Hands On: Introduction to MATLAB - Part 5 - LTI Systems

This is the fifth MATLAB live script of the collection ***Hands On: Using MATLAB in the 267MI "System Dynamics" course***, devoted to introduce the MATLAB/Simulink environment and tools for solving practical problems related to the topics of the 267MI course, i.e. performance analysis of dynamic systems, parametric estimation, identification of models from data, and prediction of the evolution of dynamic systems.

Use this link to go back to the main live script of the collection.

Table of Contents

Objectives

The aim of this module is to understand how to create and manipulate **LTI systems** in MATLAB.

Time Invariant Linear Dynamic Systems

AN LTI system can be described

- using a state-space description
- using the transfer function matrix for a MIMO system or a single transfer function for a SISO system

In MATLAB, it is possible to define a linear time-invariant dynamical system as an object of type `LTI model` from any of these descriptions (use `help ltimodels` for details]). Using these MATLAB objects, it is possible to analyze the properties of the corresponding dynamic system, and also to simulate the evolution of the dynamic system over time with assigned initial conditions and inputs.

From now on, for simplicity, we only consider SISO systems.

LTI-System Object from the State-Space Description

Starting from the state equation, you can

- define the matrices A, B, C, D of the state equations in the MATLAB workspace;
- define the LTI system by means of the command `ss`

```
help ss
```

```
 ss  State-space models.
 
    Construction:
      SYS = ss(A,B,C,D) creates an object SYS representing the continuous-
      time state-space model
           dx/dt = Ax(t) + Bu(t)
            y(t) = Cx(t) + Du(t)
      You can set D=0 to mean the zero matrix of appropriate size. SYS is
      of type ss when A,B,C,D are dense numeric arrays, of type GENSS when
      A,B,C,D depend on tunable parameters (see REALP and GENMAT), and
      of type USS when A,B,C,D are uncertain matrices (requires Robust
      Control Toolbox). Use SPARSS when A,B,C,D are sparse matrices.
 
      SYS = ss(A,B,C,D,Ts) creates a discrete-time state-space model with
      sample time Ts (set Ts=-1 if the sample time is undetermined).
 
      SYS = ss(D) specifies a static gain matrix D.
 
      You can set additional model properties by using name/value pairs.
      For example,
         sys = ss(-1,2,1,0,'InputDelay',0.7,'StateName','position')
      also sets the input delay and the state name. Type "properties(ss)"
      for a complete list of model properties, and type
         help ss.<PropertyName>
      for help on a particular property. For example, "help ss.StateName"
      provides information about the "StateName" property.
 
    Arrays of state-space models:
      You can create arrays of state-space models by using ND arrays for
      A,B,C,D. The first two dimensions of A,B,C,D define the number of
      states, inputs, and outputs, while the remaining dimensions specify
      the array sizes. For example,
         sys = ss(rand(2,2,3,4),[2;1],[1 1],0)
      creates a 3x4 array of SISO state-space models. You can also use
      indexed assignment and STACK to build ss arrays:
         sys = ss(zeros(1,1,2))     % create 2x1 array of SISO models
         sys(:,:,1) = rss(2)        % assign 1st model
         sys(:,:,2) = ss(-1)        % assign 2nd model
         sys = stack(1,sys,rss(5))  % add 3rd model to array
 
    Conversion:
      SYS = ss(SYS) converts any dynamic system SYS to the state-space
      representation. The resulting model SYS is always of class ss.
 
      SYS = ss(SYS,'min') computes a minimal realization of SYS.
```

```
    SYS = ss(SYS,'explicit') computes an explicit realization (E=I) of SYS.
    An error is thrown if SYS is improper.

    See also dss, delayss, rss, drss, sparss, mechss, ssdata, tf, zpk, frd, genss, uss, DynamicSystem.

  Documentation for ss
  Other uses of ss
```

A few Examples

Given the following state-space description of a continuous-time dynamical system

$$\begin{cases} \dot{x}_1(t) &= -x_2(t) + 3u(t) \\ \dot{x}_2(t) &= -3x_1(t) + 2x_2(t) \\ y(t) &= 4x_1(t) + 2u(t) \end{cases}$$

the corresponding MATLAB object is given by

```matlab
A = [0 -1;-3 2]; B = [3;0]; C=[4 0]; D=2; % the system matrices
sysC = ss(A, B, C, D)
```

```
sysC =

  A =
       x1  x2
   x1   0  -1
   x2  -3   2

  B =
       u1
   x1   3
   x2   0

  C =
       x1  x2
   y1   4   0

  D =
       u1
   y1   2

Continuous-time state-space model.
```

Consider a discrete time dynamical system, described by the model

$$\begin{cases} x_1(k+1) &= -x_2(k) + 3u(k) \\ x_2(k+1) &= -3x_1(k) + 2x_2(k) \\ y(k) &= 4x_1(k) + 2u(k) \end{cases}$$

then

```matlab
A = [0 -1;-3 2]; B = [3;0]; C=[4 0]; D=2; % the system matrices
sysD = ss(A, B, C, D, -1) % the sampling time is unpsecified
```

```
sysD =

  A = 
       x1  x2
   x1   0  -1
   x2  -3   2

  B = 
       u1
   x1   3
   x2   0

  C = 
       x1  x2
   y1   4   0

  D = 
       u1
   y1   2

Sample time: unspecified
Discrete-time state-space model.
```

LTI-System Object from the Transfer Function

Given a transfer function as model for an LTI system, you can create an LTImodel object by using the `tf` command:

```
help tf
```

```
  tf  Construct transfer function or convert to transfer function.
 
    Construction:
      SYS = tf(NUM,DEN) creates a continuous-time transfer function SYS with
      numerator NUM and denominator DEN. SYS is an object of type tf when
      NUM,DEN are numeric arrays, of type GENSS when NUM,DEN depend on tunable
      parameters (see REALP and GENMAT), and of type USS when NUM,DEN are
      uncertain (requires Robust Control Toolbox).
 
      SYS = tf(NUM,DEN,TS) creates a discrete-time transfer function with
      sample time TS (set TS=-1 if the sample time is undetermined).
 
      S = tf('s') specifies the transfer function H(s) = s (Laplace variable).
      Z = tf('z',TS) specifies H(z) = z with sample time TS.
      You can then specify transfer functions directly as expressions in S
      or Z, for example,
         s = tf('s');  H = exp(-s)*(s+1)/(s^2+3*s+1)
 
      SYS = tf creates an empty tf object.
      SYS = tf(M) specifies a static gain matrix M.
 
      You can set additional model properties by using name/value pairs.
      For example,
         sys = tf(1,[1 2 5],0.1,'Variable','q','IODelay',3)
      also sets the variable and transport delay. Type "properties(tf)"
      for a complete list of model properties, and type
         help tf.<PropertyName>
      for help on a particular property. For example, "help tf.Variable"
      provides information about the "Variable" property.
```

```
    By default, transfer functions are displayed as functions of 's' or 'z'.
    Alternatively, you can use the variable 'p' in continuous time and the
    variables 'z^-1', 'q', or 'q^-1' in discrete time by modifying the
    "Variable" property.

  Data format:
    For SISO models, NUM and DEN are row vectors listing the numerator
    and denominator coefficients in descending powers of s,p,z,q or in
    ascending powers of z^-1 (DSP convention). For example,
       sys = tf([1 2],[1 0 10])
    specifies the transfer function (s+2)/(s^2+10) while
       sys = tf([1 2],[1 5 10],0.1,'Variable','z^-1')
    specifies (1 + 2 z^-1)/(1 + 5 z^-1 + 10 z^-2).

    For MIMO models with NY outputs and NU inputs, NUM and DEN are
    NY-by-NU cell arrays of row vectors where NUM{i,j} and DEN{i,j}
    specify the transfer function from input j to output i. For example,
       H = tf( {-5 ; [1 -5 6]} , {[1 -1] ; [1 1 0]})
    specifies the two-output, one-input transfer function
       [     -5 /(s-1)      ]
       [ (s^2-5s+6)/(s^2+s) ]

  Arrays of transfer functions:
    You can create arrays of transfer functions by using ND cell arrays
    for NUM and DEN above. For example, if NUM and DEN are cell arrays
    of size [NY NU 3 4], then
       SYS = tf(NUM,DEN)
    creates the 3-by-4 array of transfer functions
       SYS(:,:,k,m) = tf(NUM(:,:,k,m),DEN(:,:,k,m)),  k=1:3,  m=1:4.
    Each of these transfer functions has NY outputs and NU inputs.

    To pre-allocate an array of zero transfer functions with NY outputs
    and NU inputs, use the syntax
       SYS = tf(ZEROS([NY NU k1 k2...])) .

  Conversion:
    SYS = tf(SYS) converts any dynamic system SYS to the transfer function
    representation. The resulting SYS is always of class tf.

  See also tf/exp, filt, tfdata, zpk, ss, frd, genss, uss, DynamicSystem.

   Documentation for tf
   Other uses of tf
```

A few Examples

Consider the LTI systems

$$G_1(s) = \frac{s+1}{s^2+3s+16} \qquad\qquad G_2(z) = \frac{(z+1.0)\,(z+0.4)}{(z-1)\,(z+0.8)}$$

then

```
numG1 = [ 1 1 ]; denG1 = [ 1 3 16 ];
G1 = tf(numG1, denG1)
```

```
G1 =

      s + 1
  --------------
  s^2 + 3 s + 16

Continuous-time transfer function.
```

```
numG2 = conv([1 1],[1 0.4]); denG2 = conv([1  -1],[1  0.8]);
G2 = tf(numG2, denG2, -1)
```

```
G2 =

  z^2 + 1.4 z + 0.4
  -----------------
  z^2 - 0.2 z - 0.8

Sample time: unspecified
Discrete-time transfer function.
```

Alternative Way of Using the tf Command

Continuous-time and discrete-time "elementary" transfer functions can be defined, and used as "building blocks" in writing the expressions of the actual transfer functions.

Consider

$$G_1(s) = e^{-s}\frac{s+1}{s^2+3s+16}$$

```
s=tf('s');
Gs = exp(-2*s) * (s+1)/(s^2+3*s+2)
```

```
Gs =

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

Continuous-time transfer function.
```

In the discrete time case

$$H(z) = z^{-3}\frac{(z+1.0)\,(z+0.4)}{(z-1)\,(z+0.8)}$$

```
z = tf('z',-1);
Hz = z^(-3)*(z+1)*(z+0.4)/(z-1)/(z+0.8)
```

```
Hz =

     z^2 + 1.4 z + 0.4
  -----------------------
```

```
   z^5 - 0.2 z^4 - 0.8 z^3

 Sample time: unspecified
 Discrete-time transfer function.
```

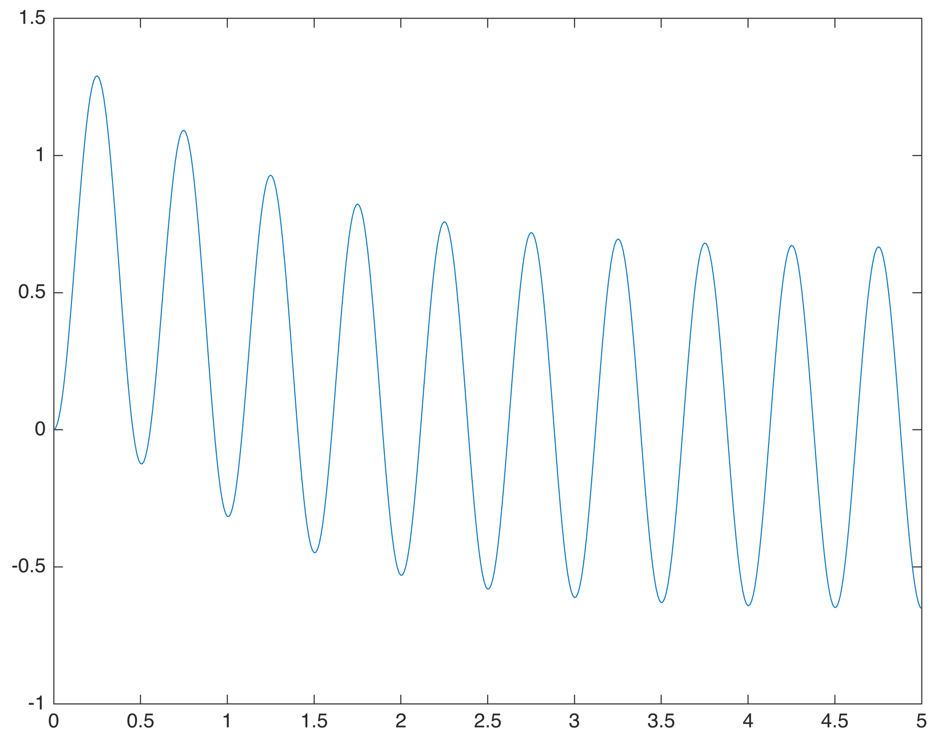
Simulation of LTI Systems

The following MATLAB commands are available, both for continuos-time and discrete-time LTI system:

- `impulse`: simulation of impulse response;
- `step`: simulation of the step response;
- `initial`: simulation of the free state and output movements;
- `lsim`: simulation of the forced movements.

First Example: Forced Output Movement of a Continuous-Time System

```
a=[-1 ,0;3,-4];
b=[2;1];
c=[1,2];d=0; % the system matrices
sysC = ss(a,b,c,d); % the LTI system object

t=(0:0.01:5); % a vector contining some time instants
u=2*sin(2*pi*2*t); % the corresponding input values

y=lsim(sysC,u,t); % evaluate the forced output movement

plot(t,y); % plotting the results
```



An Exercise

Write a piece of code for the evaluation of the free state movement of the system, starting from a random initial state.

Hint: have a look to

```
help initial
```

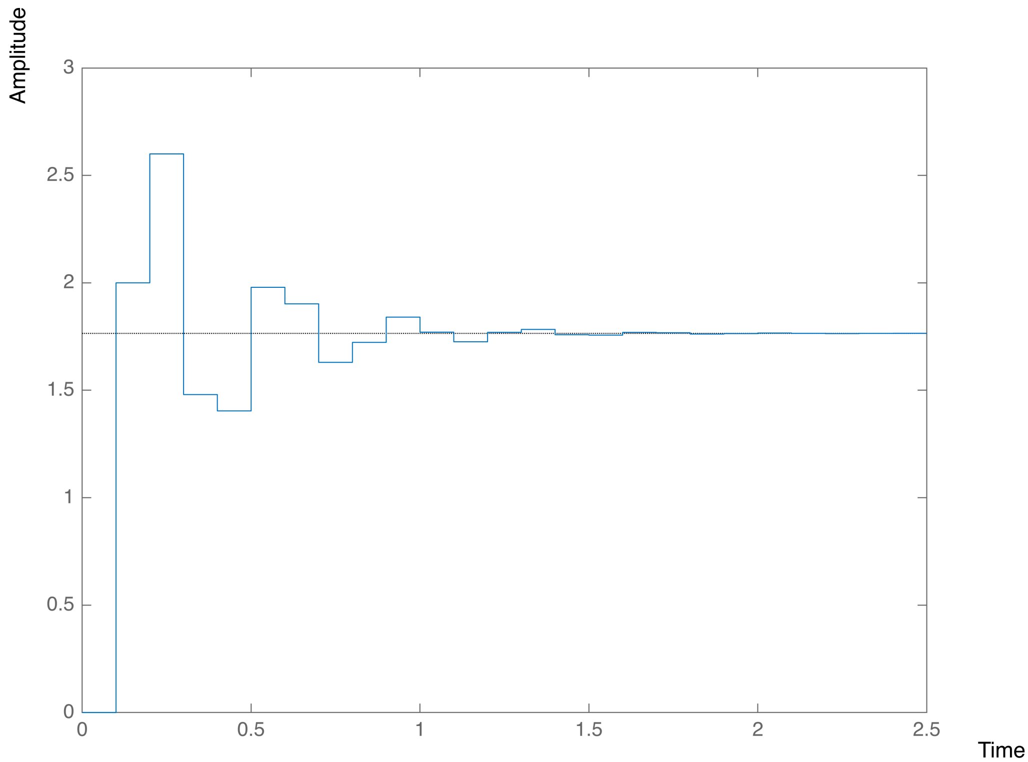
```
 initial  Initial condition response of state-space models.

    initial(SYS,X0) plots the undriven response of the state-space model SYS
    (created with SS) with initial condition X0 on the states. This response
    is characterized by the equations
                                .
      Continuous time:    x = A x ,  y = C x ,  x(0) = x0

      Discrete time:  x[k+1] = A x[k],  y[k] = C x[k],  x[0] = x0 .

    The time range and number of points are chosen automatically.

    initial(SYS,X0,TFINAL) simulates the time response from t=0 to the final
    time t=TFINAL (expressed in the time units specified in SYS.TimeUnit).
    For discrete-time models with unspecified sample time, TFINAL is
    interpreted as the number of sampling periods.

    initial(SYS,X0,T) uses the time vector T for simulation (expressed in the
    time units of SYS). For discrete-time models, T should be of the form
    Ti:Ts:Tf where Ts is the sample time. For continuous-time models, T
    should be of the form Ti:dt:Tf where dt is the sampling period for the
    discrete approximation of SYS.
```

```
initial(SYS1,SYS2,...,X0,T) plots the response of several systems
SYS1,SYS2,... on a single plot. The time vector T is optional. You can
also specify a color, line style, and marker for each system, for
example:
   initial(sys1,'r',sys2,'y--',sys3,'gx',x0).

When invoked with left hand arguments,
   [Y,T,X] = initial(SYS,X0)
returns the output response Y, the time vector T used for simulation,
and the state trajectories X. No plot is drawn on the screen. The
matrix Y has LENGTH(T) rows and as many columns as outputs in SYS.
Similarly, X has LENGTH(T) rows and as many columns as states. The
time vector T is expressed in the time units of SYS.

See also initialplot, impulse, step, lsim, ltiview, DynamicSystem.

Documentation for initial
Other uses of initial
```

```
% insert here the code
```

Another Example: Step Response of a Sampled-Time LTI System

```
% sampled time LTI system
Ts = 0.1; % the sampling period
sysD = tf([2 1],[1 0.2 0.5], Ts);

figure;step(sysD); % the step response
```



to be continued ...

Summary

Using this live script you have:

- learnt how to describe an LTI system in MATLAB;
- ...

Back to the Index

Use this link to go back to the main live script of the collection.

Back to the Previous Part: Functions

Use this link to go back to the previous live script of this collection.