

System ID Example

1 System to be Identified

1.1 Introduction

This document is intended to give you an example of using MATLAB to work with equations (8.9)–(8.13) in the notes to identify a discrete-time LTI dynamic system. I'll do an example similar to the one I did in class. Everything will be done from the “command window” although it could be a single script file.

1.2 Third-Order System

Consider an underdamped second order system with an added first-order mode. The second order portion will have natural frequency f_n and damping ratio ζ ; the first-order mode will have time constant τ . The transfer function for a unity-gain system of this type is

$$G(s) = \frac{Y(s)}{U(s)} = \frac{\omega_n^2}{(\tau s + 1)(s^2 + 2\zeta\omega_n s + \omega_n^2)} = \frac{\omega_n^2/\tau}{(s + 1/\tau)(s^2 + 2\zeta\omega_n s + \omega_n^2)} \quad (1)$$

1.2.1 Numerical Values

The following numerical values will be used:

$$\begin{aligned} f_n &= 10 \text{ Hz} \\ \zeta &= 0.3 \\ \tau &= 0.01 \text{ sec} \end{aligned}$$

The first-order mode will have an effect on the system response, but the second-order model should be dominant.

1.2.2 MATLAB Representation

The continuous $G(s)$ can be entered into MATLAB in the following manner:

```
>> fn = 10;           % Natural frequency in Hz
>> zeta = 0.3;       % Damping ratio
>> tau = 0.01;      % Time constant
>> wn = 2*pi*fn;    % Natural frequency in rad/s

>> num = [0 0 0 wn^2/tau]; % Define numerator polynomial
>> den = conv([1 1/tau],[1 2*zeta*wn wn^2]); % Define denominator polynomial using conv
>> Gc = tf(num,den)
```

Transfer function:

3.948e05

s^3 + 137.7 s^2 + 7718 s + 3.948e05

Note that I used the conv convolution function to multiply the two polynomials in the denominator of (1). This saves a little manual algebra.

1.2.3 Step Response of Continuous Systems

The step response of the $G(s)$ is shown in Figure 1, along with the step response if the first-order mode were not present. We see that the first-order mode does have an effect, but the second-order response nature dominates.

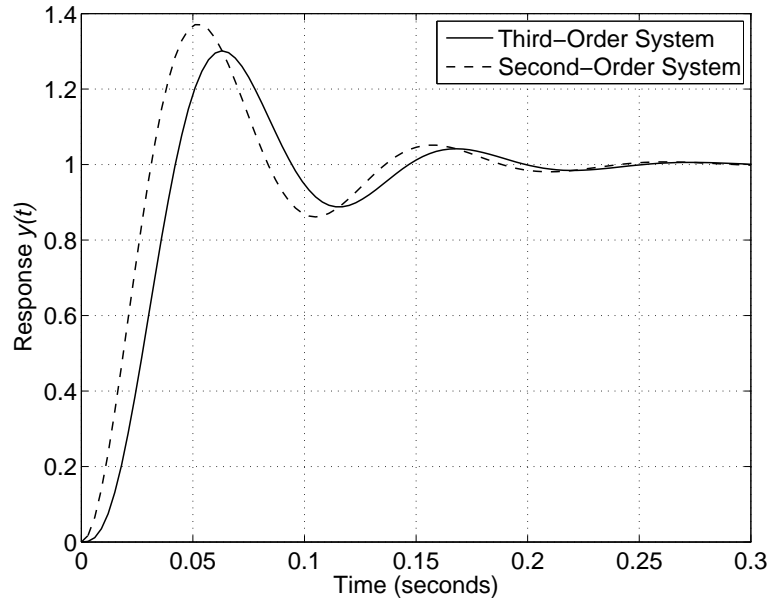


Figure 1: Step response of the continuous third-order and second-order systems. The additional first-order mode affects the response, but not drastically.

2 Experimental Data and Model Fit

With $G(s)$ in MATLAB, we can conduct the “experimental” portion of this exercise. This would be actual laboratory measurements of the system output $y(t)$ when being driven with either a random or binary (PRBS) input $u(t)$. In this example we drive the model with the given input to get the “experimental” output.

2.1 Discretization of $G(s)$

To represent the discrete system we need to discretize the $G(s)$ of (1). What sample frequency? I’ll select a sampling frequency of $f_s = 100$ Hz, which means $T = 1/f_s = 0.01$ sec.

Using the ‘zoh’ method in `c2d`, we get the following:

```
>> fs = 100; % Sampling frequency in Hz
>> T = 1/fs; % Sampling period in sec
>> c2d(Gc,T)
```

Transfer function:

```
0.07559 z^2 + 0.1761 z + 0.02345
-----
z^3 - 1.503 z^2 + 0.871 z - 0.09283
```

Sampling time: 0.01

So the discrete transfer function $G(z)$ of our system is

$$G(z) = \frac{Y(z)}{U(z)} = \frac{0.07559z^2 + 0.1761z + 0.02345}{z^3 - 1.503z^2 + 0.871z - 0.09283} = \frac{0.07559z^{-1} + 0.1761z^{-2} + 0.02345z^{-3}}{1 - 1.503z^{-1} + 0.871z^{-2} - 0.09283z^{-3}} \quad (2)$$

We are not supposed to know the transfer function of (2)!!

2.1.1 Poles and Zeros

It might be interesting to see the location of the poles and zeros of $G(z)$. Two MATLAB functions help with this: `zpk` and `damp`. Converting the transfer function to zero-pole-K form yields

```
>> zpk(Gd)
```

```
Zero/pole/gain:
  0.075592 (z+2.188) (z+0.1418)
-----
(z-0.1353) (z^2 - 1.368z + 0.6859)
```

which shows the two zeros and the real pole, but not the complex poles. If we using the `damp` function we get

```
>> damp(Gd)
```

Eigenvalue	Magnitude	Equiv. Damping	Equiv. Freq. (rad/s)
6.84e-01 + 4.67e-01i	8.28e-01	3.00e-01	6.28e+01
6.84e-01 - 4.67e-01i	8.28e-01	3.00e-01	6.28e+01
1.35e-01	1.35e-01	1.00e+00	2.00e+02

which shows the complex poles are at $z = 0.684 \pm j0.467$, with an equivalent damping ratio of 0.3 and natural frequency of 62.8 rad/s. Again, we're not supposed to know this! But this is an *educational* example.

2.2 Experimental Data

2.2.1 Random Inputs

Two 201-sample inputs will be used: a random signal with a uniform distribution $u_r(t)$ and a random binary sequence $u_b(t)$. I'm going to assume that each input should have a range of ± 5 . Since the MATLAB `rand` function produces numbers in the range 0..1 I will bias those down by 0.5, then scale up to a magnitude of 5. The binary input will be in the range ± 1 so it will be scaled up by 5.

```
>> ur = (rand(201,1)-0.5)*10; % 201 random samples, biased and scaled to be +/- 5
>> ub = sign(randn(201,1))*5; % 201 binary samples, scaled to be +/- 5
```

2.2.2 System Response

To use `lsim` we need a time vector corresponding to the input.

```
>> k = [0:200]'; % Column vector from 0..200 (I prefer column to row vectors)
>> t = k*T; % Corresponding time samples
```

Next use `lsim` to get the output. **This would done on real hardware in practice!**

```
>> yr = lsim(Gd,ur,t); % Random input
>> yb = lsim(Gd,ub,t); % Binary input
```

Plots of both I/O records are shown in Figure 2 below. I used the MATLAB 'stairs' function to plot the binary input so the "steps" are properly formatted.

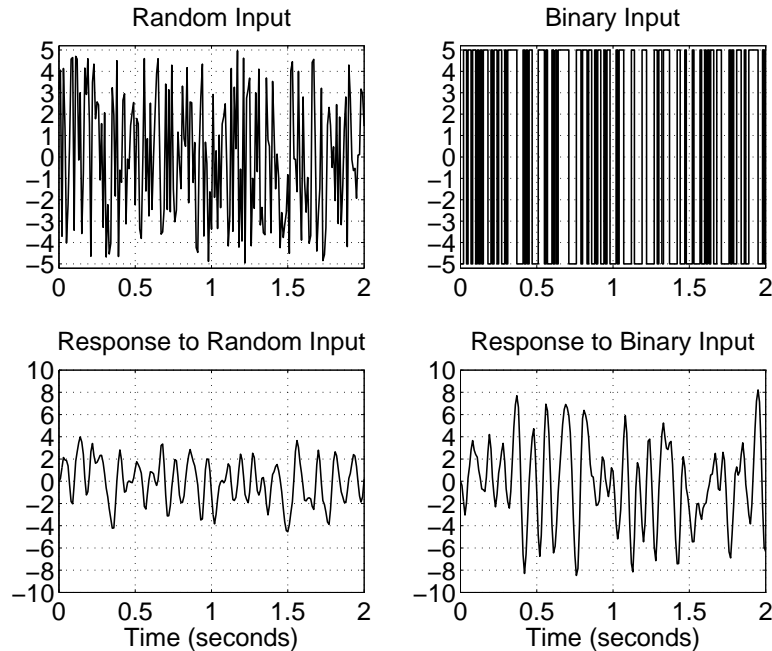


Figure 2: Input/Output data records for the system. It appears that the binary input "exercises" the system a little more.

2.3 Model Fitting

I think it is *very helpful* to write a MATLAB function to do the least-square fitting. I did that with my own function,

```
>> help idlsq
G = IDLSQ(u,y,n,T) does a least-square model fit using a nth order model
and N input/output datapoints. Vectors u and y are column vectors of
input and output, respectively. Scalar n is the desired model order,
while T is the sample period in seconds. The returned parameter G is an
LTI transfer function model.
```

There are only *12 lines* of code in that function. However, I won't give it to you...

2.3.1 Data Formatting

The data should be put in the form of equations (8.9) and (8.10) in the notes. Below is what I did to you samples from 10 to 201 in the fit (early samples omitted because of "start-up" transient):

```
>> Psi = [y(12:200) y(11:199) y(10:198) u(12:200) u(11:199) u(10:198)]; % 3rd order model
>> size(Psi)
```

```
ans = 189      6
```

```
>> Y = y(13:201);
>> size(Y)
```

```
ans = 189      1
```

In the above case, variables 'u' and 'u' are either for the random or binary runs.

2.3.2 Least-Square Fits

Third order model. Finally, use (8.13) to perform the fit, like this:

```
>> theta = inv(Psi'*Psi)*Psi'*Y
```

```
theta = 1.5030
        -0.8710
         0.0928
         0.0756
         0.1761
         0.0235
```

These are the b_i and a_i , respectively. Comparison with (2) shows a perfect fit.

Reduced-order model. What about trying a 2nd order model? Arrange the data in the same fashion as before. To save time I used my 'idlsq' function, which (using the binary data) gave

```
>> idlsq(ub(10:200),yb(10:200),2,T)
```

```
Transfer function:
  0.07675 z + 0.1798
-----
 z^2 - 1.455 z + 0.7507
```

The step responses of the original $G(s)$ and this “reduced-order” $G(z)$ are shown in Figure 3 below. You may think there’s substantial error, but I would argue that the reduced-order model captures most of the “dynamic” behavior, and would be adequate for controller design.

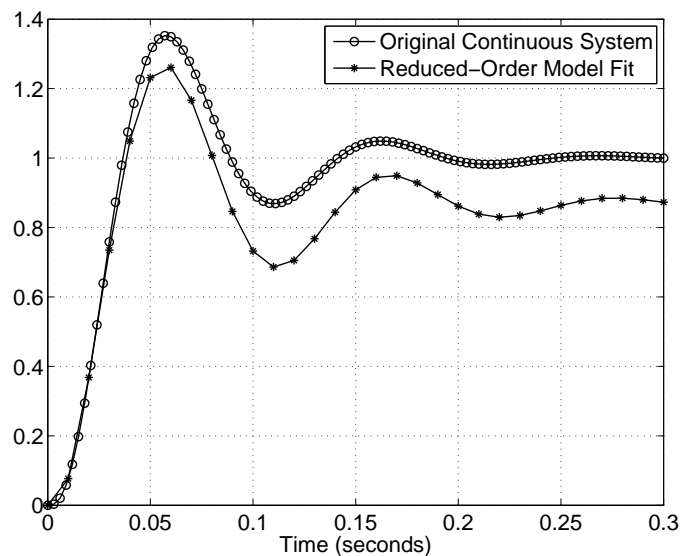


Figure 3: Step response of original continuous $G(s)$ and reduced-order model $G(z)$.

3 Conclusion

I neglected a number of topics in this example—quantization noise, comparison of the error (residual), *etc.* By the way, if you want to find the error ϵ , simply take the same input you used on the real system and drive your model with it.

Call the actual output y , and the output of your model \hat{y} ; take the difference and square it, and you've got the error (squared), that is,

$$\epsilon = [y - \hat{y}]^T [y - \hat{y}] \quad (3)$$

If you fit several model orders and calculate ϵ for each, you can see at which model order your error begins to increase, indicating a worse fit.