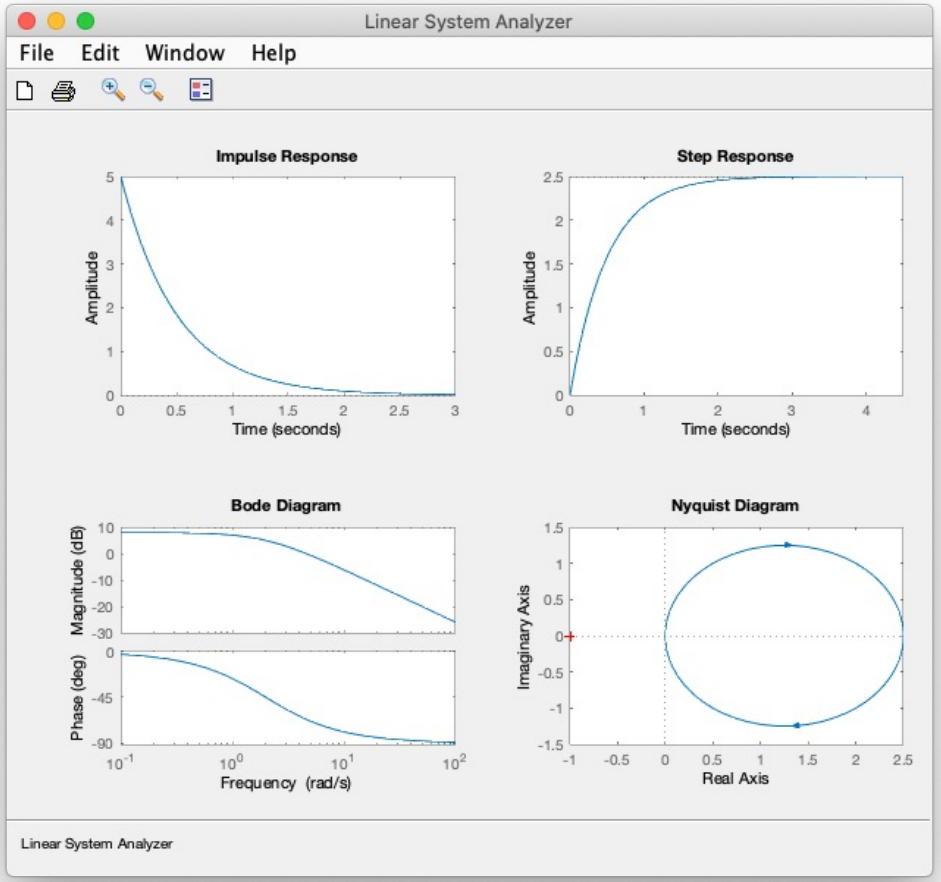
```
% display the Nyquist diagram on an interactive plot
figure;
nyquist(sysG);

% get the the real/imag part of the frequency response
% evaluated at the freq points w (in [rad/s])
[ReG, ImG] = nyquist(sysG, w);
```



The plot generated by **`nyquist`** is an interactive plot (*right-click* on it to access useful information related to the frequency response).

LTI viewer

`ltiview` provides an interactive tool to sketch and analyze time and frequency responses of multiple LTI models.



The plot generated by **`ltiview`** is an interactive plot (*right-click* on it to configure the `ltiview` outputs).

Poles & zeros

`pole` and **`zero`** return poles and zeros of an LTI model in *encapsulated form* (LTI object).

$$G(s) = \frac{s^2 + 15s + 50}{s^6 + 10s^5 + 47s^4 + 142s^3 + 250s^2 + 200s}$$

```
numG = [1, 15, 50];
denG = [1, 10, 47, 142, 250, 200, 0];
sysG = tf(numG, denG);

p1 = pole(sysG)
```

```
p1 = 6×1 complex
     0.0000 + 0.0000i
    -1.0000 + 3.0000i
    -1.0000 - 3.0000i
    -4.0000 + 0.0000i
    -2.0000 + 1.0000i
    -2.0000 - 1.0000i
```

```
z1 = zero(sysG)  % also available: [z,k] = zero(sysP)
```

```
z1 = 2×1
    -10
     -5
```

Poles & zeros

For *non-encapsulated forms*, use **`roots`** and **`eig`** to get the zeros, poles and eigenvalues.

$$G(s) = \frac{s^2 + 15s + 50}{s^6 + 10s^5 + 47s^4 + 142s^3 + 250s^2 + 200s}$$

```
p2 = roots(denG)
```

```
p2 = 6×1 complex
     0.0000 + 0.0000i
    -1.0000 + 3.0000i
    -1.0000 - 3.0000i
    -4.0000 + 0.0000i
    -2.0000 + 1.0000i
    -2.0000 - 1.0000i
```

```
z2 = roots(numG)
```

```
z2 = 2×1
    -10
     -5
```

```
A = ssdata(sysG);
eig(A)
```

```
ans = 6×1 complex
     0.0000 + 0.0000i
    -1.0000 + 3.0000i
    -1.0000 - 3.0000i
    -4.0000 + 0.0000i
    -2.0000 + 1.0000i
    -2.0000 - 1.0000i
```

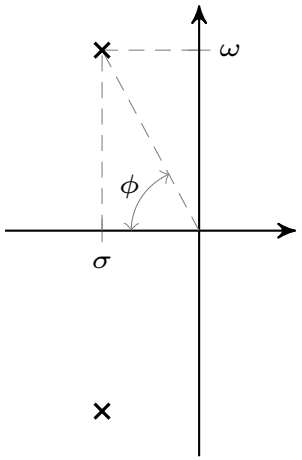
Poles & zeros

Use **`damp`** to get the natural frequency ω_n and the damping factor δ of the system poles or eigenvalues.

```
% natural freq and damping factor of
% system poles using damp
% note: poles are sorted by increasing freq
[wn1, d1] = damp(sysG)
```

```
wn1 = 6×1
         0
    2.2361
    2.2361
    3.1623
    3.1623
    4.0000

d1 = 6×1
   -1.0000
    0.8944
    0.8944
    0.3162
    0.3162
    1.0000
```

Complex conjugate pole pair

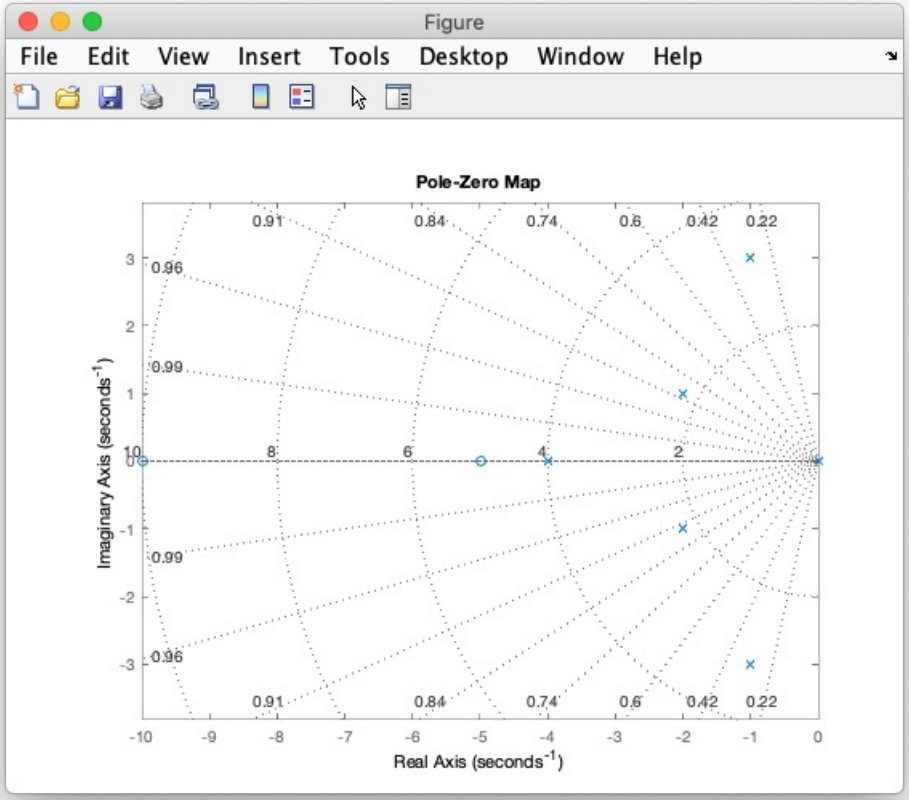


$$p_{1,2} = \sigma \pm j\omega$$
$$= -\delta\omega_n \pm j\,\omega_n\sqrt{1-\delta^2}$$

Poles & zeros

Use **`pzmap`** to plot poles ($\times$) and zeros ($\circ$) on the complex plane.

```
pzmap(sysG);
axis equal;
sgrid;
```

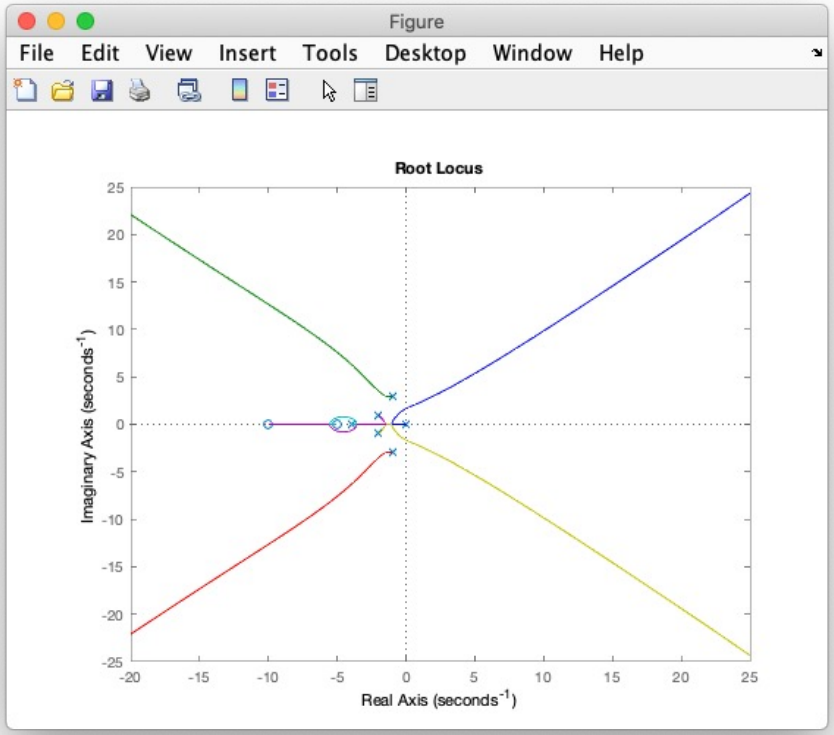


Use **`sgrid`** (**`zgrid`** for discrete-time case) to superimpose a grid showing the constant damping ratio and natural frequency loci.

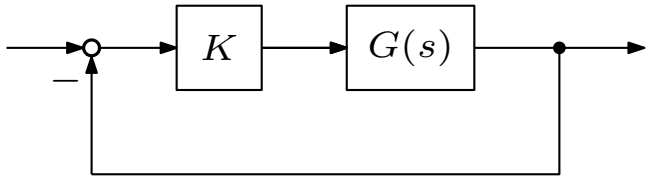
Root locus

Use **`rlocus`** to plot the (positive) root locus.

```
% show the root locus
% on the complex plane
figure;
rlocus(sysG);
```



Note:



If $G(s) = B(s)/A(s)$, the root locus shows how the roots of the *characteristic polynomial* (i.e. poles of the closed-loop system):

$$p(s) = A(s) + K\,B(s)$$

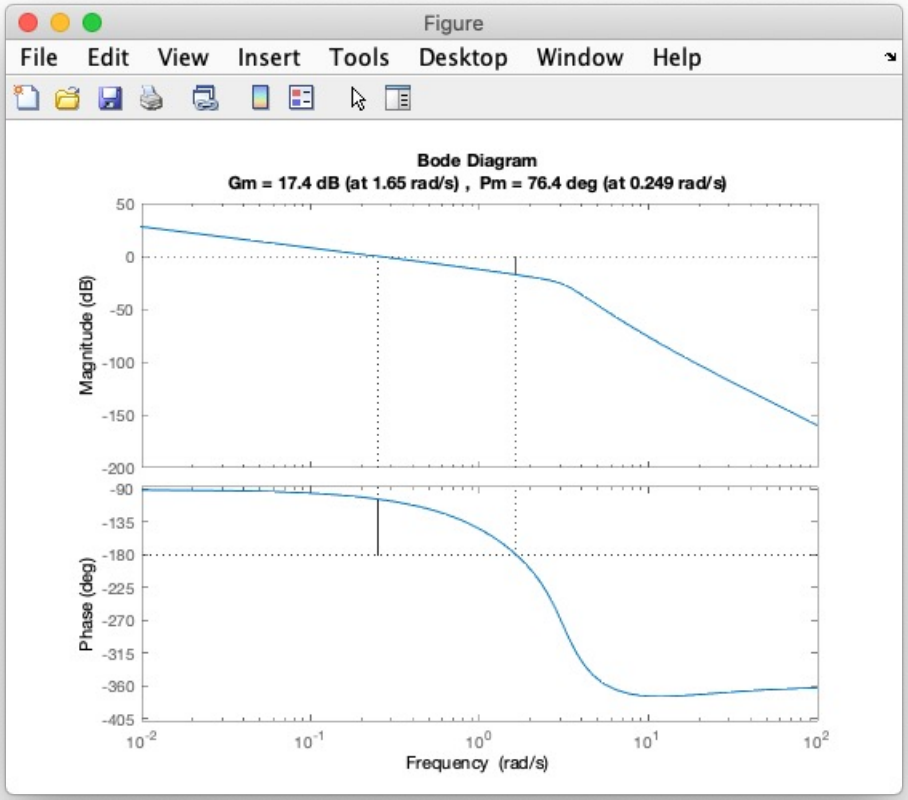
move on the complex plane as K varies from 0 to $+\infty$.

Stability margins

Use **`margin`** to evaluate the stability margins (*gain* and *phase*) of a specified system.

```
% show the stability margins on the Bode plot
figure;
margin(sysG);

% return the stability margins
[gm, phim, wgc, wcp] = margin(sysG)
```

```
gm = 7.4492
phim = 76.4281
wgc = 1.6473
wcp = 0.2493
```



Model discretization

Use **`c2d`** to discretize a given continuous-time model with one of the following methods:

↳ *zero-order hold* (**`zoh`**): exact for *staircase* inputs.

↳ *first-order hold* (**`foh`**): exact for *piecewise linear* inputs.

↳ *impulse-invariant* (**`impulse`**): exact for *impulse train* inputs.

↳ *Tustin* (**`tustin`**): yields the best frequency-domain match between the continuous-time and discretized systems.

↳ *zero-pole matching* (**`matched`**): discrete-time equivalent with same DC gain, and poles/zeros obtained with the transformation $z = e^{s\,T_s}$.

Model discretization

$$G(s) = \frac{-s+1}{s^2+0.6s+1}$$

```
% continuous-time model
sysG = tf([-1 1], [1, 2*0.3, 1]);

% discretizations
Ts = 0.4;
sysGd1 = c2d(sysG, Ts, 'zoh')
```

```
sysGd1 =

    -0.2732 z + 0.4136
  ----------------------
  z^2 - 1.646 z + 0.7866

Sample time: 0.4 seconds
Discrete-time transfer function.
```

```
sysGd2 = c2d(sysG, Ts, 'foh')
```

```
sysGd2 =

  -0.1576 z^2 + 0.1074 z + 0.1905
  -------------------------------
      z^2 - 1.646 z + 0.7866

Sample time: 0.4 seconds
Discrete-time transfer function.
```

```
sysGd3 = c2d(sysG, Ts, 'impulse')
```

```
sysGd3 =

   -0.4 z^2 + 0.5093 z
  ----------------------
  z^2 - 1.646 z + 0.7866

Sample time: 0.4 seconds
Discrete-time transfer function.
```

```
sysGd4 = c2d(sysG, Ts, 'tustin')
```

```
sysGd4 =

  -0.1379 z^2 + 0.06897 z + 0.2069
  --------------------------------
       z^2 - 1.655 z + 0.7931

Sample time: 0.4 seconds
Discrete-time transfer function.
```

```
sysGd5 = c2d(sysG, Ts, 'matched')
```

```
sysGd5 =

    -0.2854 z + 0.4258
  ----------------------
  z^2 - 1.646 z + 0.7866

Sample time: 0.4 seconds
Discrete-time transfer function.
```

Model discretization

Note: **`c2d`** does not implement the *forward* and *backward Euler* discretization methods.

Convenient approach for Forward (FE) and Backward Euler (BE) discretization

FE transformation

$$s = \frac{z-1}{T_s}$$

```
% define z-domain variable
z = tf('z', Ts);

% define Forward Euler (FE) transformation
s = (z-1)/Ts;

% obtain FE discrete equivalent
sysGd6 = (-s+1)/(s^2 + 2*0.3*s + 1);
sysGd6 = minreal(sysGd6)
```

```
sysGd6 =

     -0.4 z + 0.56
  --------------------
  z^2 - 1.76 z + 0.92

Sample time: 0.4 seconds
Discrete-time transfer function.
```

BE transformation

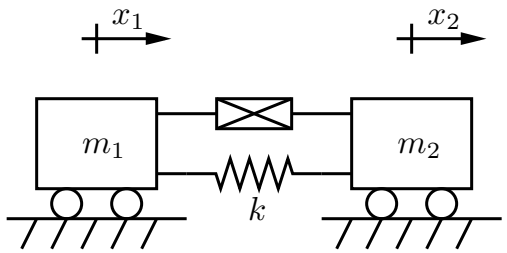
$$s = \frac{1-z^{-1}}{T_s}$$

```
% define Backward Euler (BE) transformation
s = (1-z^-1)/Ts;

% obtain BE discrete equivalent
sysGd7 = (-s+1)/(s^2 + 2*0.3*s + 1);
sysGd7 = minreal(sysGd7)
```

```
sysGd7 =

  -0.1714 z^2 + 0.2857 z
  ----------------------
   z^2 - 1.6 z + 0.7143

Sample time: 0.4 seconds
Discrete-time transfer function.
```

Controllability and observability

Two-mass system with *infrastructural* actuator



Force exerted by the actuator on the two carts

$$\dot{\boldsymbol{x}} = \boldsymbol{A}_1 \boldsymbol{x} + \boldsymbol{B}_1 u$$

$$\boldsymbol{A}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -k/m_1 & 0 & k/m_1 & 0 \\ 0 & 0 & 0 & 1 \\ k/m_2 & 0 & -k/m_2 & 0 \end{bmatrix}, \quad \boldsymbol{B}_1 = \begin{bmatrix} 0 \\ 1/m_1 \\ 0 \\ 1/m_2 \end{bmatrix}$$

$$\boldsymbol{x} = [x_1,\, \dot{x}_1,\, x_2,\, \dot{x}_2]^T$$

Controllability and observability

```
%  state-space model (state = [x1; dotx1; x2; dotx2])
m1 = 1; m2 = 1; k = 2;
A1 = [0, 1, 0, 0; ...
     -k/m1, 0, k/m1, 0; ...
     0, 0, 0, 1;
     k/m2, 0, -k/m2, 0];

B1 = [0; 1/m1; 0; -1/m2];
```

Use **`ctrb`** to evaluate the *controllability matrix*

$$\mathcal{R}_1 = \begin{bmatrix} B_1, & A_1 B_1, & A_1^2 B_1, & A_1^3 B_1 \end{bmatrix}$$

```
%  get controllability matrix [B, A*B, A^2*B, A^3*B]
Mc1 = ctrb(A1,B1)
```

```
Mc1 = 4×4
     0     1     0    -4
     1     0    -4     0
     0    -1     0     4
    -1     0     4     0
```

```
%  rank of controllability matrix
rank(Mc1)
```

```
ans = 2
```

```
%  number of uncontrollable states
Nc1 = size(Mc1,1) - rank(Mc1)
```

```
Nc1 = 2
```

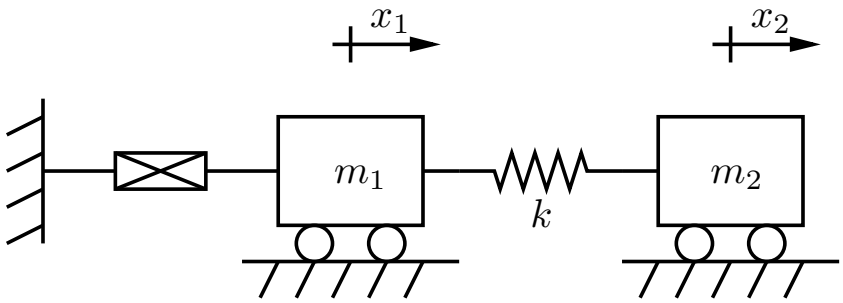
The controllability matrix is *not full rank*

⇓

with an infrastructural actuator, it is impossible to control the "common-mode" position and velocity; only the relative displacement and velocity of the two carts can be controlled.

Controllability and observability

Two-mass system with actuator *fixed to ground*



$$\dot{\boldsymbol{x}} = \boldsymbol{A}_1 \boldsymbol{x} + \boldsymbol{B}_1 u$$

$$\boldsymbol{A}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -k/m_1 & 0 & k/m_1 & 0 \\ 0 & 0 & 0 & 1 \\ k/m_2 & 0 & -k/m_2 & 0 \end{bmatrix}, \quad \boldsymbol{B}_2 = \begin{bmatrix} 0 \\ 1/m_1 \\ 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{x} = [x_1,\, \dot{x}_1,\, x_2,\, \dot{x}_2]^T$$

Controllability and observability

```
% state-space model
A2 = A1;
B2 = [0; 1/m1; 0; 0];
```

```
% get controllability matrix [B, A*B, A^2*B, A^3*B]
Mc2 = ctrb(A2, B2)
```

```
Mc2 = 4×4
     0     1     0    -2
     1     0    -2     0
     0     0     0     2
     0     0     2     0
```

```
%  rank of controllability matrix
rank(Mc2)
```

```
ans = 4
```

```
% number of uncontrollable states
Nc2 = size(Mc2,1) - rank(Mc2)
```

```
Nc2 = 0
```

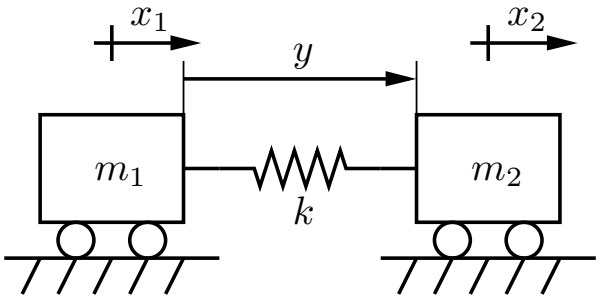
The controllability matrix is *full rank*

⇓

with the actuator fixed to ground, the system becomes fully controllable.

Controllability and observability

Two-mass system with *infrastructural position sensor*



$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}_3\,\boldsymbol{x} \\ y = \boldsymbol{C}_3\,\boldsymbol{x} \end{cases}$$

$$\boldsymbol{A}_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -k/m_1 & 0 & k/m_1 & 0 \\ 0 & 0 & 0 & 1 \\ k/m_2 & 0 & -k/m_2 & 0 \end{bmatrix}, \quad \boldsymbol{C}_3 = \begin{bmatrix} -1 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{x} = [x_1,\, \dot{x}_1,\, x_2,\, \dot{x}_2]^T$$

Controllability and observability

```
% state-space model
A3 = A1;
C3 = [-1, 0, 1, 0];
```

```
% get observability matrix [C; C*A, C*A
Mo3 = obsv(A3, C3)
```

```
Mo3 = 4×4
    -1     0     1     0
     0    -1     0     1
     4     0    -4     0
     0     4     0    -4
```

Use **`obsv`** to evaluate the *observability matrix*

$$\mathcal{O}_3 = \begin{bmatrix} C_3 \\ C_3 A_3 \\ C_3 A_3^2 \\ C_3 A_3^3 \end{bmatrix}$$

```
%  rank of controllability matrix
rank(Mo3)
```

```
ans = 2
```

```
% number of unobservable states
No3 = size(Mo3,2) - rank(Mo3)
```

```
No3 = 2
```

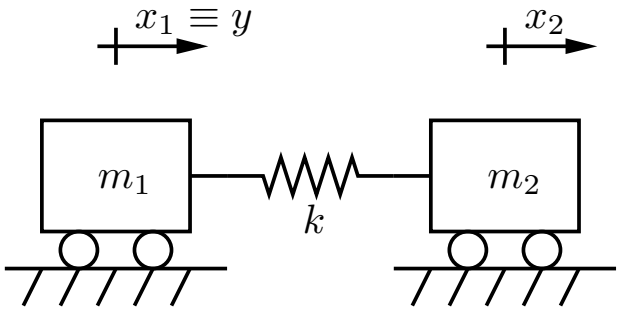
The observability matrix is *not full rank*

⇓

with an infrastructural sensor, it is impossible to observe the "common-mode" position and velocity; only the relative displacement and velocity of the two carts can be observed.

Controllability and observability

Two-mass system with *absolute position sensor* attached to one cart



$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}_4\, \boldsymbol{x} \\ y = \boldsymbol{C}_4\, \boldsymbol{x} \end{cases}$$

$$\boldsymbol{A}_4 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -k/m_1 & 0 & k/m_1 & 0 \\ 0 & 0 & 0 & 1 \\ k/m_2 & 0 & -k/m_2 & 0 \end{bmatrix}, \quad \boldsymbol{C}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{x} = [x_1,\, \dot{x}_1,\, x_2,\, \dot{x}_2]^T$$

Controllability and observability

```
% state space model
A4 = A1;
C4 = [1, 0, 0, 0];
```

```
% get observability matrix [C; C*A, C*A^2; C*A^3]
Mo4 = obsv(A4, C4)
```

```
Mo4 = 4×4
     1     0     0     0
     0     1     0     0
    -2     0     2     0
     0    -2     0     2
```

```
%  rank of controllability matrix
rank(Mo4)
```

```
ans = 4
```

```
% number of unobservable states
No4 = size(Mo4,2) - rank(Mo4)
```

```
No4 = 0
```

The observability matrix is *full rank*

$\Downarrow$

by measuring the absolute position of the first cart, the system becomes fully observable.

Pole placement design

- Use **`place`** to design the state-feedback gain $\boldsymbol{K}$ that places the closed-loop eigenvalues of the controllable pair $(\boldsymbol{A}, \boldsymbol{B})$, i.e. the eigenvalues of $\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}$, to the desired locations on complex plane.

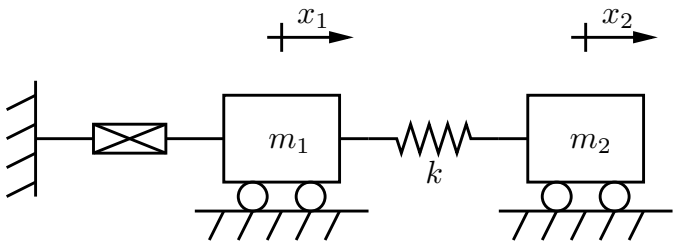
  The routine works for both SISO and MIMO systems, but requires that the *multiplicity of the closed-loop eigenvalues is* $<$ $\operatorname{rank} \boldsymbol{B}$.

- **`acker`** is an alternative routine, and allows to allocate eigenvalues with multiplicity > 1. However, it *only works for SISO models*, and is *less numerically reliable*.

Pole placement design

```
%  open-loop eigenvals
lam2 = eig(A2);

%  poles sorted in descending order of real part
lam2 = esort(lam2)
```

```
lam2 = 4×1 complex
     0.0000 + 0.0000i
    -0.0000 + 2.0000i
    -0.0000 - 2.0000i
    -0.0000 + 0.0000i
```



Desired closed-loop eigenvalues

$$\lambda_{d,1} = e^{j(-\pi+\pi/3)} = -\frac{1}{2} + j\,\frac{\sqrt{3}}{2}, \qquad \lambda_{d,4} = \lambda_{d,1}^{*} = -\frac{1}{2} - j\,\frac{\sqrt{3}}{2}$$

$$\lambda_{d,2} = e^{j(-\pi+\pi/6)} = -\frac{\sqrt{3}}{2} + j\,\frac{1}{2}, \qquad \lambda_{d,3} = \lambda_{d,2}^{*} = -\frac{\sqrt{3}}{2} - j\,\frac{1}{2}$$

```
% desired closed-loop eigenvals
lamd1 = 1.0 * exp(1i*(-pi + pi/3));
lamd2 = 1.0 * exp(1i*(-pi + pi/6));
lamd3 = conj(lamd2);
lamd4 = conj(lamd1);
lamd = [lamd1, lamd2, lamd3, lamd4];
```

```
% state feedback control design
K = place(A2, B2, lamd)
```

```
K = 1×4
    -0.2679    2.7321    0.7679   -1.3660
```

The state-feedback matrix $\boldsymbol{K}$ can be obtained with the **`place`** routine (all the eigenvalues have multiplicity = 1).

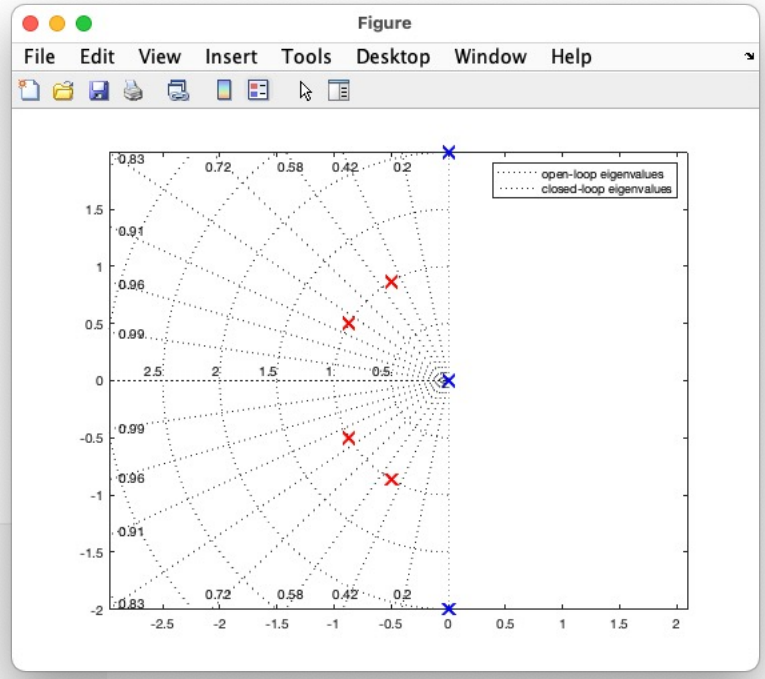
Pole placement design

```
%  desired closed-loop eigenvals
%  (sorted in descending order of real parts)
lamd = esort(lamd)
```

```
lamd = 4×1 complex
    -0.5000 + 0.8660i
    -0.5000 - 0.8660i
    -0.8660 + 0.5000i
    -0.8660 - 0.5000i
```

```
%  closed-loop eigenvals placed by "place"
A2c = A2 - B2*K;
lamc = eig(A2c)
```

```
lamc = 4×1 complex
    -0.5000 + 0.8660i
    -0.5000 - 0.8660i
    -0.8660 + 0.5000i
    -0.8660 - 0.5000i
```




```
% plot open vs closed loop eigenvals
figure;
plot(lam2, 'bx', 'MarkerSize', 10, 'LineWidth', 2);
hold on
plot(lamc, 'rx', 'MarkerSize', 10, 'LineWidth', 2);
axis equal;
sgrid;
legend('open-loop eigenvalues', 'closed-loop eigenvalues');
```

Pole placement design

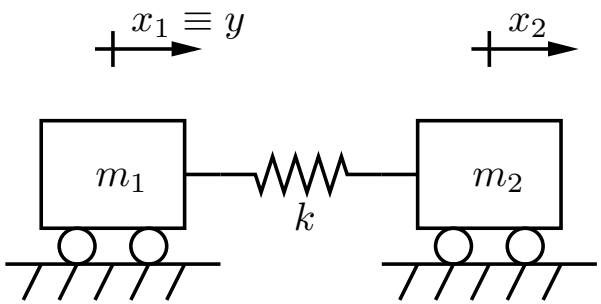
Note: by *duality theorem*, the **`place`** and **`acker`** routines can be also used to design the state estimator gain matrix $\boldsymbol{L}$ that places the estimator eigenvalues, i.e. the eigenvalues of $\boldsymbol{A} - \boldsymbol{LC}$, to the desired locations on complex plane.

The estimator gain matrix $\boldsymbol{L}$ for the pair $(\boldsymbol{A}, \boldsymbol{C})$ is designed as the controller gain matrix $\mathbf{K} = \boldsymbol{L}^{\boldsymbol{T}}$ for the dual system $(\boldsymbol{A}^{\boldsymbol{T}}, \boldsymbol{C}^{\boldsymbol{T}})$.

Pole placement design

```
%  open-loop eigenvals
lam4 = eig(A4);

%  poles sorted in descending order of real part
lam4 = esort(lam4)
```

```
lam4 = 4×1 complex
     0.0000 + 0.0000i
    -0.0000 + 2.0000i
    -0.0000 - 2.0000i
    -0.0000 + 0.0000i
```




```
% desired estimator eigenvals
lamd = -5 * ones(4,1);
```

Desired estimator eigenvalues

$$\lambda_{d,1} = \lambda_{d,2} = \lambda_{d,3} = \lambda_{d,4} = -5$$

```
% state estimator design
L = acker(A4.', C4.', lamd).'
```

```
L = 4×1
    20.0000
   146.0000
   230.0000
   166.5000
```

The state estimator matrix $\boldsymbol{L}$ can be obtained with the **`acker`** routine (there is only one desired eigenvalue with multiplicity $= 4$).

Linear Quadratic Regulator (LQR)

For models in *encapsulated form* (i.e. LTI objects):

↳ **`lqr`** : LQR design with state + input weighting.

↳ **`lqry`** : LQR design with output + input weighting.

↳ **`lqi`** : LQR design with *integral action*.

Note: they accept both continuous and discrete-time models.

For models in *non-encapsulated* form:

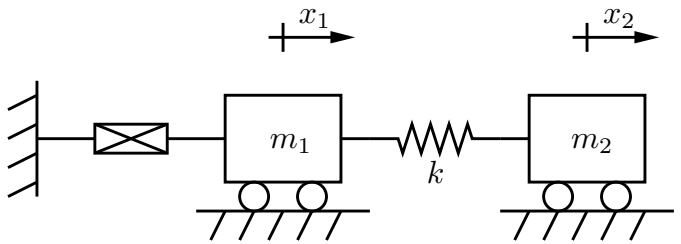
↳ **`lqr`**, **`lqry`**, **`lqi`** : LQR designs for continuous-time models.

↳ **`dlqr`**, **`dlqry`** : LQR designs for discrete-time models.

Linear Quadratic Regulator (LQR)

LQR cost function

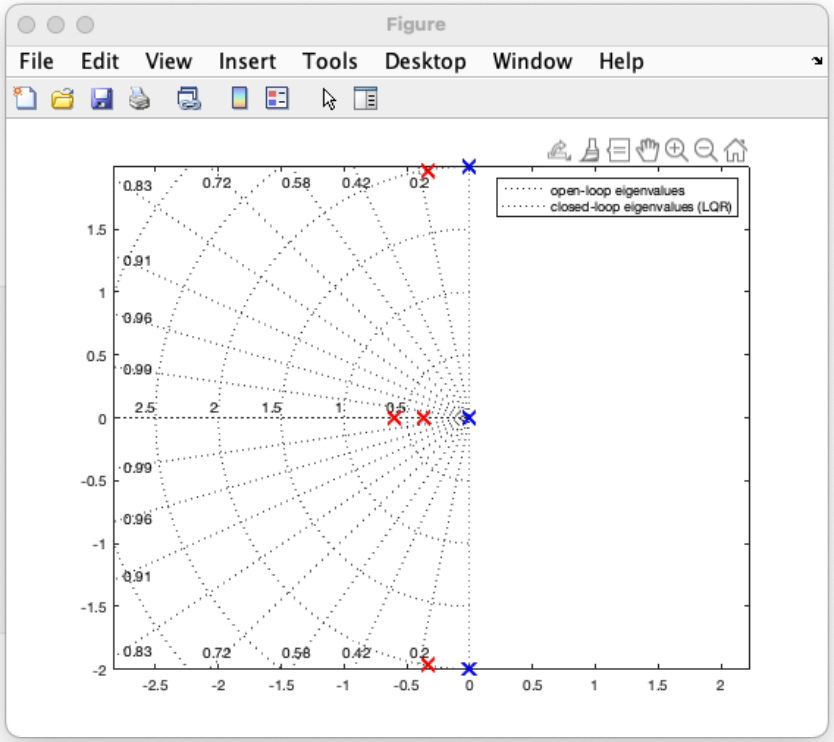
$$J = \int_0^{+\infty} \boldsymbol{x}^T(t)\,\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^T(t)\,\boldsymbol{R}\,\boldsymbol{u}(t)\,dt$$




```
% cost function weights
Q = diag([1, 10, 1, 10]);   % state weighting matrix
R = 10;                     % input weight

% LQR design
K = lqr(A2, B2, Q, R)
```

```
K = 1×4
    0.8615    1.6501   -0.4143    0.3683
```

```
% plot open vs closed loop eigenvals
figure;
plot(lam2, 'bx', 'MarkerSize', 10, 'LineWidth', 2);
hold on
plot(lamc, 'rx', 'MarkerSize', 10, 'LineWidth', 2);
axis equal;
sgrid;
legend('open-loop eigenvalues', 'closed-loop eigenvalues
```



sisotool

sisotool is a convenient interactive tool for the analysis and design of SISO control systems.