A picture containing fireworks, dark, water, flying

Description automatically generated

College of Engineering

School of Aeronautics and Astronautics

AAE 564

System Analysis and Synthesis

Homework 7

Input/Output Responses of LTI Systems

*Author:*

Tomoki Koike

*Supervisor:*

Martin Corless

October 16th, 2020 Friday

Purdue University

West Lafayette, Indiana

Exercise 1

Consider the differential equation

where A is a square matrix. Show that if has as an eigenvalue, then there is a nonzero initial state such that the equation has a solution which satisfies .

If the given equation has an eigenvalue of

the complex conjugate is also an eigenvalue of the matrix . Thus, we know that

From this we can deduce the solution of this equation to be

For

**Exercise 2**

Compute at for

using all four methods mentioned in the notes.

Method 1: MATLAB “expm()”

% Problem 2

% Method 1 expm()

A = [0, 1; 1, 0];

t = log(2);

eA1 = expm(A\*t);

The result is

|  |
| --- |
| eA1 = 2×2  1.2500 0.7500  0.7500 1.2500 |

Method 2: Numerical Simulation

Using ode45 we accomplish this. Following is the function

function dx = numSim(t, x, A)

dx = A\*x;

end

and the following is the MATLAB script

tspan = 0:0.1:10;

A = [0, 1; 1, 0];

x0 = [1, 0];

res = ode45(@(t,x) numSim(t,x,A), tspan, x0)

E1 = deval(res, log(2))

x0 = [0, 1];

res = ode45(@(t,x) numSim(t,x,A), tspan, x0)

E2 = deval(res, log(2))

This gives us the following result

|  |  |
| --- | --- |
| E1 = 2×1  1.2500  0.7500 | E2 = 2×1  0.7500  1.2500 |

This agrees with the answer of method 1.

Method 3: Jordan form

The eigenvalue of this matrix is

The eigenvectors become,

The transition matrix becomes

In MATLAB, if

% Method 2 numerical simulation

[v, d] = eig(A)

T = [-1, 1; 1, 1];

Phi = T\*[exp(d(1,1)\*t), 0; 0, exp(d(2,2)\*t)]/T

|  |
| --- |
| Phi = 2×2  1.2500 0.7500  0.7500 1.2500 |

Which is equal to .

Method 4: Laplace style

We already know the eigenvalues and eigenvectors, so we get the following relationship

Since,

Plug in ,

Exercise 3

Compute for matrix

1. Using the eigenvalues and eigenvectors of A;
2. Using the Laplace Transform

(a).

The eigenvalue of this matrix is

The eigenvectors become,

The transition matrix becomes

(b).

Since we already know the eigenvalues and eigenvectors

Since,

the answer becomes

Exercise 4

Compute at for

Using MATLAB, we can compute the results

A = [0, 1; -1, 0];

t = log(2);

eA = expm(A\*t)

|  |
| --- |
| eA = 2×2  0.7692 0.6390  -0.6390 0.7692 |

**Exercise 5**

Obtain (by hand) the state response of each of the following systems due to a unit impulse input and the zero initial conditions. For each case, determine whether the response contains all the system modes.

1. .

(b).

(c).

The matrix for all three (a)~(c) are the same,

But the matrix is different. If the input is a unit impulse input , and we know that function of becomes

Since we assume a zero initial condition

We know can compute .

The eigenvalue of this matrix is

The eigenvectors become,

And,

The transition matrix becomes

(a).

For this given system

Hence,

(b).

For this given system

Hence,

(c).

For this given system

Hence,

We have a mode of continuously growing do not have the other two modes. So we cannot observe all modes with these responses.

**Exercise 6**

Consider the system with input output and state variables described by

Obtain (by hand) an expression for the impulse response of this system. Does it contain all the state space modes?

The matrix is

The state space

for this system is represented by the matrices

If the input is a unit impulse input , we know that the output response is expressed as

Assuming a zero initial condition for input response

We know can compute .

The eigenvalue of this matrix is

The eigenvectors become,

And,

And,

And,

The transition matrix becomes

Thus,

And, the output response is characterized as

Since the input is an impulse input,

We have only a continuously decaying but do not have the other two. So we cannot observe all modes with these responses.

**Exercise 7**

Consider an LTI system described by

Is there a persistent input (does not go to zero) u for which the corresponding output always going to zero regardless of initial conditions? If answer is yes provided an example.

The matrix is

The state space

for this system is represented by the matrices

We can compute the eigenvalues and eigenvectors as well as using MATLAB

Consider the term

When , this goes to 0. Thus, A is stable.

The steady state part of the output response is defined as

If is 0 of , then the steady state response is also 0.

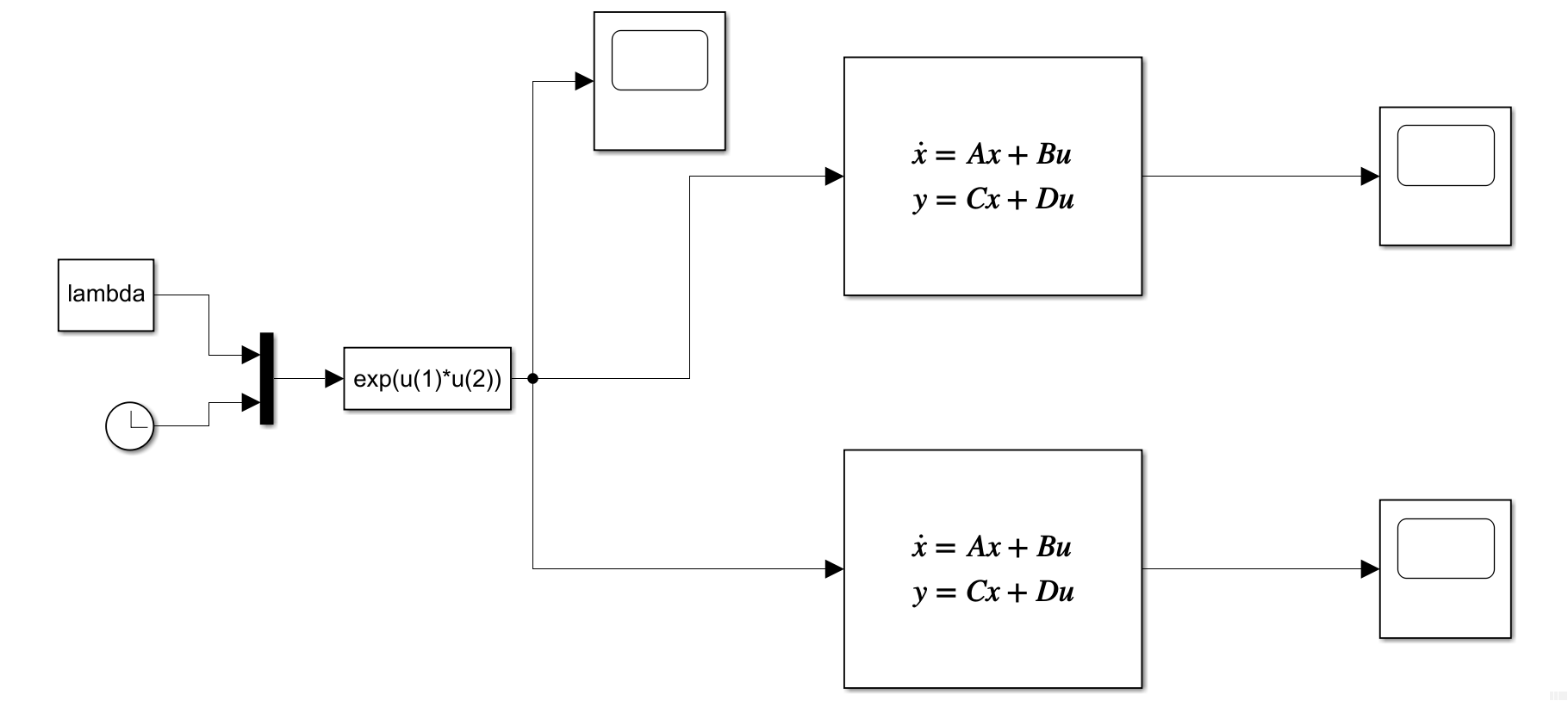
Next we find,

This goes to zero when . Thus,

If the input of this system is

With this input the steady state part of the output response becomes 0

We verify this with Simulink



% Matrices

A = [0, 1; -2, -3];

B = [0; 1];

C = [3, -1];

D = 0;

lambda = 3;

% random numbers between 0 and 20

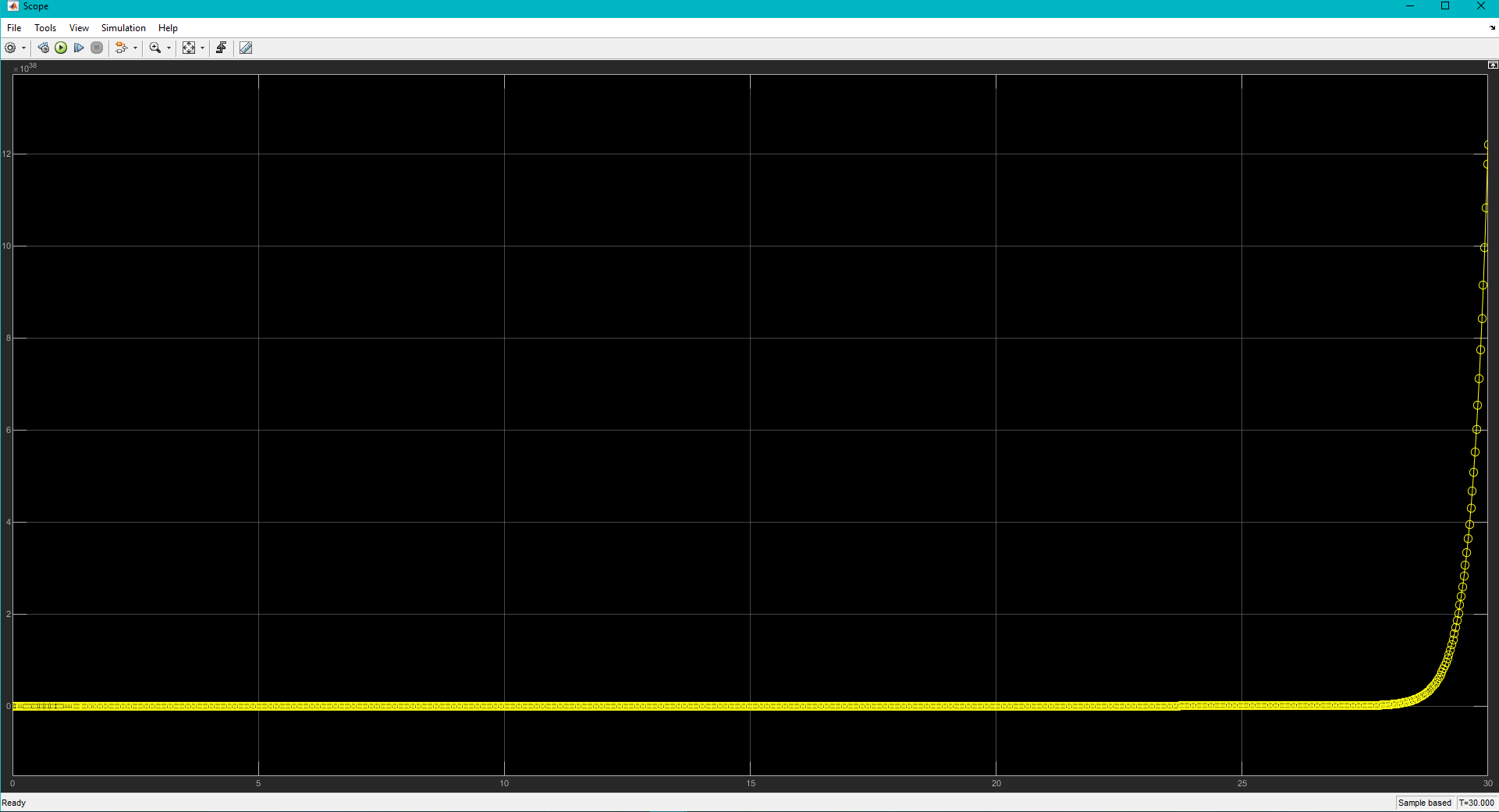
low = -20; high = 20;

x01 = low + (high - low) \* rand(2,1)

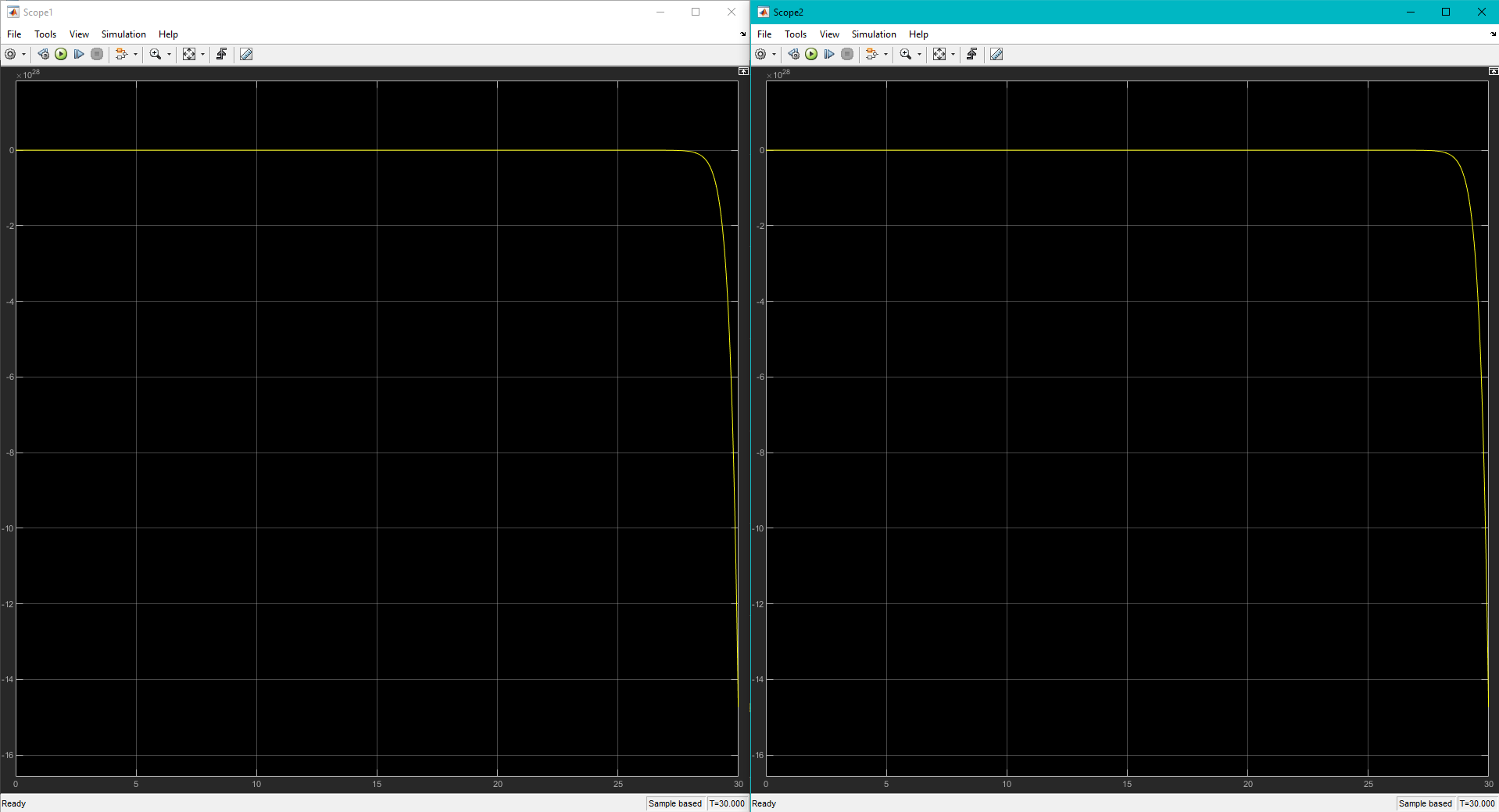
x02 = low + (high - low) \* rand(2,1)

sim('ex7\_v2');

This is the exponential input



The responses were (for 2 random initial conditions)



**Exercise 8**

Consider an LTI system described by

Is there a persistent input (does not go to zero) u for which the corresponding output always going to zero regardless of initial conditions? If answer is yes provided an example.

The matrix is

The state space

for this system is represented by the matrices

We can compute the eigenvalues and eigenvectors as well as using MATLAB

Consider the term

When , this goes to 0. Thus, A is stable.

The steady state part of the output response is defined as

If is 0 of , then the steady state response is also 0.

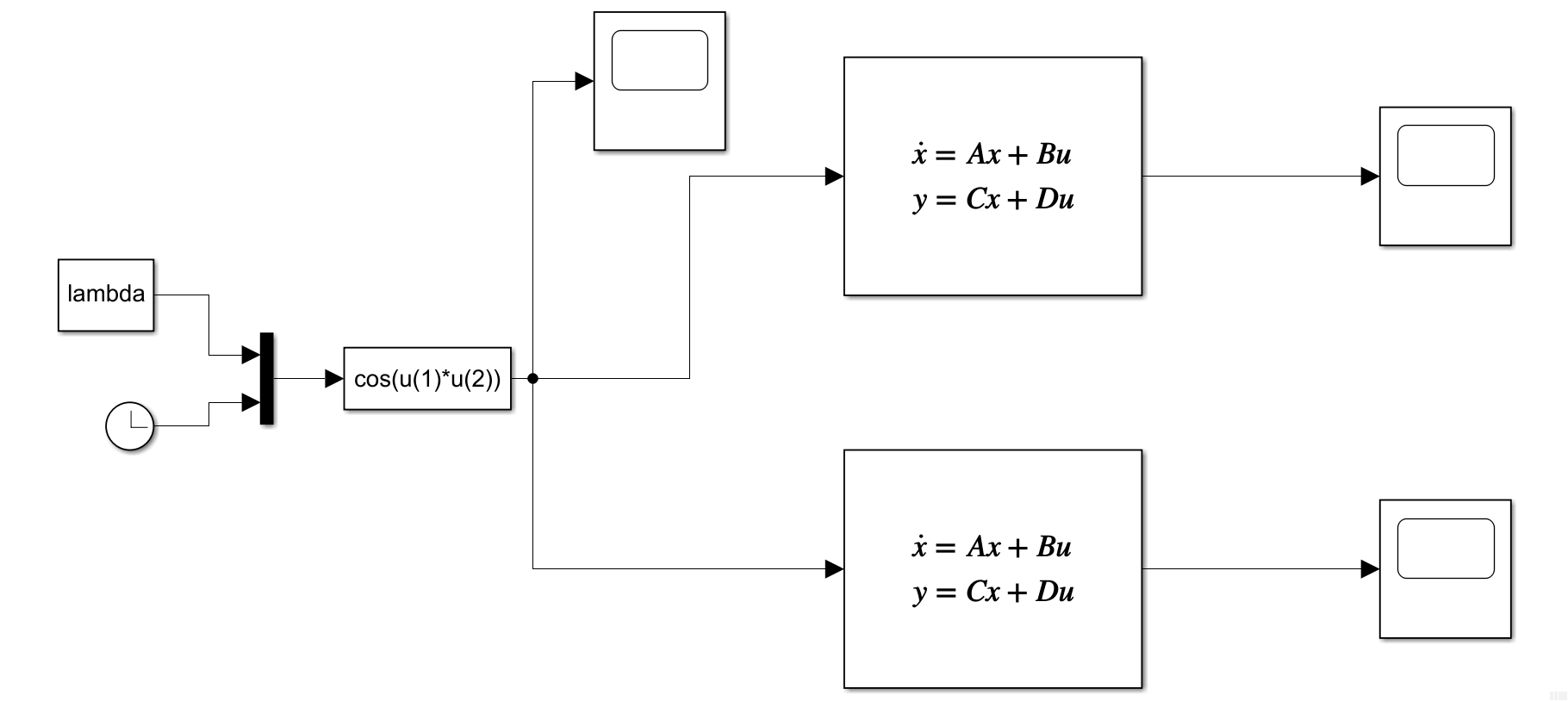
Next we find,

This goes to zero when . Thus,

If the input of this system is

With this input the steady state part of the output response becomes 0

Verify this with Simulink



% Matrices

A = [0, 1; -2, -3];

B = [0; 1];

C = [-1, -3];

D = 1;

lambda = 1;

% random numbers between -20 and 20

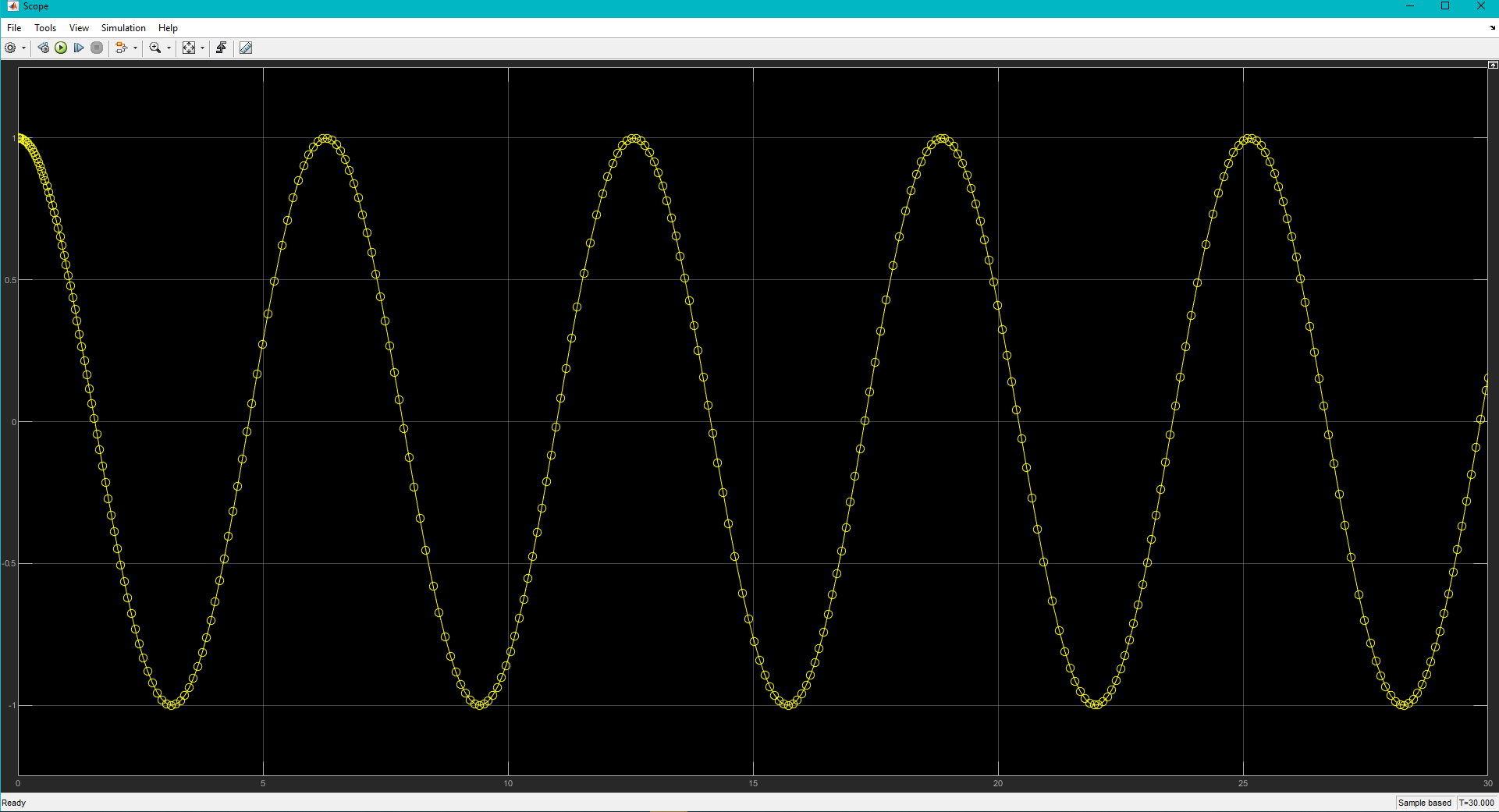
low = -20; high = 20;

x01 = low + (high - low) \* rand(2,1)

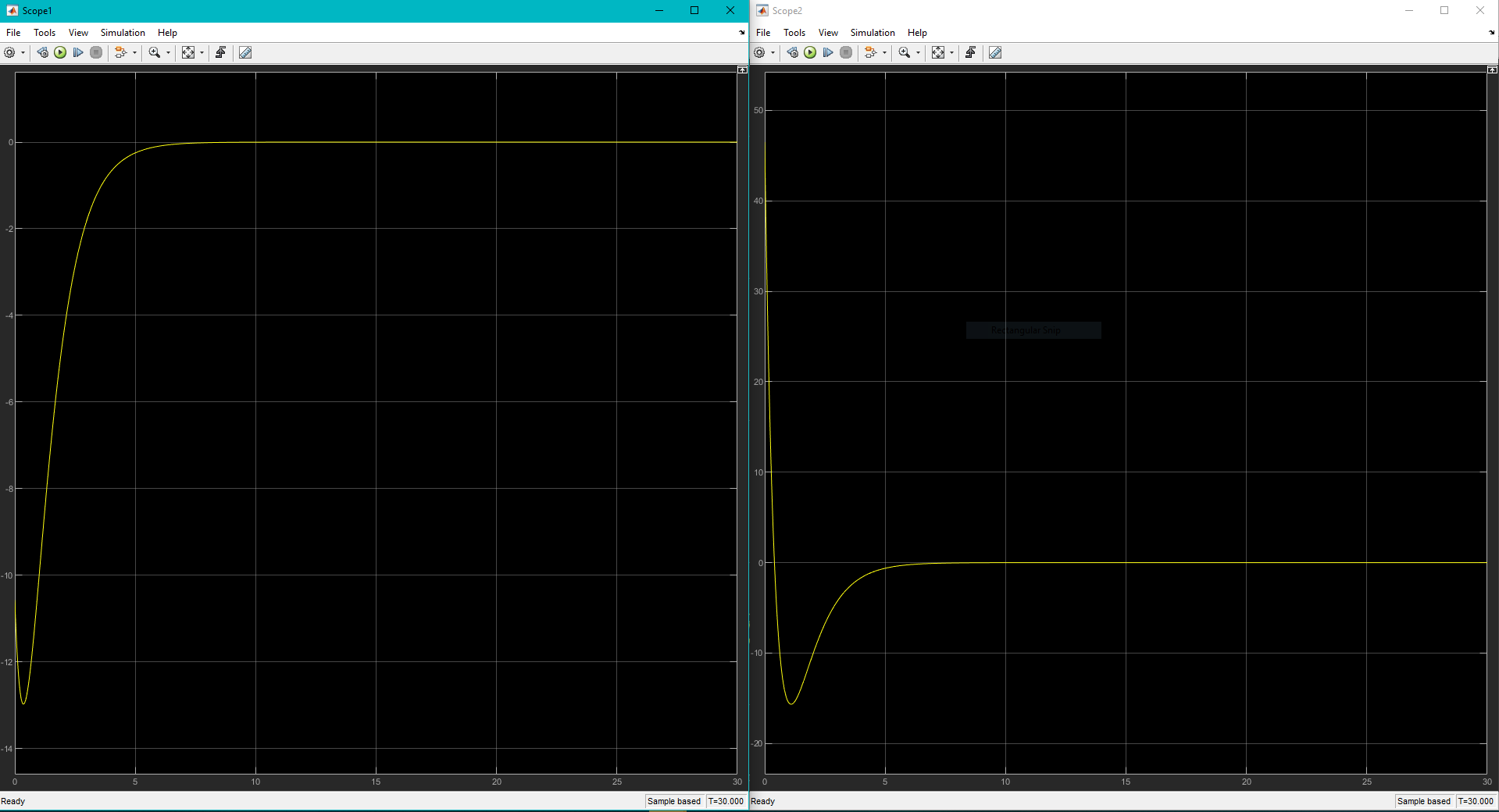
x02 = low + (high - low) \* rand(2,1)

sim('ex7\_v2');

This is the cosine input



The responses were this (for 2 random inputs)



**Exercise 9**

Disturbing the cart. Consider the pendulum cart system with parameter set P4.

We will subject it to passive stabilization and a disturbance input w, that is, we let

where and . Regard the resulting system as an input-output system with input and output and answer the following questions.

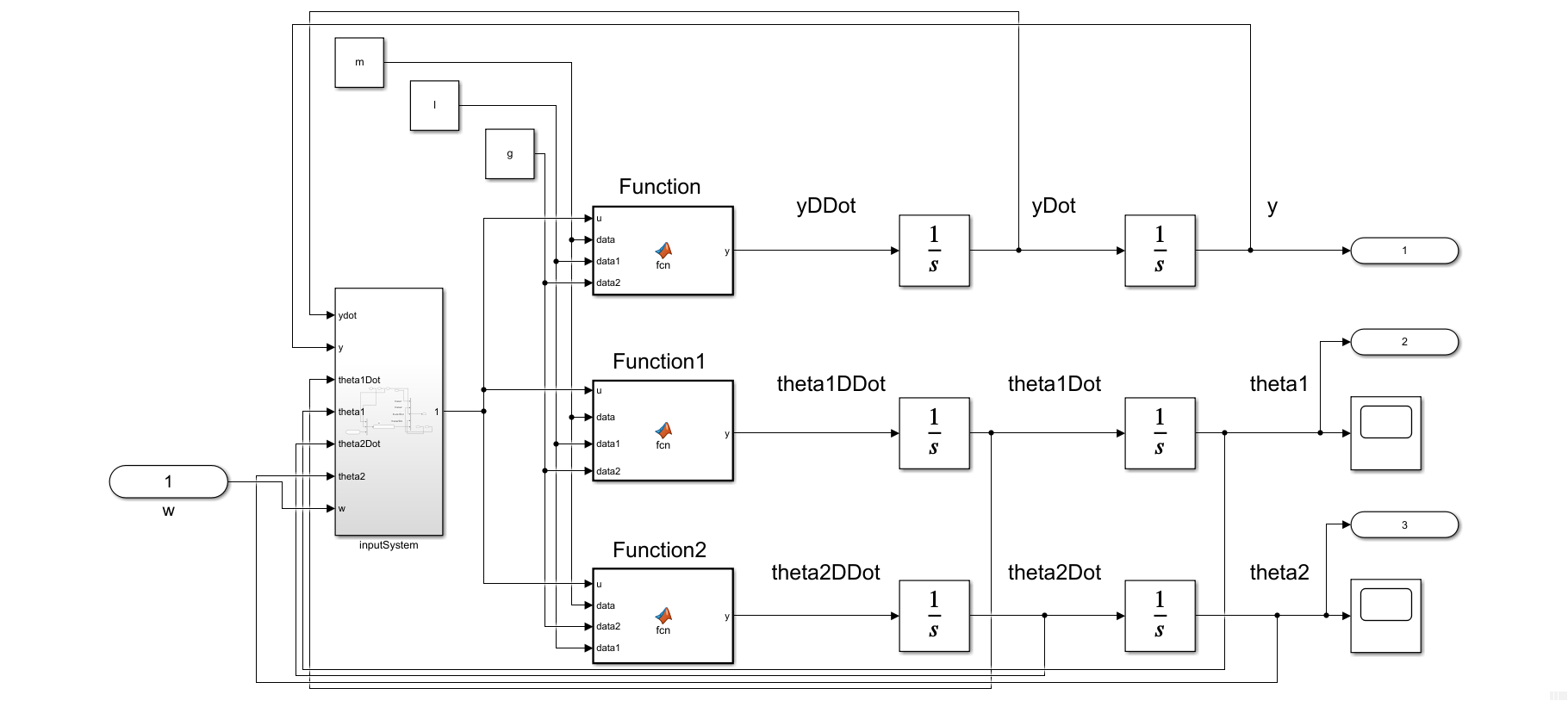
1. Using MATLAB, obtain the poles and zeros of the system linearized about .
2. Consider a sinusoidal disturbance input of the form

Choose so that the steady state response of the linearized system to this disturbance is zero. Simulate both the nonlinear system and the linearized system with zero initial conditions and this disturbance.

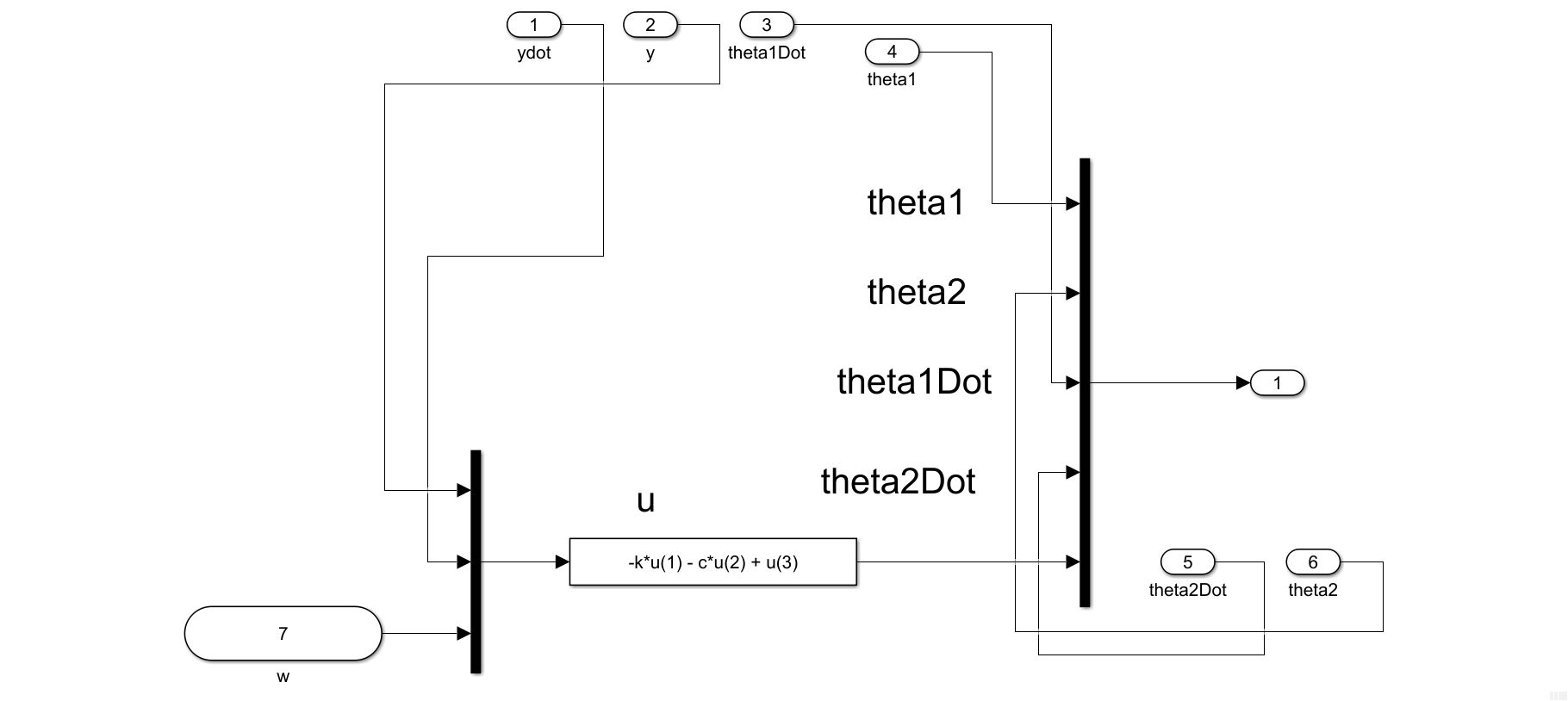
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| P1 | 2 | 1 | 1 | 1 | 1 | 1 | 0 |
| P2 | 2 | 1 | 1 | 1 | 0.99 | 1 | 0 |
| P3 | 2 | 1 | 0.5 | 1 | 1 | 1 | 0 |
| P4 | 2 | 1 | 1 | 1 | 0.5 | 1 | 0 |

(a).

The linear Simulink model is the following



The “inputSystem” subsystem is the following



The Embedded MATLAB Blocks are the following

Function:

function y = fcn(u, data, data1, data2)

%{

EMBEDDED MATLAB BLOCK FUNCTION

%}

m0 = data(1); m1 = data(2); m2 = data(3); l1 = data1(1); l2 = data1(2);

g = data2;

num = -m1\*l1\*sin(u(1))\*u(3)\*u(3) - m2\*l2\*sin(u(2))\*u(4)\*u(4)...

- m1\*g\*sin(u(1))\*cos(u(1)) - m2\*g\*sin(u(2))\*cos(u(2))...

+ u(5);

den = m0 + m1 + m2 - m1\*cos(u(1))^2 - m2\*cos(u(2))^2;

y = num / den;

end

Function1:

function y = fcn(u, data, data1, data2)

%{

EMBEDDED MATLAB BLOCK FUNCTION1

%}

m0 = data(1); m1 = data(2); m2 = data(3); l1 = data1(1); l2 = data1(2);

g = data2;

num = -(m1\*l1\*cos(u(1))\*sin(u(1))\*u(3)\*u(3) + m2\*l2\*cos(u(1))\*sin(u(2))\*u(4)\*u(4))...

+ m2\*g\*(sin(u(1))\*cos(u(2))^2 - cos(u(1))\*sin(u(2))\*cos(u(2)))...

- (m0 + m1 + m2)\*g\*sin(u(1)) + u(5)\*cos(u(1));

den = l1\*(m0 + m1 + m2 - m1\*cos(u(1))^2 - m2\*cos(u(2))^2);

y = num / den;

end

Function2:

function y = fcn(u, data, data2, data1)

%{

EMBEDDED MATLAB BLOCK FUNCTION2

%}

m0 = data(1); m1 = data(2); m2 = data(3); l1 = data1(1); l2 = data1(2);

g = data2;

num = -(m1\*l1\*cos(u(2,1))\*sin(u(1))\*u(3)\*u(3) + m2\*l2\*cos(u(2))\*sin(u(2))\*u(4)\*u(4))...

+ m1\*g\*(sin(u(2))\*cos(u(1))^2 - cos(u(2))\*sin(u(1))\*cos(u(1)))...

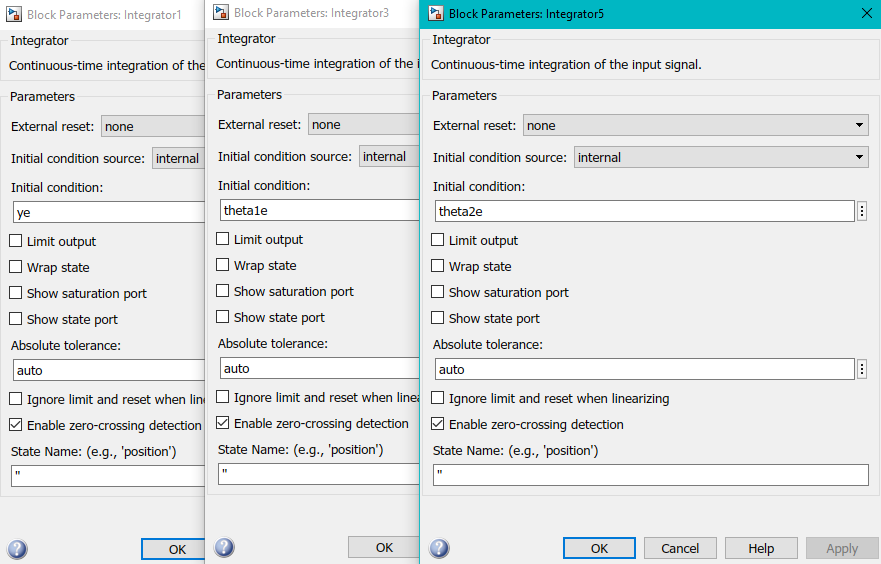
- (m0 + m1 + m2)\*g\*sin(u(2)) + u(5)\*cos(u(2));

den = l2\*(m0 + m1 + m2 - m1\*cos(u(1))^2 - m2\*cos(u(2))^2);

y = num / den;

end

For the equilibriums conditions E1, we set the initial conditions of the integrator block of y, , and correspondingly to ; like in the following windows,



And set arbitary values for the constants and (which presumably are the spring constant and damping coefficient)

|  |  |
| --- | --- |
|  |  |
| 1 | 1 |

Then we run the following MATLAB code to find the trim conditions and perform linearization on the system.

% Set parameters and equilibrium conditions

m = [2,1,1]; l = [1,0.5]; g = 1; % P4

ye = 0; theta1e = 0; theta2e = 0; % E1

k = 1; % spring constant

c = 1; % damping coefficient

xe = trim("db\_pend\_cart\_lin\_modInput")

[A, B, C, D] = linmod('db\_pend\_cart\_lin\_modInput',xe)

sys\_ss = ss(A, B, C, D); % get the state space system

sys\_tf = tf(sys\_ss) % get the transfer function

p = pole(sys\_tf) % get the eigenvalues

% Define a transfer function for each one to obtain the zeros

tf1 = tf(sys\_tf.Numerator(1), sys\_tf.Denominator(1))

tf2 = tf(sys\_tf.Numerator(2), sys\_tf.Denominator(2))

tf3 = tf(sys\_tf.Numerator(3), sys\_tf.Denominator(3))

z1 = zero(tf1)

z2 = zero(tf2)

z3 = zero(tf3)

p1 = pole(tf1)

p2 = pole(tf2)

p3 = pole(tf3)

The results are the following

|  |  |
| --- | --- |
| A = 6×6  0 0 0 1.0000 0 0  0 0 0 0 1.0000 0  0 0 0 0 0 1.0000  -0.5000 -0.5000 -0.5000 -0.5000 0 0  -0.5000 -1.5000 -0.5000 -0.5000 0 0  -1.0000 -1.0000 -3.0000 -1.0000 0 0 | B = 6×1  0  0  0  0.5000  0.5000  1.0000 |
| C = 3×6  1 0 0 0 0 0  0 1 0 0 0 0  0 0 1 0 0 0 | **D = 3×1**  **0**  **0**  **0** |

The transfer function for the first input “tf1” (for ):

|  |
| --- |
| tf1 =    0.5 s^4 + 6.688e-17 s^3 + 1.5 s^2 + 6.688e-17 s + 1  ---------------------------------------------------  s^6 + 0.5 s^5 + 5 s^4 + 1.5 s^3 + 5.5 s^2 + s + 1    Continuous-time transfer function. |

The corresponding poles and zeros

|  |  |
| --- | --- |
| p1 = 6×1 complex  -0.1259 + 1.8518i  -0.1259 - 1.8518i  -0.0205 + 1.1189i  -0.0205 - 1.1189i  -0.1036 + 0.4702i  -0.1036 - 0.4702i | z1 = 4×1 complex  -0.0000 + 1.4142i  -0.0000 - 1.4142i  0.0000 + 1.0000i  0.0000 - 1.0000i |

The transfer function for the first input “tf2” (for ):

|  |
| --- |
| tf2 =    0.5 s^4 + 8.668e-17 s^3 + s^2 + 1.742e-16 s + 4.697e-18  -------------------------------------------------------  s^6 + 0.5 s^5 + 5 s^4 + 1.5 s^3 + 5.5 s^2 + s + 1    Continuous-time transfer function. |

The corresponding poles and zeros

|  |  |
| --- | --- |
| p2 = 6×1 complex  -0.1259 + 1.8518i  -0.1259 - 1.8518i  -0.0205 + 1.1189i  -0.0205 - 1.1189i  -0.1036 + 0.4702i  -0.1036 - 0.4702i | z2 = 4×1 complex  0.0000 + 1.4142i  0.0000 - 1.4142i  -0.0000 + 0.0000i  -0.0000 - 0.0000i |

The transfer function for the first input “tf3” (for ):

|  |
| --- |
| tf3 =    s^4 + 2.388e-16 s^3 + s^2 + 1.254e-16 s - 6.89e-18  --------------------------------------------------  s^6 + 0.5 s^5 + 5 s^4 + 1.5 s^3 + 5.5 s^2 + s + 1    Continuous-time transfer function. |

The corresponding poles and zeros

|  |  |
| --- | --- |
| p3 = 6×1 complex  -0.1259 + 1.8518i  -0.1259 - 1.8518i  -0.0205 + 1.1189i  -0.0205 - 1.1189i  -0.1036 + 0.4702i  -0.1036 - 0.4702i | z3 = 4×1 complex  -0.0000 + 1.0000i  -0.0000 - 1.0000i  -0.0000 + 0.0000i  0.0000 + 0.0000i |

(b).

Since the code in part (a)

[xe, ue, ye, dxe] = trim("db\_pend\_cart\_lin\_modInput")

where the system was linearized for an equilibrium point of (0, 0, 0). The equilibrium condition for the input was found to be 0 by the variable “ue” which is highlighted red above.

Thus, if

and

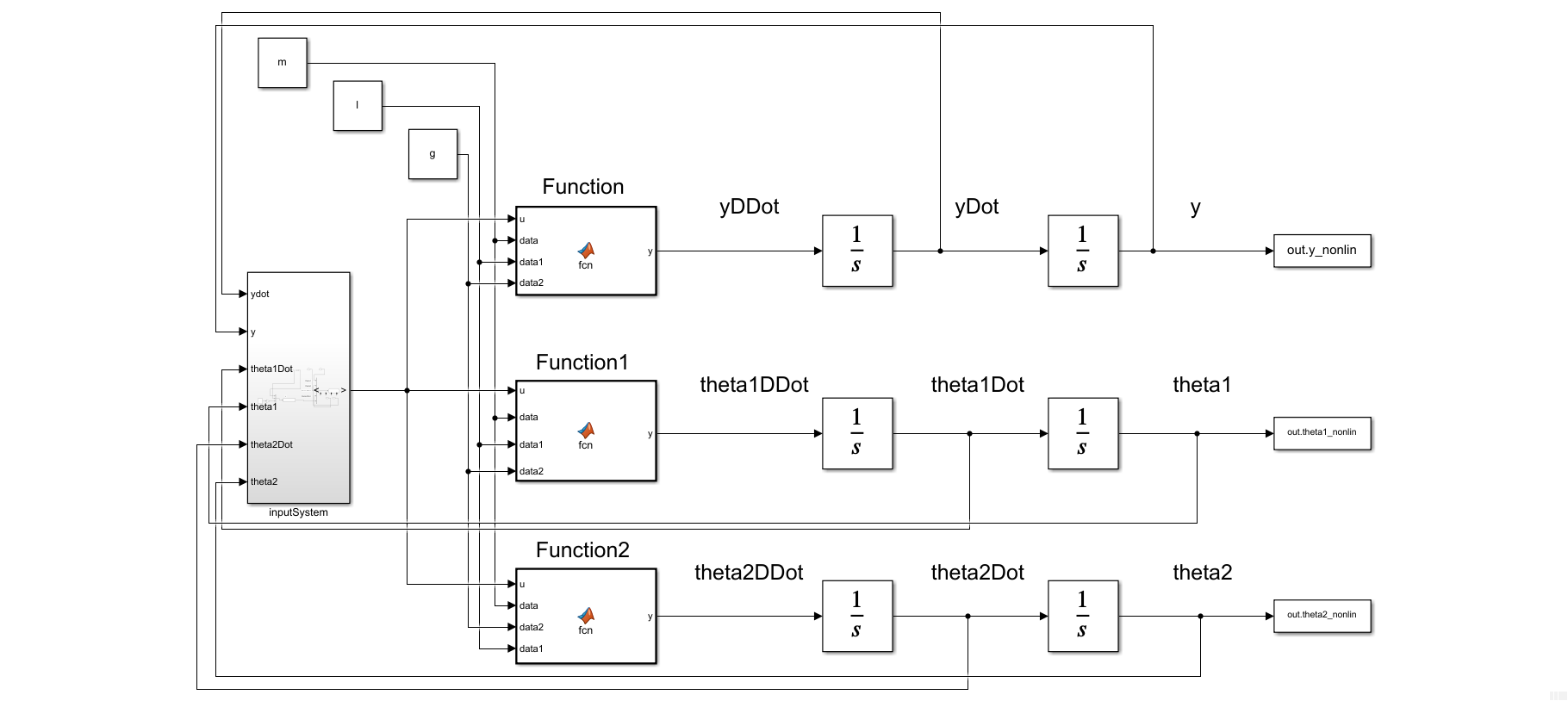
For the initial condition,

Thus,

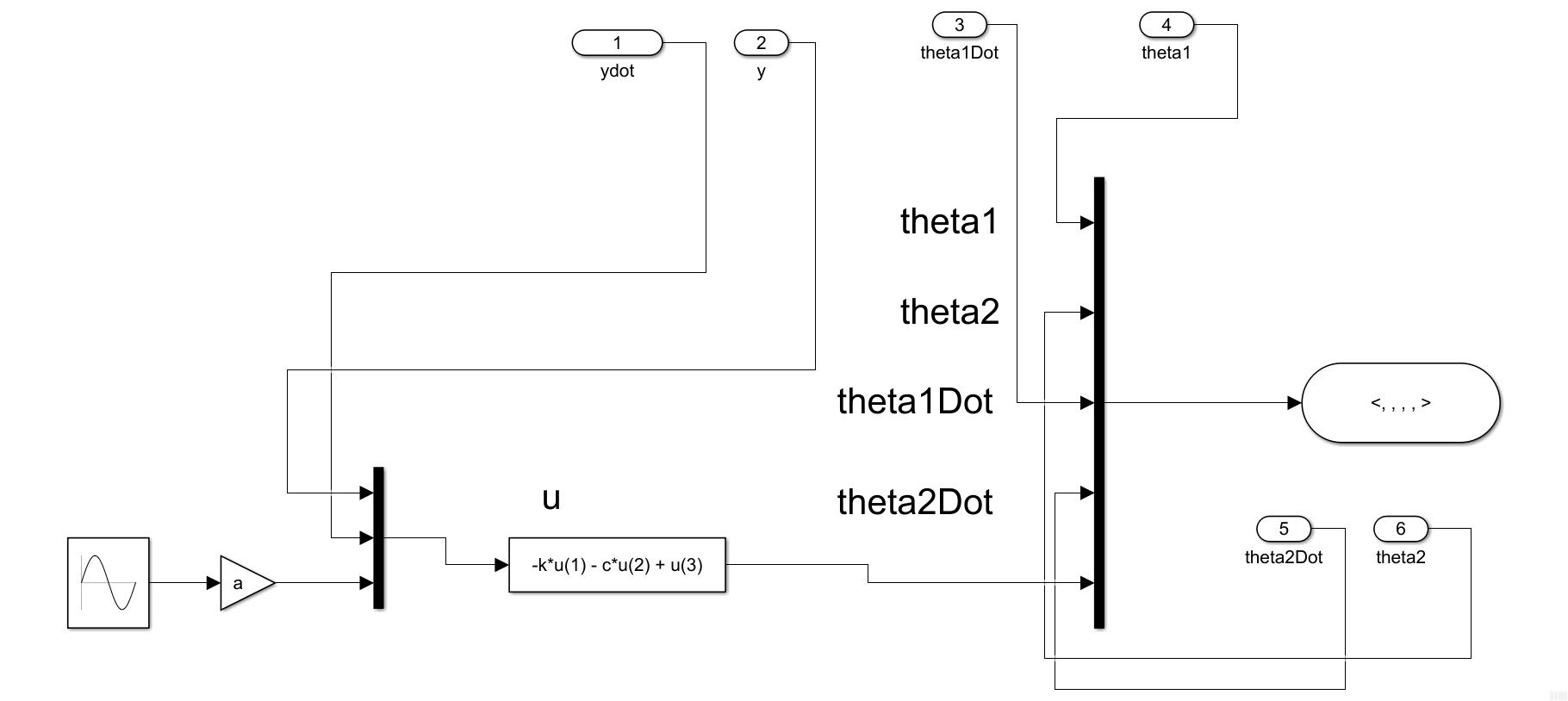
For E1, the initial condition

Then,

For the nonlinear system we use the following model

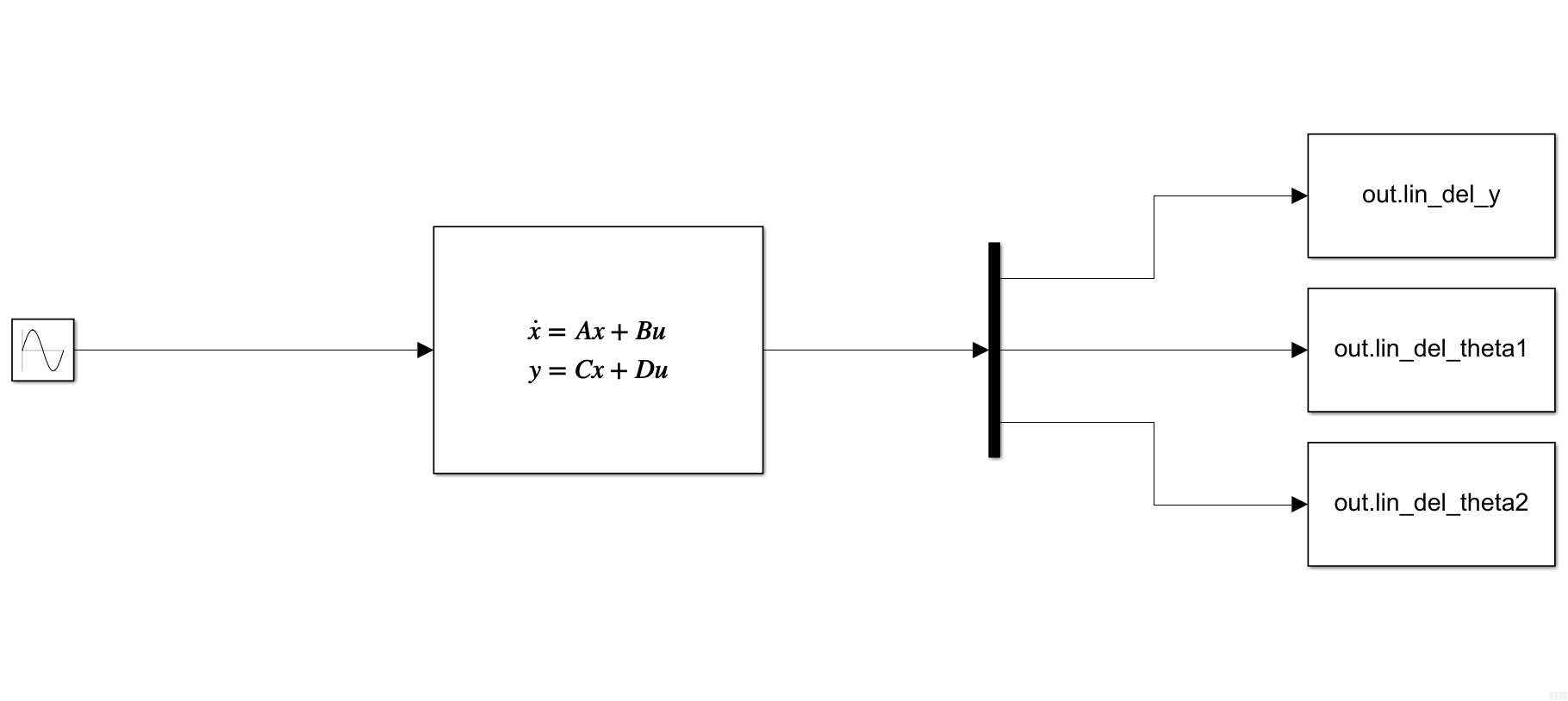


The subsystem “inputSystem” is the following



The Embedded MATLAB Blocks are all the same as the linearization model.

For the linearized one we use the following model

Using MATLAB, we can solve of this matrix for the linearized model it becomes

Solving each numerator to be zero, we find that the combination of (for the input of y) which corresponds to the first row

So, for when we can choose an omega of 1!

Then we have the following plot of the simulation.

Graphical user interface

Description automatically generated

The y-displacement does decay to 0. So the simulation is congruent with our results.

This was done with the following code

% (b)

warning("off")

% Set parameters and equilibrium conditions

m = [2,1,1]; l = [1,0.5]; g = 1; % P4

ye = 0; theta1e = 0; theta2e = 0; % E1

k = 1; % spring constant

c = 1; % damping coefficient

a = 1; % amplitude in disturbance

omega = 1;

xe = trim("db\_pend\_cart\_lin\_modInput")

[A, B, C, D] = linmod("db\_pend\_cart\_lin\_modInput",xe)

% % % Proof

% syms t s

% G = vpa(simplify(C\*inv(s\*eye(6)-A)\*B+D))

% res1 = solve(G(1)==0, s)

% res2 = solve(G(2)==0, s)

% res3 = solve(G(3)==0, s)

% Initial conditions for the perturbation variables

del\_yi = 0; del\_theta1i = 0; del\_theta2i = 0;

del\_yi\_dot = 0; del\_theta1i\_dot = 0; del\_theta2i\_dot = 0;

% Initial conditions for the variables

yi = ye + del\_yi;

theta1i = theta1e + del\_theta1i;

theta2i = theta2e + del\_theta2i;

yi\_dot = del\_yi\_dot;

theta1i\_dot = del\_theta1i\_dot;

theta2i\_dot = del\_theta2i\_dot;

% Linearized system

IC\_lin = [del\_yi, del\_theta1i, del\_theta2i, 0, 0, 0];

lin\_res = sim('ss\_lin\_sys');

% Nonlinear system

nonlin\_res = sim('db\_pend\_cart\_nonlin\_modInput')

% get values from results

tspan\_lin = lin\_res.tout;

y\_lin = lin\_res.lin\_del\_y.signals.values;

theta1\_lin = lin\_res.lin\_del\_theta1.signals.values;

theta2\_lin = lin\_res.lin\_del\_theta2.signals.values;

tspan\_nonlin = nonlin\_res.tout;

y\_nonlin = nonlin\_res.y\_nonlin.signals.values;

theta1\_nonlin = nonlin\_res.theta1\_nonlin.signals.values;

theta2\_nonlin = nonlin\_res.theta2\_nonlin.signals.values;

% Plotting

fig1 = figure('Renderer',"painters", 'Position', [10 10 900 1000]);

subplot(3,1,1)

hold on; grid on; grid minor; box on;

plot(tspan\_lin, y\_lin)

plot(tspan\_nonlin, y\_nonlin)

ylabel('y [m]')

hold off

subplot(3,1,2)

hold on; grid on; grid minor; box on;

plot(tspan\_lin, theta1\_lin)

plot(tspan\_nonlin, theta1\_nonlin)

ylabel('$\theta\_1$ [rad]')

hold off

subplot(3,1,3)

hold on; grid on; grid minor; box on;

plot(tspan\_lin, theta2\_lin)

plot(tspan\_nonlin, theta2\_nonlin)

ylabel('$\theta\_2$ [rad]')

xlabel('time, [sec]')

h = legend('linear','nonlinear'); set(h, 'Position', [0.8, 0.05, .1, .025]);

hold off

sgtitle({['P4/E1 Time History of Double Pendulum Cart System with Passive ' ...

'Stabilization'], ' and Disturbance Input $0\leq t \leq 100$ - T. Koike'})

saveas(fig1, 'p9\_b.png')