UNIVERSITÀ
DEGLI STUDI
DI PADOVA

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

- Control System Toolbox (CST).
- LTI objects.
- Data encapsulation.
- Conversions between LTI model representations.
- Discrete-time LTI objects.
- Connections of LTI models.

Preliminaries

A MATLAB *Total Academic Headcount* (TAH) *License* is available for all the students and employees of University of Padova.

The license allows to install a full copy of MATLAB and companion toolboxes on personally-owned computers.

Instructions for downloading and installing the software can be found here:

https://www.csia.unipd.it/servizi/servizi-utenti-istituzionali/contratti-software-e-licenze/matlab

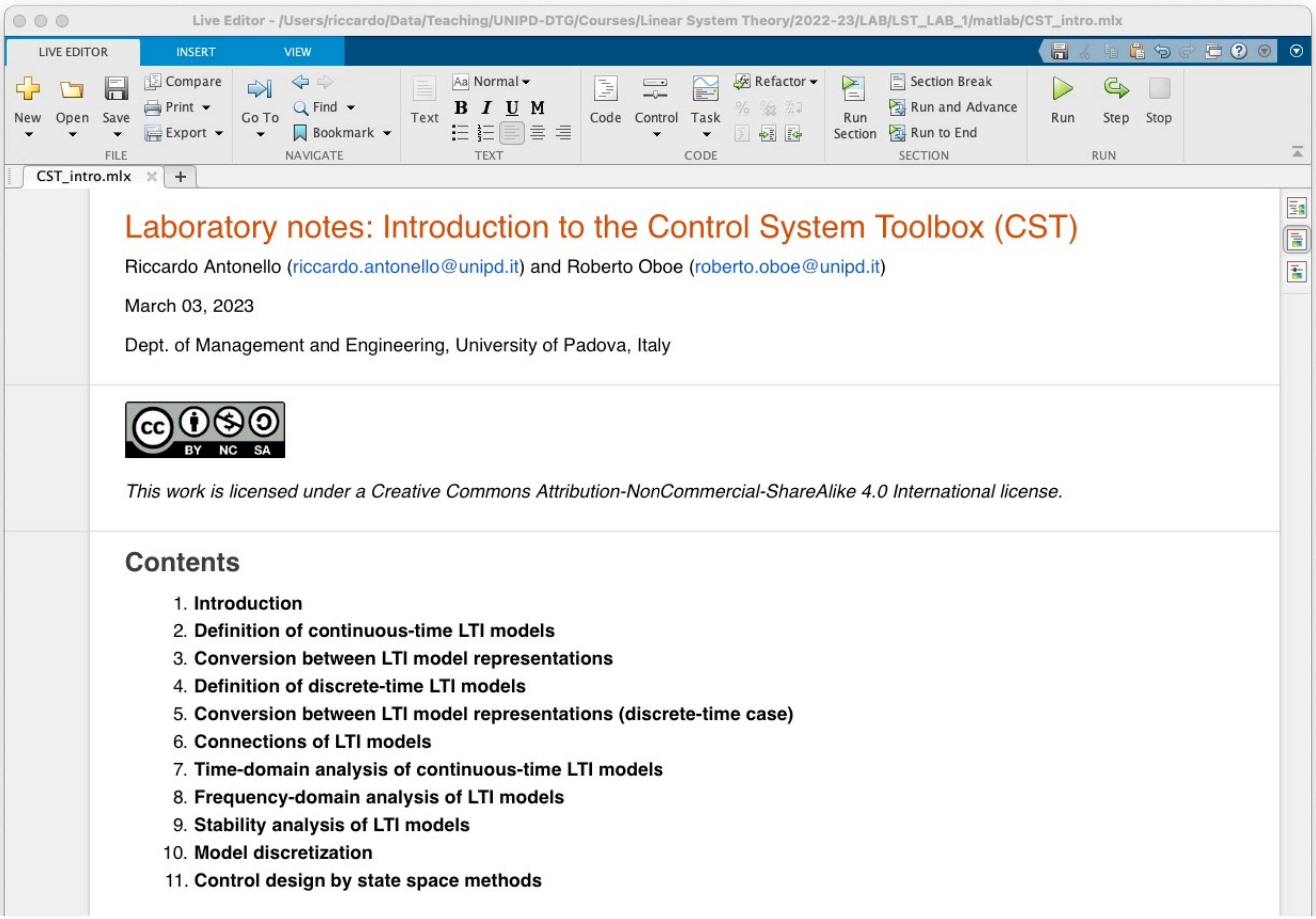
Preliminaries

- MATLAB version used in this course: **R2022b**
- Required toolbox: **Control System Toolbox**

```
Command Window
New to MATLAB? See resources for Getting Started.
>> ver
-----------------------------------------------------------------------------------------------------
MATLAB Version: 9.11.0.1837725 (R2021b) Update 2
MATLAB License Number: 40523914
Operating System: macOS  Version: 12.6.3 Build: 21G419
Java Version: Java 1.8.0_202-b08 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode
-----------------------------------------------------------------------------------------------------
MATLAB                                                Version 9.11        (R2021b)
Simulink                                              Version 10.4        (R2021b)
Control System Toolbox                                Version 10.11       (R2021b)
Curve Fitting Toolbox                                 Version 3.6         (R2021b)
DSP System Toolbox                                    Version 9.13        (R2021b)
Embedded Coder                                        Version 7.7         (R2021b)
Fixed-Point Designer                                  Version 7.3         (R2021b)
Image Processing Toolbox                              Version 11.4        (R2021b)
Instrument Control Toolbox                            Version 4.5         (R2021b)
MATLAB Coder                                          Version 5.3         (R2021b)
MATLAB Compiler                                       Version 8.3         (R2021b)
MATLAB Compiler SDK                                   Version 6.11        (R2021b)
Optimization Toolbox                                  Version 9.2         (R2021b)
ROS Toolbox                                           Version 1.4         (R2021b)
Robotics System Toolbox                               Version 3.4         (R2021b)
Robust Control Toolbox                                Version 6.11        (R2021b)
Signal Processing Toolbox                             Version 8.7         (R2021b)
```

Preliminaries

All the material presented in these notes is available as a MATLAB *Live Script* on Moodle.



Control System Toolbox (CST)

The **Control System Toolbox (CST)** is a MATLAB toolbox conceived for the *analysis, design and tuning* of *linear, time-invariant* (*LTI*) control systems.

Only LTI systems are supported by CST; they can be

↳ either SISO or MIMO;

↳ either continuous-time or discrete-time (but not "hybrid-time/sampled-data" or "multi-rate" !).

LTI objects

CST uses data structures called **LTI objects** to store model-related data.

An LTI object is an object according to the common definition of the *object-oriented programming (OOP) paradigm*, i.e. a data structure that *encapsulates* both *data* and *functions* operating on such data[(1)].

(1)	C++ :	**member data / member function**
	Java :	**attribute / method**
	MATLAB :	***property / method***

LTI objects

An LTI object can be defined starting from different representations of the same dynamical system.

Accepted representations are:

1. Transfer function numerator / denominator (**TF**)
2. Transfer function zeros, poles and gain (**ZPK**)
3. State space model (**SS**)
4. Frequency response data (**FRD**)

Continuous-time TF objects

$$G(s) = \frac{5s + 50}{s^2 + 101s + 100} = 5\,\frac{s + 10}{(s + 1)(s + 100)}$$

1) Definition by using a **TF object**:

```
%   non-encapsulated TF data
num = 5 * [1, 10];
den = [1, 101, 100];

%   TF data encapulated in a TF object
sysG1 = tf(num,den)
```

```
sysG1 =

      5 s + 50
  -----------------
  s^2 + 101 s + 100

Continuous-time transfer function.
```

LTI model data in *non-encapsulated* form (*numerator* and *denominator* polynomials)

LTI model data in *encapsulated* form (*LTI object – TF form*)

Continuous-time TF objects

Data encapsulation:

```
%  show all the LTI object fields
get(sysG1)
```

```
      Numerator: {[0 5 50]}
    Denominator: {[1 101 100]}
       Variable: 's'
        IODelay: 0
     InputDelay: 0
    OutputDelay: 0
             Ts: 0
       TimeUnit: 'seconds'
      InputName: {''}
      InputUnit: {''}
     InputGroup: [1×1 struct]
     OutputName: {''}
     OutputUnit: {''}
    OutputGroup: [1×1 struct]
          Notes: [0×1 string]
       UserData: []
           Name: ''
   SamplingGrid: [1×1 struct]
```

An LTI object *encapsulates* all the relevant information pertaining the representation (in a certain form) of an LTI model.

```
%   access a particular field
sysG1.Numerator{1}
```

```
ans = 1×3
     0     5    50
```

Data fields can be accessed by using the "dot-notation".

Continuous-time TF objects

$$G(s) = \frac{5s + 50}{s^2 + 101s + 100} = 5\,\frac{s + 10}{(s+1)(s+100)}$$

Alternative: define the LTI object for the *Laplace variable s*, and use the operators +, ∗, / defined (i.e. *overloaded*) for the class of LTI objects.

```
%   define the Laplace variable "s" (LTI object)
s = tf('s');

%   define the TF object by using the "s" variable and the +, *, / operators
sysG2 = 5*(s+10)/((s+1)*(s+100))
```

```
sysG2 =

      5 s + 50
  -----------------
  s^2 + 101 s + 100

Continuous-time transfer function.
```

Continuous-time ZPK objects

$$G(s) = \frac{5s + 50}{s^2 + 101s + 100} = 5\,\frac{s + 10}{(s+1)(s+100)}$$

2) Definition by using a **ZPK object**:

```
%   non-encapsulated ZPK data
z = -10;
p = [-1, -100];
k = 5;

%   ZPK data encapulated in a ZPK object
sysG3 = zpk(z,p,k)
```

LTI model data in *non-encapsulated* form (*zeros*, *poles* and *gain*)

```
sysG3 =

    5 (s+10)
  -------------
  (s+1) (s+100)

Continuous-time zero/pole/gain model.
```

LTI model data in *encapsulated* form (*LTI object – ZPK form*)

Continuous-time SS objects

$$G(s) = \frac{5s + 50}{s^2 + 101s + 100} = 5\,\frac{s + 10}{(s+1)(s+100)}$$

⇩

Possible state space realization
(in *controllable canonical form*)

$$A = \begin{bmatrix} 0 & 1 \\ -100 & -101 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 50 & 5 \end{bmatrix}, \qquad D = 0$$

Continuous-time SS objects

```
%   non-encapsulated SS data
A = [0, 1; -100, -101];
B = [0; 1];
C = [50, 5];
D = 0;

%   SS data encapulated in a SS object
sysG4 = ss(A, B, C, D)
```

```
sysG4 =

  A =
          x1    x2
   x1      0     1
   x2   -100  -101

  B =
        u1
   x1    0
   x2    1

  C =
        x1  x2
   y1   50   5

  D =
        u1
   y1    0

Continuous-time state-space model.
```

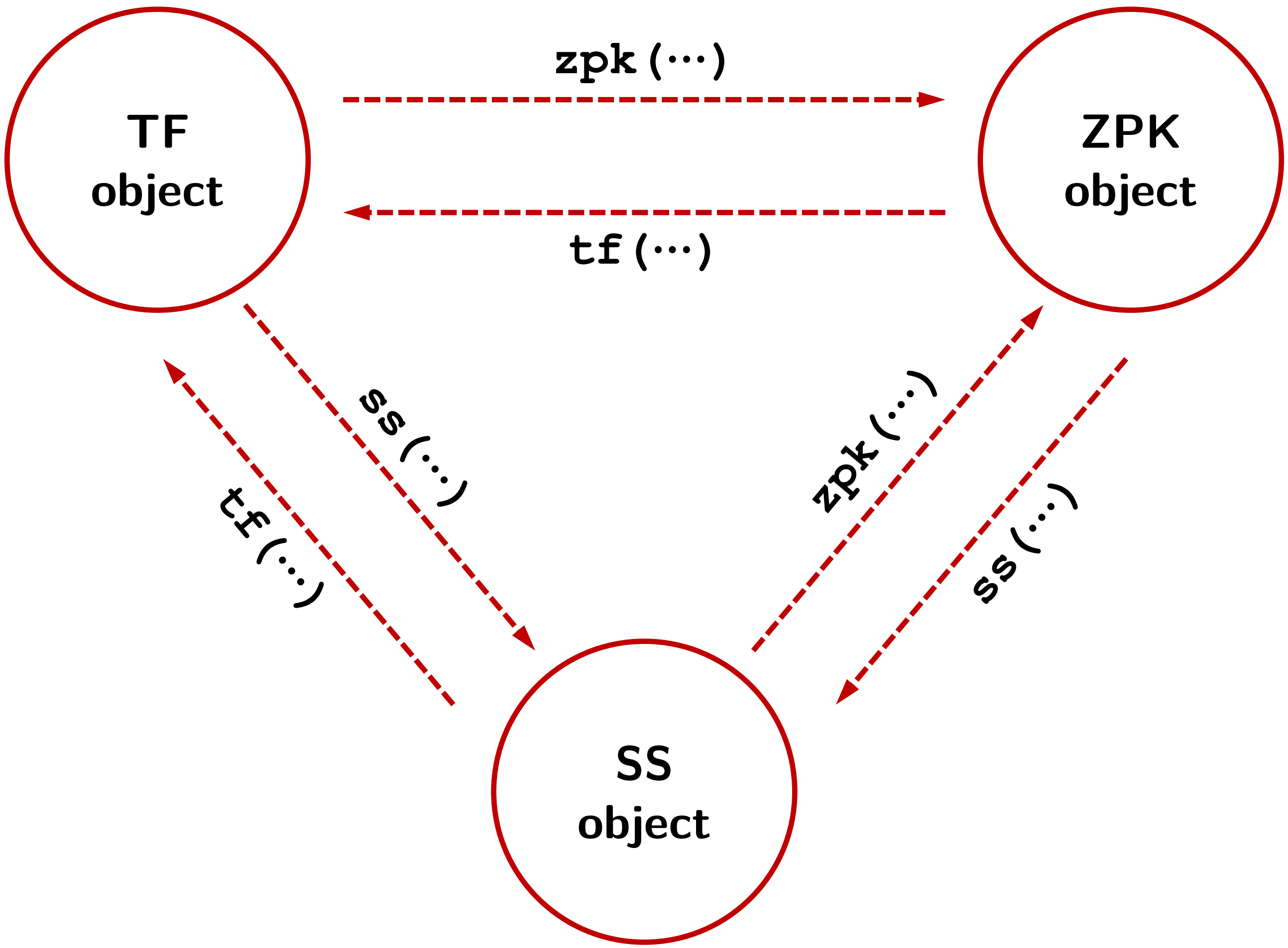
LTI model data in *non-encapsulated* form (*state space matrices*)

$$A = \begin{bmatrix} 0 & 1 \\ -100 & -101 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 50 & 5 \end{bmatrix}, \qquad D = 0$$

LTI model data in *encapsulated* form (*LTI object – SS form*)

Conversions between model representations (*encapsulated form*)



Conversions between model representations (*encapsulated form*)

```
%   remind: sysG1 is a TF obj
sysG1
```

```
sysG1 =

      5 s + 50
  -----------------
  s^2 + 101 s + 100

Continuous-time transfer function.
```

zpk(⋯) →

```
%   from TF to ZPK object
sysG1a = zpk(sysG1)
```

```
sysG1a =

    5 (s+10)
  ------------
  (s+100) (s+1)

Continuous-time zero/pole/gain model.
```

ss(⋯) ↘

```
%   from TF to SS object
sysG1b = ss(sysG1)
```

```
sysG1b =

  A =
           x1     x2
   x1    -101  -12.5
   x2       8      0

  B =
         u1
   x1     4
   x2     0

  C =
           x1     x2
   y1    1.25  1.562

  D =
         u1
   y1     0

Continuous-time state-space model.
```

Note: The state-space realization is selected by `ss(⋯)`.

It is not necessarily a *canonical realization* (e.g. *controllable*, *observable*, *modal*).

Conversions between model representations (*encapsulated form*)

```
%  from ZPK to TF object
sysG3a = tf(sysG3)
```

```
sysG3a =
 
      5 s + 50
  -----------------
  s^2 + 101 s + 100
 
Continuous-time transfer function.
```

tf(···)

```
%  remind: sysG3 is a ZPK obj
sysG3
```

```
sysG3 =
 
     5 (s+10)
  --------------
  (s+1) (s+100)
 
Continuous-time zero/pole/gain model.
```

ss(···)

```
%  from ZPK to SS object
sysG3b = ss(sysG3)
```

```
sysG3b =
 
  A = 
          x1    x2
   x1     -1     3
   x2      0  -100
 
  B = 
        u1
   x1    0
   x2    4
 
  C = 
          x1    x2
   y1   3.75  1.25
 
  D = 
        u1
   y1    0
 
Continuous-time state-space model.
```

Note: The state-space realization is selected by `ss(···)`.

It is not necessarily a *canonical realization* (e.g. *controllable*, *observable*, *modal*).

Conversions between model representations (*encapsulated form*)

```
% from SS to TF object
sysG4a = tf(sysG4)
```

```
sysG4a =

      5 s + 50
  -----------------
  s^2 + 101 s + 100

Continuous-time transfer function.
```

tf(...)

```
% from SS to ZPK object
sysG4b = zpk(sysG4)
```

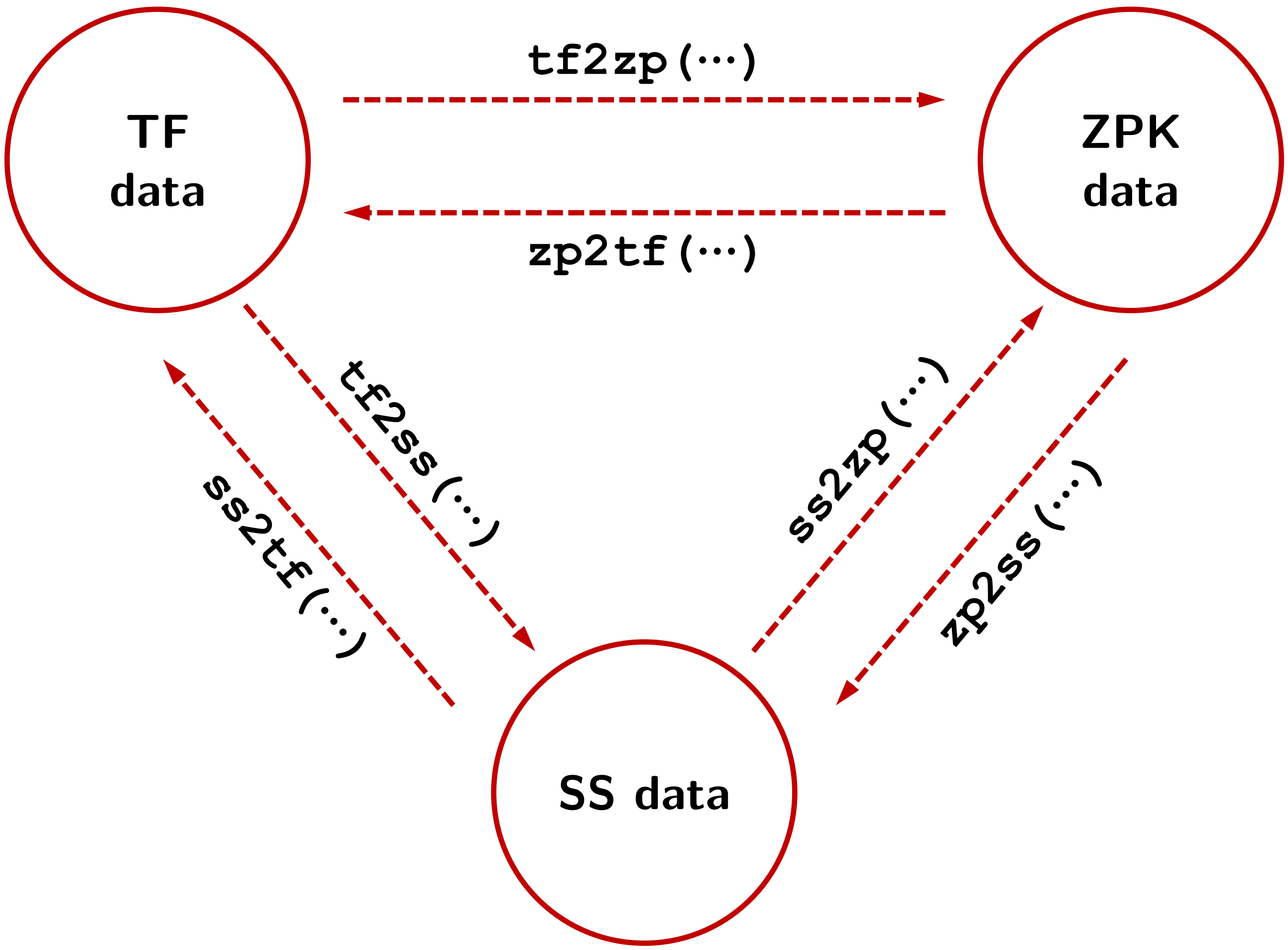
```
sysG4b =

     5 (s+10)
  -------------
  (s+1) (s+100)

Continuous-time zero/pole/gain model.
```

zpk(...)

```
% remind: sysG4 is a SS obj
sysG4
```

```
sysG4 =

  A =
         x1    x2
   x1     0     1
   x2  -100  -101

  B =
       u1
   x1   0
   x2   1

  C =
       x1  x2
   y1  50   5

  D =
       u1
   y1   0

Continuous-time state-space model.
```

Conversions between model representations (*non-encapsulated form*)



Conversions between model representations (*non-encapsulated form*)

```
%  TF data
num = 5 * [1, 10];
den = [1, 101, 100];
```

tf2zp(···)

```
%  from TF to ZPK data
[z,p,k] = tf2zp(num,den)
```

```
z = -10
 p = 2x1
     -100
       -1

k = 5
```

tf2ss(···)

```
%  from TF to SS data
[A,B,C,D] = tf2ss(num,den)
```

```
A = 2x2
    -101  -100
       1     0

B = 2x1
       1
       0

C = 1x2
       5    50

D = 0
```

Note: similar to a *controllable canonical form*, with the *characteristic polynomial* on the 1^{st} row of A (instead of last row).

Conversions between model representations (*non-encapsulated form*)

```
% ZPK data
z = -10;
p = [-1, -100];
k = 5;
```

```
%  from ZPK to TF data
[num,den] = zp2tf(z,p,k)
```

```
num = 1×3
     0     5    50

den = 1×3
     1   101   100
```

`zp2tf(···)`

`zp2ss(···)`

```
%  from ZPK to SS data
[A,B,C,D] = zp2ss(z,p,k)
```

```
A = 2×2
   -101   -10
     10     0

B = 2×1
      1
      0

C = 1×2
      5     5

D = 0
```

Note: not a *controllable canonical form.*

Conversions between model representations (*non-encapsulated form*)

```
%  from SS to TF data
[num,den] = ss2tf(A,B,C,D)
```

```
num = 1×3
        0     5    50

den = 1×3
        1   101   100
```

```
%  from SS to ZPK data
[z,p,k] = ss2zp(A,B,C,D)
```

```
z = -10
 p = 2×1
        -1
      -100

k = 5
```

ss2tf(...)

ss2zp(...)

```
%  SS data
A = [0, 1; -100, -101];
B = [0; 1];
C = [50, 5];
D = 0;
```

From encapsulated to non-encapsulated forms

TF object ⟶ TF data

```
%  TF object
sysG1
```

```
sysG1 =

      5 s + 50
  -----------------
  s^2 + 101 s + 100

Continuous-time transfer function.
```

```
%  from TF object to TF data
[num,den] = tfdata(sysG1, 'v')
```

```
num = 1×3
     0     5    50

den = 1×3
     1   101   100
```

ZPK object ⟶ ZPK data

```
%  ZPK object
sysG2
```

```
sysG2 =

      5 s + 50
  -----------------
  s^2 + 101 s + 100

Continuous-time transfer function.
```

```
%  from ZPK object to ZPK data
[z,p,k]   = zpkdata(sysG2, 'v')
```

```
z = -10
p = 2×1
   -100
     -1

k = 5
```

SS object ⟶ SS data

```
%  SS data
sysG3
```

```
sysG3 =

     5 (s+10)
  -------------
  (s+1) (s+100)

Continuous-time zero/pole/gain model.
```

```
%  from SS object to SS data
[A,B,C,D] = ssdata(sysG3)
```

```
A = 2×2
      -1     3
       0  -100

B = 2×1
       0
       4

C = 1×2
     3.7500    1.2500

D = 0
```

Note: for a SISO model, the **'v'** option returns row vectors, instead of cell arrays.

Conversion between state-space representations

$$\Sigma = (A, B, C, D)$$

$x' = Tx$ ⇒ State transformation

$$\Sigma' = (A', B', C', D') = (TAT^{-1}, TB, CT^{-1}, D)$$

```
% SS object
sysG4
```

```
sysG4 =

  A =
         x1    x2
   x1     0     1
   x2  -100  -101

  B =
       u1
   x1   0
   x2   1

  C =
       x1  x2
   y1  50   5

  D =
       u1
   y1   0

Continuous-time state-space model.
```

```
% state transformation matrix
T = [1, 1; 1, -1];

% convert to new state z=T*x
sysG5 = ss2ss(sysG4, T)
```

```
sysG5 =

  A =
         x1    x2
   x1  -100     0
   x2   101    -1

  B =
       u1
   x1   1
   x2  -1

  C =
         x1    x2
   y1  27.5  22.5

  D =
       u1
   y1   0

Continuous-time state-space model.
```

Discrete-time LTI objects

Discrete-time LTI objects are defined similarly to continuous-time objects (e.g. by using the **`tf`**, **`zpk`**, **`ss`** routines), but a *sampling time* $T_s > 0$ must be specified.

<u>Convention</u>:

$T_s > 0$ $\Rightarrow$ discrete-time model.

$T_s = 0$ $\Rightarrow$ continuous-time model.

$T_s = -1$ $\Rightarrow$ discrete-time model with "unspecified" sampling time.

Discrete-time TF objects

$$H(z) = \frac{5z - 1.5}{z^2 + 0.1z - 0.02} = 5\,\frac{z - 0.3}{(z - 0.1)(z + 0.2)} \qquad T_s = 1$$

```
%  sampling time
Ts = 1;

%  non-encapsulated TF data
num = 5 * [1, -0.3];
den = conv([1, -0.1], [1, 0.2]);

%  TF data encapsulated in a TF object
sysH1 = tf(num, den, Ts)
```

```
sysH1 =

      5 z - 1.5
  ------------------
  z^2 + 0.1 z - 0.02

Sample time: 1 seconds
Discrete-time transfer function.
```

LTI model data in *non-encapsulated* form (*numerator* and *denominator* polynomials)

LTI model data in *encapsulated* form (*LTI object – TF form*)

Discrete-time TF objects

$$H(z) = \frac{5z - 1.5}{z^2 + 0.1z - 0.02} = 5\,\frac{z - 0.3}{(z-0.1)(z+0.2)} \qquad T_s = 1$$

Alternative: define the LTI object for the $\mathcal{Z}$*-transform variable* z, and use the operators +, $*$, / defined (i.e. *overloaded*) for the class of LTI objects.

```
%   define the Z-transform variable "z" (LTI object)
z = tf('z', Ts);

%   define the TF object by using the "z" variable and the +, *, / operators
sysH2 = 5*(z-0.3)/((z-0.1)*(z+0.2))
```

```
sysH2 =

      5 z - 1.5
  ------------------
  z^2 + 0.1 z - 0.02

Sample time: 1 seconds
Discrete-time transfer function.
```

Discrete-time ZPK objects

$$H(z) = \frac{5z - 1.5}{z^2 + 0.1z - 0.02} = 5\,\frac{z - 0.3}{(z-0.1)(z+0.2)} \qquad T_s = 1$$

```
%  non-encapsulated ZPK data
z = 0.3;
p = [0.1, -0.2];
k = 5;

%  ZPK data encapsulated in a ZPK object
sysH4 = zpk(z, p, k, Ts)
```

LTI model data in *non-encapsulated* form (*zeros*, *poles* and *gain*)

```
sysH4 =

     5 (z-0.3)
  ---------------
  (z-0.1) (z+0.2)

Sample time: 1 seconds
Discrete-time zero/pole/gain model.
```

LTI model data in *encapsulated* form (*LTI object – ZPK form*)

Discrete-time SS objects

$$H(z) = \frac{5z - 1.5}{z^2 + 0.1z - 0.02} = 5\,\frac{z - 0.3}{(z - 0.1)(z + 0.2)} \qquad T_s = 1$$

⇩

Possible state space realization
(in *controllable canonical form*)

$$A = \begin{bmatrix} 0 & 1 \\ -0.02 & 0.1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} -1.5 & 5 \end{bmatrix}, \qquad D = 0$$

Discrete-time SS objects

```
%  non-encapsulated SS data
A = [0, 1; -0.02, 0.1];
B = [0; 1];
C = 5 * [-0.3, 1];
D = 0;

%  SS data encapsulated in a SS object
sysH5 = ss(A, B, C, D, Ts)
```

```
sysH5 =

  A =
           x1      x2
   x1       0       1
   x2   -0.02     0.1

  B =
        u1
   x1    0
   x2    1

  C =
          x1     x2
   y1   -1.5      5

  D =
        u1
   y1    0

Sample time: 1 seconds
Discrete-time state-space model.
```

LTI model data in *non-encapsulated* form (*state space matrices*)

$$A = \begin{bmatrix} 0 & 1 \\ -0.02 & 0.1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} -1.5 & 5 \end{bmatrix}, \qquad D = 0$$

LTI model data in *encapsulated* form (*LTI object – SS form*)

Conversions between discrete-time LTI model representations

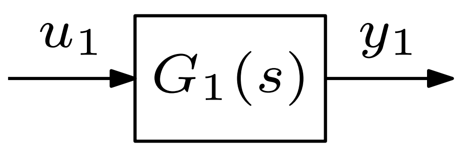
Conversions among discrete-time LTI model representations are performed similarly to continuous-time:

↳ *Encapsulated* data: use **`tf`**, **`zpk`**, **`ss`**

↳ *Non-encapsulated* data: use **`tf2zp`**, **`tf2ss`**, **`zp2ss`**, ...

Connections of LTI models

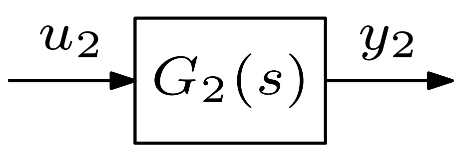
Consider the following two LTI models:



$$G_1(s) = \frac{1}{s(s+1)}$$

```
sysG1 = zpk([], [0, -1], 1)
```

```
sysG1 =

      1
   -------
   s (s+1)

Continuous-time zero/pole/gain model.
```



$$G_2(s) = \frac{s+1}{s+2}$$

```
sysG2 = tf([1,1], [1,2])
```

```
sysG2 =

   s + 1
   -----
   s + 2

Continuous-time transfer function.
```

Connections of LTI models

- Parallel connection:

use the operator " + "

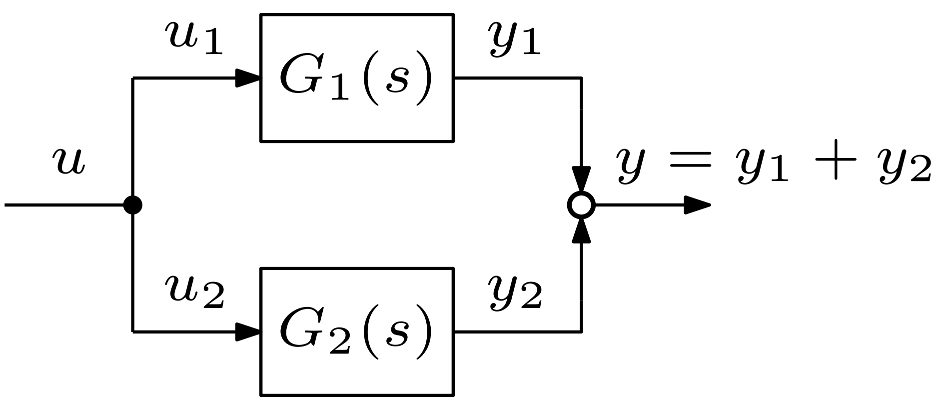
```
sysGp = sysG1 + sysG2
```

```
sysGp =

  (s+1.544) (s^2 + 0.4563s + 1.296)
  ---------------------------------
          s (s+1) (s+2)

Continuous-time zero/pole/gain model.
```

... or the function **parallel** of the CST

```
sysGp = parallel(sysG1, sysG2)
```

```
sysGp =

  (s+1.544) (s^2 + 0.4563s + 1.296)
  ---------------------------------
          s (s+1) (s+2)

Continuous-time zero/pole/gain model.
```



$$G_p(s) = G_1(s) + G_2(s) = \frac{1}{s(s+1)} + \frac{s+1}{s+2} = \frac{s^3 + 2s^2 + 2s + 2}{s(s+1)(s+2)}$$

Connections of LTI models

- Series connection:

use the operator "*"

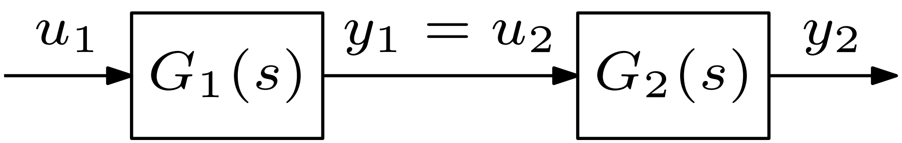
```
sysGs = sysG1 * sysG2
```

```
sysGs =

       (s+1)
  -------------
  s (s+1) (s+2)

Continuous-time zero/pole/gain model.
```

... or the function **series** of the CST

```
sysGs = series(sysG1, sysG2)
```

```
sysGs =

       (s+1)
  -------------
  s (s+2) (s+1)

Continuous-time zero/pole/gain model.
```



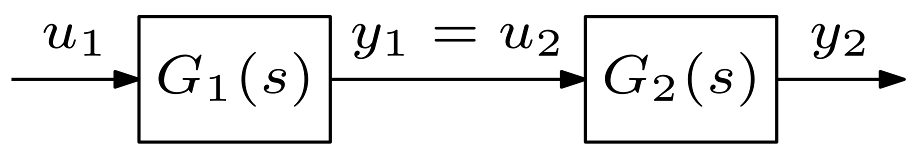
$$G_s(s) = G_1(s)\,G_2(s) = \frac{\cancel{s+1}}{s\cancel{(s+1)}(s+2)} = \frac{1}{s(s+2)}$$

Connections of LTI models

Note: use **minreal** (*minimal realization* of an LTI model) to simplify common factors between the numerator and denominator:

```
sysGs = minreal(sysGs)
```

```
sysGs =

     1
  -------
  s (s+2)

Continuous-time zero/pole/gain model.
```



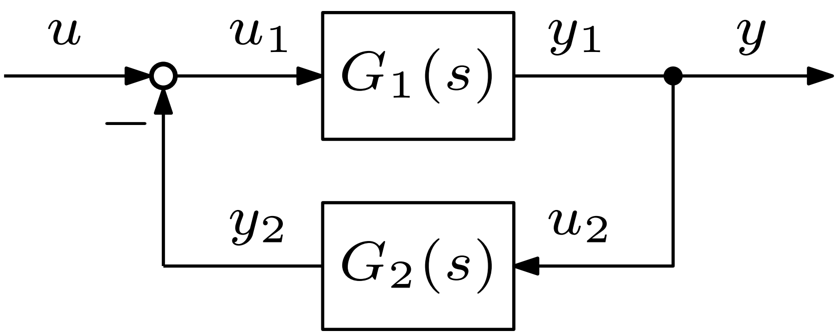
$$G_s(s) = G_1(s)\,G_2(s) = \frac{\cancel{s+1}}{s\cancel{(s+1)}(s+2)} = \frac{1}{s(s+2)}$$

Connections of LTI models

- Feedback connection:

```
% sysG1 in feedforward path
% sysG2 in feedback path
% (default: negative feedback)
sysGf1 = feedback(sysG1, sysG2)
```

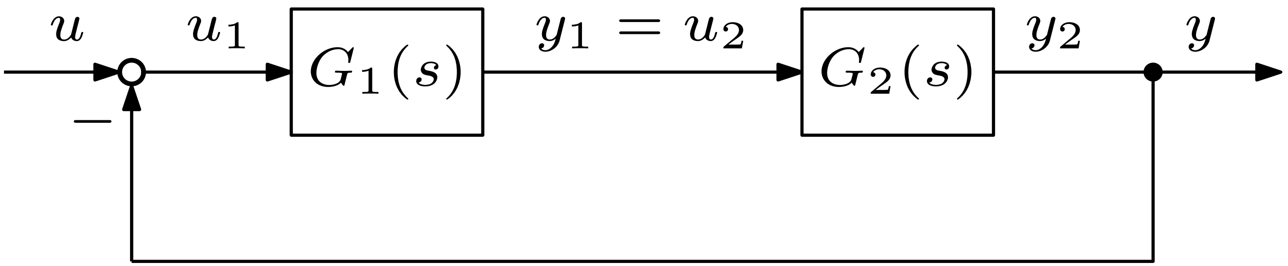
```
sysGf1 =

   (s+2)
  -------
  (s+1)^3

Continuous-time zero/pole/gain model.
```



$$G_{f1}(s) = \frac{\dfrac{1}{s(s+1)}}{1 + \dfrac{1}{s(s+2)}} = \frac{s+2}{(s+1)^3}$$

Connections of LTI models

- Feedback connection:




```
% sysG1*sysG2 in feedforward path
% unitary feedback
% (default: negative feedback)
sysGf2 = feedback(sysG1*sysG2, 1)
```

```
sysGf2 =

   (s+1)
  -------
  (s+1)^3

Continuous-time zero/pole/gain model.
```

```
sysGf2 = minreal(sysGf2, 1e-5)
```

```
sysGf2 =

         1
  ---------------
  (s^2 + 2s + 1)

Continuous-time zero/pole/gain model.
```

$$G_{f2}(s) = \frac{\frac{1}{s(s+2)}}{1 + \frac{1}{s(s+2)}} = \frac{1}{(s+1)^2}$$

Connections of LTI objects

`Parallel`, **`series`** and **`feedback`** routines also accept LTI models in *non-encapsulated* form:

```
%  get TF data from TF objects
[numG1, denG1] = tfdata(sysG1, 'v');
[numG2, denG2] = tfdata(sysG2, 'v');

%  evaluate interconnections of LTI models in non-encapsulated form
[numGp, denGp] = parallel(numG1, denG1, numG2, denG2);
[numGs, denGs] = series(numG1, denG1, numG2, denG2);
[numGf1, denGf1] = feedback(numG1, denG1, numG2, denG2);

%  get SS data from SS objects
[A1, B1, C1, D1] = ssdata(sysG1);
[A2, B2, C2, D2] = ssdata(sysG2);

%  evaluate interconnections of LTI models in non-encapsulated form
[Ap, Bp, Cp, Dp] = parallel(A1, B1, C1, D1, A2, B2, C2, D2);
[As, Bs, Cs, Ds] = series(A1, B1, C1, D1, A2, B2, C2, D2);
[Af1, Bf1, Cf1, Df1] = feedback(A1, B1, C1, D1, A2, B2, C2, D2);
```