McMaster University

ELEC ENG 4CL4

Control Systems Design - Fall 2018

Phase 4 - State Feedback Control Synthesis of Inverted Pendulum

---

Decemeber 3rd, 2018

Instructor:
Dr. Shahin Sirouspour

Authors:
Michael Bagnowski - 400026446
Andy F. Gonzalez - 400044420

Teaching Assistants:
Keyvan Mohammadi
Sahand Chaffari
Ali Grivani

Abstract

For phase 4, the observer was implemented for the inverted pendulum model. In phase 4, concepts of state feedback control and control using a state estimator were introduced. First, the open loop response of the linearized system was analyzed for stability. This showed that the system was not stable due to poles in the right half plane. In order to stabilize the system response, state feedback control was introduced. This involved adding a gain controller to the closed loop response, so that the A matrix was replaced with A-BK. By calculating values of the K matrix, poles were moved to desired locations in the left half plane, stabilizing the system. To design the observer, a similar method was implemented. However, to make the system stable the observer poles were placed to converge ten times faster than the feedback controller. This pole placement minimizes the error from the observer. Two kinds of observers were simulated (state-estimator and numerical differentiator). The results indicated that a state estimator is a better design for an observer.

1 Equilibrium Positions

Case One

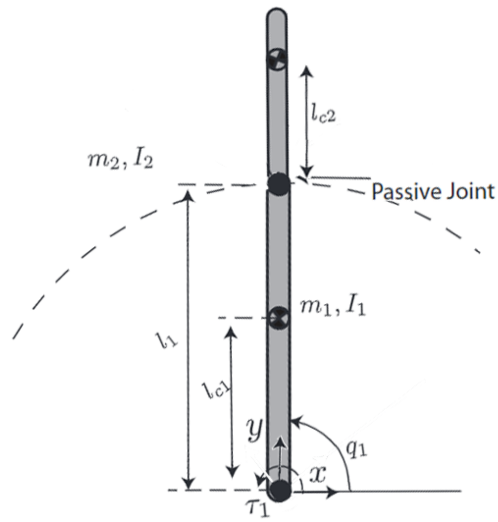


Figure 1: Equilibrium Case One Position of Dual Link Inverted Pendulum

Case Two

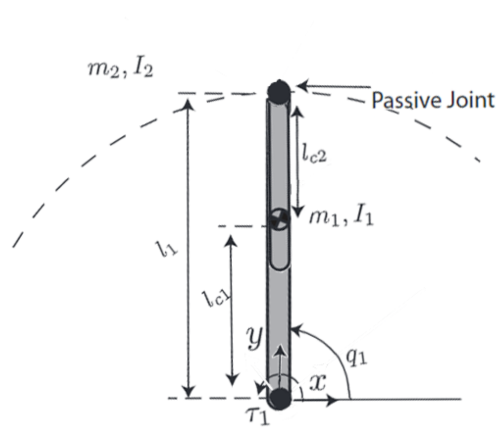


Figure 2: Equilibrium Case Two Position of Dual Link Inverted Pendulum

2 State Estimator (Observer)

2.1 Observability of both equilibrium positions.

using the Matlab function obsv(A,C) we were able to analyze the observability of both equilibrium positions within the linearized dynamics. For postion one we have the following matrices A_{pos1},

$$A_{pos1} = \begin{bmatrix} 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 1.8600 & -4.8679 & -3.3267 & 0.0033 \\ 115.1393 & 141.3019 & 12.9365 & -0.0739 \end{bmatrix} \tag{1}$$

$$A_{pos2} = \begin{bmatrix} 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 1.8600 & -4.8679 & -3.3267 & -0.0016 \\ -118.8593 & -131.5660 & -6.2831 & -0.0642 \end{bmatrix} \tag{2}$$

with the consistent C matrix defined by,

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3}$$

the observability matrix for position one was,

$$P_{observ1} = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 1.8600 & -4.8679 & -3.3267 & 0.0033 \\ 115.1393 & 141.3019 & 12.9365 & -0.0739 \\ -5.8106 & 16.6571 & 12.9695 & -4.8791 \\ 15.5524 & -73.4171 & 71.1469 & 141.3497 \end{bmatrix} rank(P_{observ1}) = 4 \tag{4}$$

while the observability for postion two was,

$$P_{observ2} = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 1.8600 & -4.8679 & -3.3267 & -0.0016 \\ -118.8593 & -131.5660 & -6.2831 & -0.0642 \\ -5.9986 & 16.4036 & 12.9371 & -4.8625 \\ -4.0589 & 39.0285 & -97.5540 & -131.5519 \end{bmatrix} rank(P_{observ2}) = 4 \tag{5}$$

Therefore, from the results of the observability test we can conclude that the linearization of both equilibrium positions are observable.

2.2 Designing The Observer

To design the observer, we place the eigen values of the new system A-LC such the the observer error reaches zero much faster than the state feedback controller. To do this, we use the function $place(A^T, C^T, p)$ were p are the pole positions 10 times larger than the state feedback controller such to reduce error. The values for the unknown angular velocities will be calculated via numerical integration. The gain of the observer for both positions are as follows.

$$Po_1 = \begin{bmatrix} 219.2 & 24.08 \\ 38.64 & 227.4 \\ 11304.06 & 2512.8 \\ 5727.29 & 13004.71 \end{bmatrix} Po_2 = \begin{bmatrix} 220.53 & -24.02 \\ -32.25 & 226.08 \\ 11443.12 & -2518.21 \\ -4307.87 & 12430.02 \end{bmatrix} \tag{6}$$

Based on these results as well as the results from phase 3 the observer was designed as such.

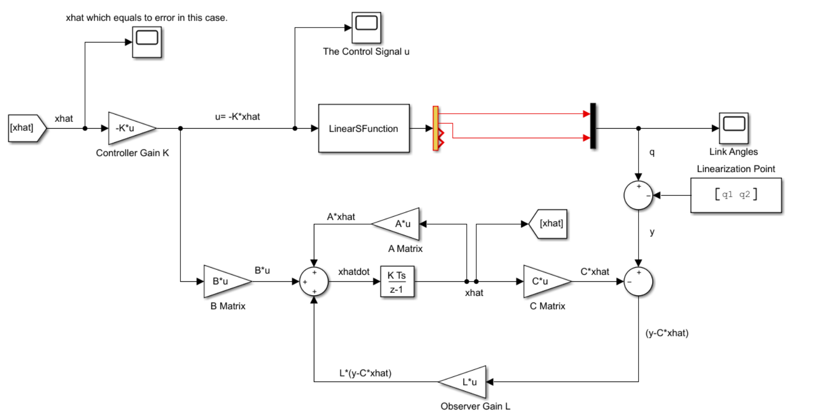
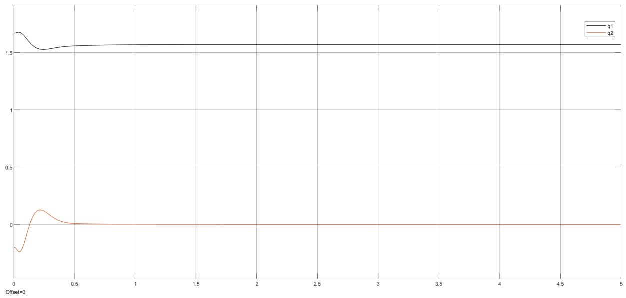


Figure 3: Closed loop observer and state feedback controller Simulink design

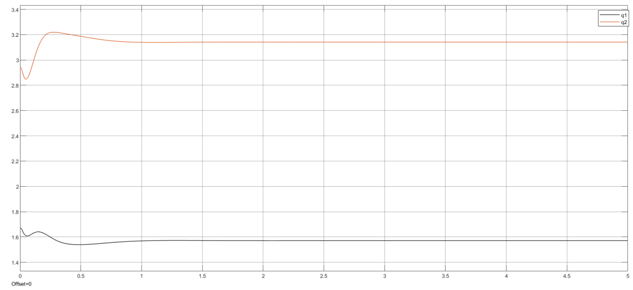
3 State-feedback Control using State Estimator

3.1 Simulating Linearized Equilibrium Positions

When simulating the linearized model with the observer/state-estimator design, We observed the same response as the non-linear model with the state-estimator/observer.Please note that for these simulations, q1 is in black and q2 is in red.



(a) Linearized model at equilibrium point 1 simulation.



(b) Linearized model at equilibrium point 2 simulation.

as we can see from the figures above, both links reach equilibrium quite fast.

3.2 Simulating Full Non-Linear Model

To simulate the full non-linear model, we had to swap the LinearSFunction from figure 3 with a non-linear sfunction. The simulation setup is displayed below.

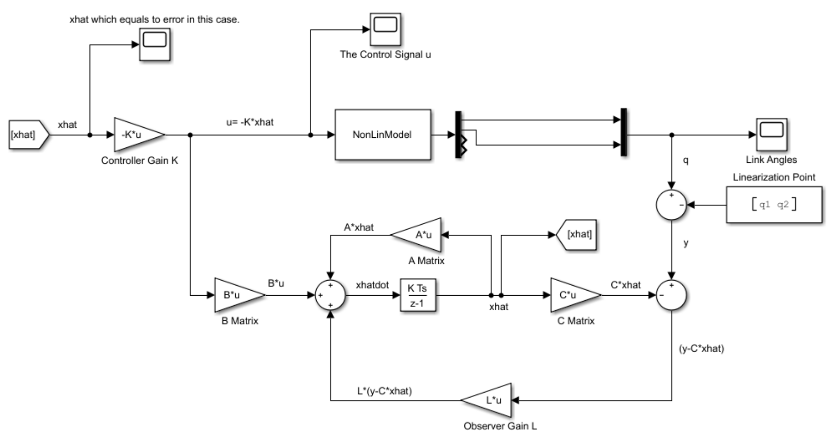
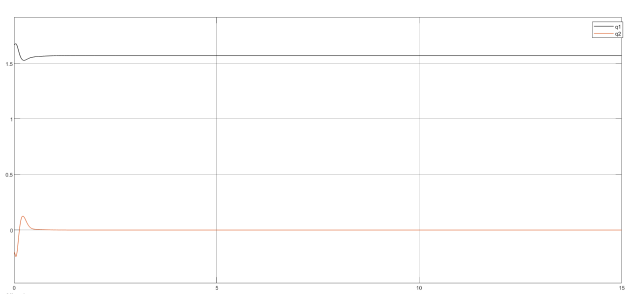
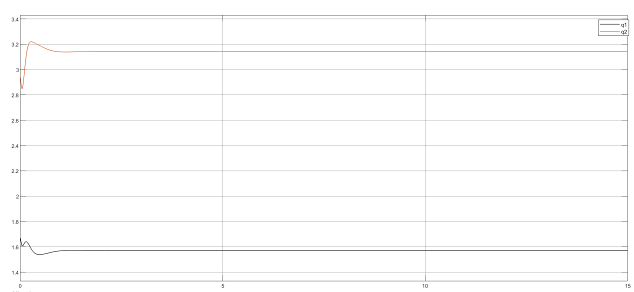


Figure 5: Closed loop observer and state feedback controller Simulink design for non-linear model



(a) Non-Linear model at equilibrium point 1 simulation.



(b) Non-Linear model at equilibrium point 2 simulation.

3.3 Simulating Non-Linear Model with Numerical Differentiator

The second task of this section was to simulate the nonlinear model with a numerical differentiator. The Simulink setup is shown below.

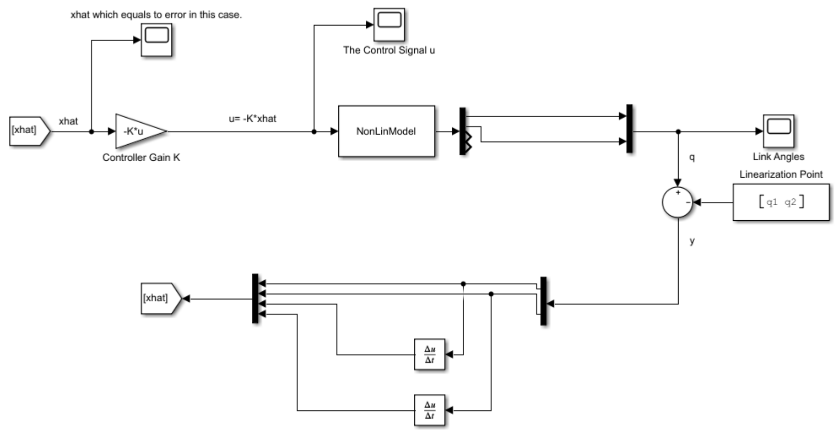
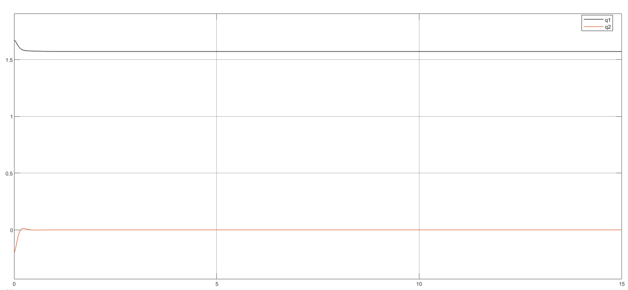
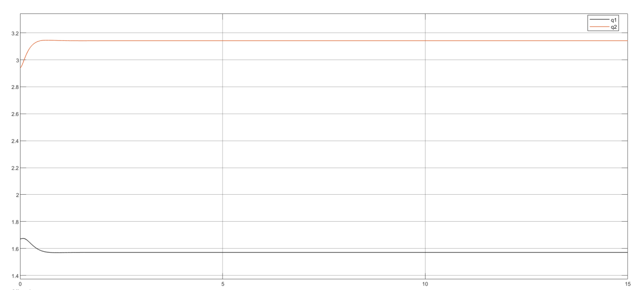


Figure 7: Closed loop observer and state feedback controller Simulink design for non-linear model with a numerical diferentiator



(a) Non-Linear model at equilibrium point 1 simulation with numerical differentiator.



(b) Non-Linear model at equilibrium point 2 simulation with numerical differentiator.

Again, the results for the numerical differentiator agree with all previous simulations. However, one thing to note with a numerical differentiator is that the system is very sensitive to noise. We go on to simulate the system with added gausian white noise and for both inputs the system is highly unstable.

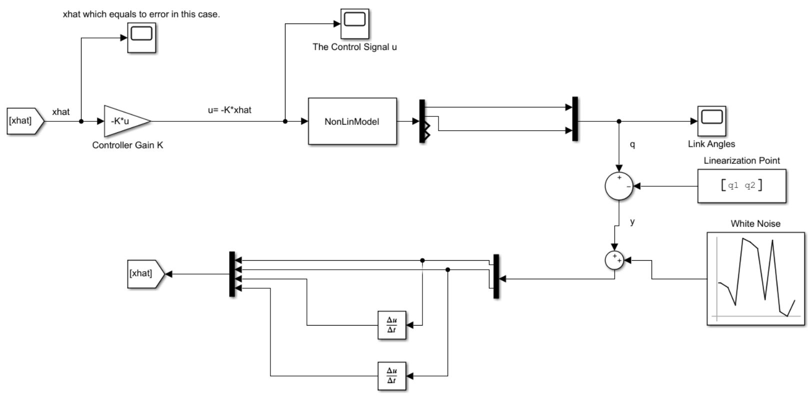
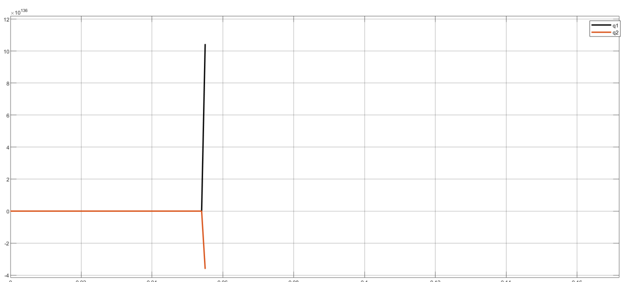
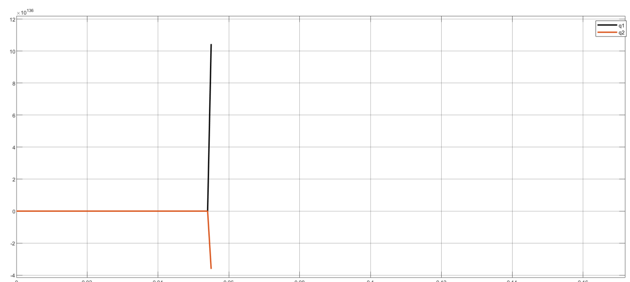


Figure 9: Closed loop observer and state feedback controller Simulink design for non-linear model with a numerical diferentiator and input noise



(a) Non-Linear model at equilibrium point 1 simulation with numerical differentiator and input noise.

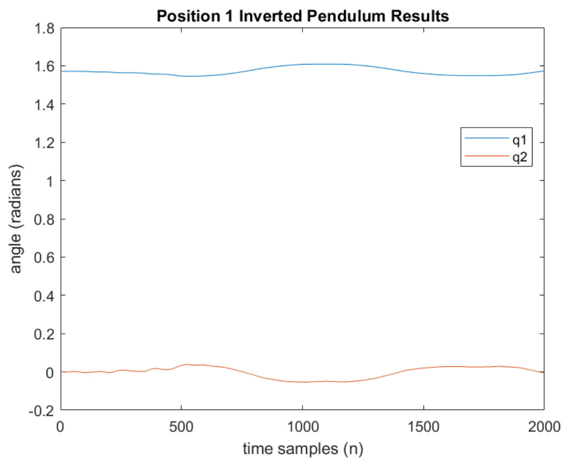


(b) Non-Linear model at equilibrium point 2 simulation with numerical differentiator and input noise.

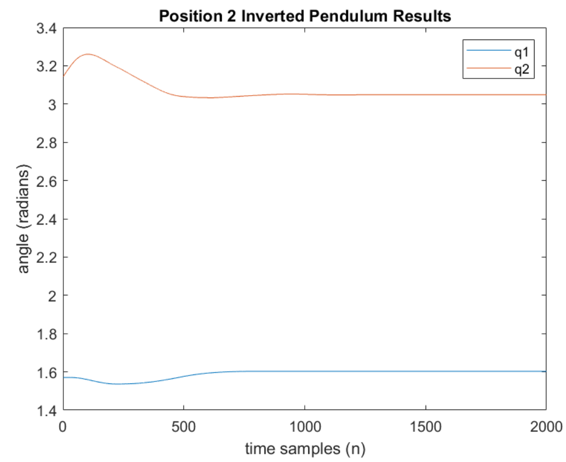
As you can tell from the simulation results, the presence of noise in the observer produces high instability because the differentiators essentially amplify the noise.

4 Experimental Evaluation of the Controllers

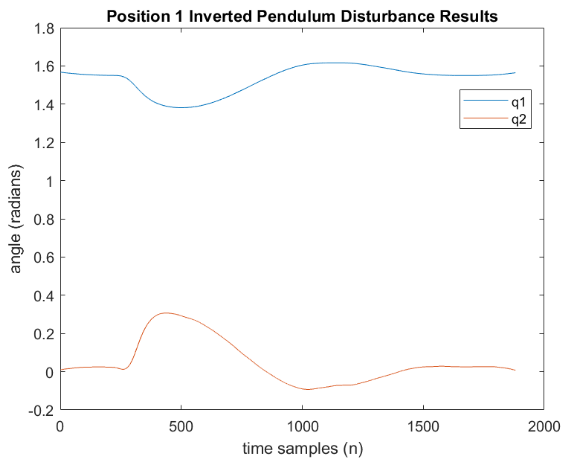
4.1 Results from system actualization



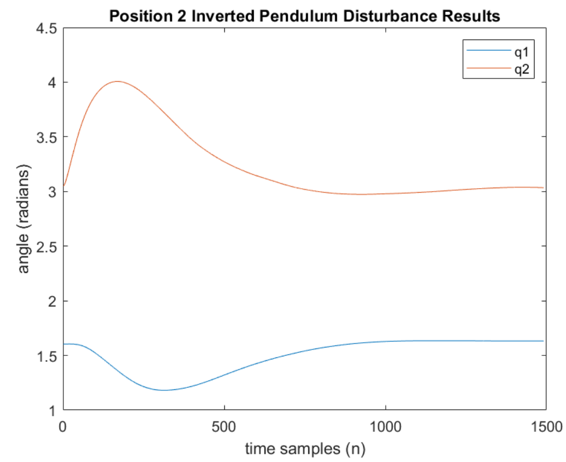
(a) System actualization output at equilibrium position 1.



(b) System actualization output at equilibrium position 2.



(c) System actualization output at equilibrium position 1 with disturbance input.



(d) System actualization output at equilibrium position 2 with disturbance input.

From the results, it is evident that position 2 behaves almost identical to the simulations. However position one has a slight oscillation in its steady state response. This is probably because position one is more unstable than position two. Therefore, when position one reaches equilibrium, its has a greater tendency to slightly deviate and thus have to be rectified by the controller.

4.2 Actualization with a numerical differentiator

Because of the high instability of a numerical differentiator, we did not actually apply it to the physical model. Refer to section 3.3. to see our results in simulation.

5 Results/Conclusion

The results from our simulations and physical actualization concluded that a state-estimator is the best design for an observer for the dual-link inverted pendulum system. It is essential when designing an observer to verify that the system is observable. This is easily accomplished by checking the rank of the observability matrix defined in section 2.1. The observer pole must be place so that the observer system converges much faster than the feedback controller. This minimizes the error in the system and allows the system to perform better under input disturbance.

The physical actualization for position 2 behaved identical to the simulations. However position 1 showed slightly different dynamics. We propose this to be because of the greater instability in position 1 than position 2. Therefore, smaller disturbances in position 1 will cause the system to react. This produces a slight oscillation in the dynamics for equilibrium position one which we do not see in position 2. Both physical actualization's responded as predicted to slight disturbances in equilibrium.

6 Contributions

Michael Bagnowski – 400026446 : Simulink Models, Abstract , Results/Conclusion

Andy F.Gonzalez – 400044420 : observer design, observability calculations, pole positioning

Appendix

Observer Design

```
%% Linearization Point
q1 = pi/2;
q2 = 0;
g = 9.81;
X0=[q1+.1 q2-.2 0 0];
T=.001;
NewPanto=1;
%% System Parameters
Parameters = [11.9253 0.4203 0.1455 7.2462 1.8150 35.1492 0.0089];
%% Linear System Definition
M = [Parameters(1)+2*Parameters(2)*cos(q2),  Parameters(3)+Parameters(2)*cos(q2);
    Parameters(3)+Parameters(2)*cos(q2),     Parameters(3)];

F = [Parameters(6), 0;
    0, Parameters(7)];

K = [-Parameters(4)*g*sin(q1)-sin(q1+q2)*g*Parameters(5),  -sin(q1+q2)*g*Parameters(5);
    -sin(q1+q2)*g*Parameters(5),                          -sin(q1+q2)*g*Parameters(5)];

A = [zeros(2,2),    eye(2,2);
     -M^-1*K,        -M^-1*F];

B = [zeros(2,2);
    M^-1]*[1;0];

C = [1 0 0 0;
     0 1 0 0];

%% Stability Analysis
EigenValues=eig(A)

%% Controllability and Controller Gain Design
Pc=ctrb(A,B)
rank(Pc)
p=[-15 -13 -10 -8];
K = place(A,B,p)
%% Observability and Estimator Gain Design
Po=obsv(A,C)
rank(Po)
po=10*p;
```

```
L = transpose(place(A', C',po))
```

Simulation Results

```
% simulation results
close all;
%% position 1
pos1 = load('equilibriumpositon_1_realsim.mat','outputy');
pos1 = pos1.outputy.signals.values;
figure;
range = 1:2000;
disturbance = 43170:45050;
pos1(:,1) = pos1(:,1) + pi/2;
plot(pos1(range,:));
title('Position 1 Inverted Pendulum Results');
xlabel("time samples (n)");
ylabel("angle (radians)");
legend("q1","q2");
figure;
plot(pos1(disturbance,:));
title('Position 1 Inverted Pendulum Disturbance Results');
xlabel("time samples (n)");
ylabel("angle (radians)");
legend("q1","q2");
%% position 2
pos2 = load('equilibriumpositon_2_realsim.mat','outputy');
pos2 = pos2.outputy.signals.values;
disturbance = 38570:40060;
figure;
range = 1:2000;
pos2(:,1) = pos2(:,1) + pi/2;
pos2(:,2) = pos2(:,2) + pi;
plot(pos2(range,:));
title('Position 2 Inverted Pendulum Results');
xlabel("time samples (n)");
ylabel("angle (radians)");
legend("q1","q2");
figure;
plot(pos2(disturbance,:));
title('Position 2 Inverted Pendulum Disturbance Results');
xlabel("time samples (n)");
ylabel("angle (radians)");
legend("q1","q2");
```

Non-Linear S-Function

```
function [sys,x0,str,ts,simStateCompliance] = NonLinModel(t,x,u,flag,X0,Parameters)
%SFUNTMPL General MATLAB S-Function Template
%   With MATLAB S-functions, you can define you own ordinary differential
%   equations (ODEs), discrete system equations, and/or just about
%   any type of algorithm to be used within a Simulink block diagram.
%
%   The general form of an MATLAB S-function syntax is:
%       [SYS,X0,STR,TS,SIMSTATECOMPLIANCE] = SFUNC(T,X,U,FLAG,P1,...,Pn)
%
%   What is returned by SFUNC at a given point in time, T, depends on the
%   value of the FLAG, the current state vector, X, and the current
%   input vector, U.
%
%   FLAG   RESULT             DESCRIPTION
%   -----  ------             --------------------------------------------
%   0      [SIZES,X0,STR,TS]  Initialization, return system sizes in SYS,
%                             initial state in X0, state ordering strings
%                             in STR, and sample times in TS.
%   1      DX                 Return continuous state derivatives in SYS.
%   2      DS                 Update discrete states SYS = X(n+1)
%   3      Y                  Return outputs in SYS.
%   4      TNEXT              Return next time hit for variable step sample
%                             time in SYS.
%   5                         Reserved for future (root finding).
%   9      []                 Termination, perform any cleanup SYS=[].
%
%
%   The state vectors, X and X0 consists of continuous states followed
%   by discrete states.
%
%   Optional parameters, P1,...,Pn can be provided to the S-function and
```

```
%   used during any FLAG operation.
%
%   When SFUNC is called with FLAG = 0, the following information
%   should be returned:
%
%      SYS(1) = Number of continuous states.
%      SYS(2) = Number of discrete states.
%      SYS(3) = Number of outputs.
%      SYS(4) = Number of inputs.
%               Any of the first four elements in SYS can be specified
%               as -1 indicating that they are dynamically sized. The
%               actual length for all other flags will be equal to the
%               length of the input, U.
%      SYS(5) = Reserved for root finding. Must be zero.
%      SYS(6) = Direct feedthrough flag (1=yes, 0=no). The s-function
%               has direct feedthrough if U is used during the FLAG=3
%               call. Setting this to 0 is akin to making a promise that
%               U will not be used during FLAG=3. If you break the promise
%               then unpredictable results will occur.
%      SYS(7) = Number of sample times. This is the number of rows in TS.
%
%
%      X0     = Initial state conditions or [] if no states.
%
%      STR    = State ordering strings which is generally specified as [].
%
%      TS     = An m-by-2 matrix containing the sample time
%               (period, offset) information. Where m = number of sample
%               times. The ordering of the sample times must be:
%
%               TS = [0      0,      : Continuous sample time.
%                     0      1,      : Continuous, but fixed in minor step
%                                      sample time.
%                     PERIOD OFFSET, : Discrete sample time where
%                                      PERIOD > 0 & OFFSET < PERIOD.
```

```
%                     -2     0];     : Variable step discrete sample time
%                                      where FLAG=4 is used to get time of
%                                      next hit.
%
%                 There can be more than one sample time providing
%                 they are ordered such that they are monotonically
%                 increasing. Only the needed sample times should be
%                 specified in TS. When specifying more than one
%                 sample time, you must check for sample hits explicitly by
%                 seeing if
%                    abs(round((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
%                 is within a specified tolerance, generally 1e-8. This
%                 tolerance is dependent upon your model's sampling times
%                 and simulation time.
%
%                 You can also specify that the sample time of the S-function
%                 is inherited from the driving block. For functions which
%                 change during minor steps, this is done by
%                 specifying SYS(7) = 1 and TS = [-1 0]. For functions which
%                 are held during minor steps, this is done by specifying
%                 SYS(7) = 1 and TS = [-1 1].
%
%     SIMSTATECOMPLIANCE = Specifices how to handle this block when saving and
%                          restoring the complete simulation state of the
%                          model. The allowed values are: 'DefaultSimState',
%                          'HasNoSimState' or 'DisallowSimState'. If this value
%                          is not speficified, then the block's compliance with
%                          simState feature is set to 'UknownSimState'.


%   Copyright 1990-2010 The MathWorks, Inc.
```

```
%   $Revision: 1.18.2.5 $

%
% The following outlines the general structure of an S-function.
%
switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes(X0);

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1,
    sys=mdlDerivatives(t,x,u,X0,Parameters);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%%%%%%
  % GetTimeOfNextVarHit %
  %%%%%%%%%%%%%%%%%%%%%%%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
  case 9,
    sys=mdlTerminate(t,x,u);

  %%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%
  otherwise
```

```
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));

end

% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes(X0)

%
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
%
% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.
%
sizes = simsizes;

sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0  = X0;

```

```
%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% Specify the block simStateCompliance. The allowed values are:
%    'UnknownSimState', < The default setting; warn and assume DefaultSimState
%    'DefaultSimState', < Same sim state as a built-in block
%    'HasNoSimState',   < No sim state
%    'DisallowSimState' < Error out when saving or restoring the model sim state
simStateCompliance = 'UnknownSimState';

% end mdlInitializeSizes

%
%============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%============================================================================
%
function sys=mdlDerivatives(t,x,u,X0,Parameters)
q1=x(1); q2=x(2); q1dot=x(3); q2dot=x(4);
g=9.81;
bf1=Parameters(6);
bf2=Parameters(7);
bf1=0;
bf2=0;

% bf1=20/.32*bf2;
% kk=.4*.052*9/.4*1000;
kk=1;
h   = -Parameters(2)*sin(q2);
d11 = Parameters(1)+2*Parameters(2)*cos(q2);
d12 = Parameters(2)*cos(q2)+Parameters(3);
d21 = d12;
d22 = Parameters(3);
```

```
g1  = Parameters(4)*g*cos(q1)+Parameters(5)*g*cos(q1+q2);
g2  = Parameters(5)*g*cos(q1+q2);
D   = [d11 d12;d21 d22];
C   = [h*q2dot   h*(q1dot+q2dot);
      -h*q1dot      0];
G   = [g1;g2];

sys(1)= x(3);
sys(2)= x(4);

qddot=inv(D)*([u/kk;0]-C*[q1dot;q2dot]-[bf1*q1dot;bf2*q2dot]-G);
sys(3:4)=qddot;

% end mdlDerivatives

%
%============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%============================================================================
%
function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

%
%============================================================================
% mdlOutputs
% Return the block outputs.
%============================================================================
%
function sys=mdlOutputs(t,x,u)

sys = x;

```

```
% end mdlOutputs

%
%===========================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%===========================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;    %  Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%
%===========================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%===========================================================================
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

Linear S-Function

```
function [sys,x0,str,ts,simStateCompliance] = LinearSFunction(t,x,u,flag,X0,A,B,C)

switch flag,

   case 0,
    [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes(X0);

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1,
    sys=mdlDerivatives(t,x,u,X0,A,B);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,u,x,C);

  %%%%%%%%%%%%%%%%%%%%%%%
  % GetTimeOfNextVarHit %
  %%%%%%%%%%%%%%%%%%%%%%%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
  case 9,
    sys=mdlTerminate(t,x,u);

  %%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%
  otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));

end
```

```
% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes(X0)

%
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
%
% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.
%
sizes = simsizes;

sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0 = X0;




%
% str is always an empty matrix
```

```
%
str = [];

%
% initialize the array of sample times
%
ts  = [0 0];

% Specify the block simStateCompliance. The allowed values are:
%    'UnknownSimState', < The default setting; warn and assume DefaultSimState
%    'DefaultSimState', < Same sim state as a built-in block
%    'HasNoSimState',   < No sim state
%    'DisallowSimState' < Error out when saving or restoring the model sim state
simStateCompliance = 'UnknownSimState';

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u,X0,A,B)
sys=A*x+B*u;

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)
```

```matlab



sys = [];

% end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys=mdlOutputs(t,u,x,C)

sys = x;

% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;    %  Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

```

```
%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```