Non Linear Equations of Motion for a Pole-Card System

```
clear; format compact
```

Part 0. System Definition and Parameter Declaration

Non-linear equations of motion for the system are derived using the euler-lagrange method as described in this article.

```
% Define the symbolic variables
syms x(t) theta(t) U
syms m M l b I g real

% Define the equations of motion
Eq1 = (M + m) * diff(x(t), t, 2) + b * diff(x(t), t) + ...
            m*l*diff(theta(t), t, t)*cos(theta(t)) - ...
            m*l*diff(theta(t), t)*sin(theta) == U;

Eq2 = (I + m*l^2)*diff(theta(t), t, 2) + m*g*l*sin(theta(t)) ...
        == -m*l*diff(x(t), t, 2)*cos(theta(t));

Eq = [Eq1; Eq2], vars = [x(t), theta(t)]
```

Eq(t) =

$$\begin{pmatrix} b\,\frac{\partial}{\partial t}\,x(t) + (M+m)\,\sigma_1 - l\,m\sin(\theta(t))\,\frac{\partial}{\partial t}\,\theta(t) + l\,m\cos(\theta(t))\,\sigma_2 = U \\ (m\,l^2 + \mathrm{I})\,\sigma_2 + g\,l\,m\sin(\theta(t)) = -l\,m\cos(\theta(t))\,\sigma_1 \end{pmatrix}$$

where

$$\sigma_1 = \frac{\partial^2}{\partial t^2}\,x(t)$$

$$\sigma_2 = \frac{\partial^2}{\partial t^2}\,\theta(t)$$

vars = $\begin{pmatrix} x(t) & \theta(t) \end{pmatrix}$

Part 1. NonLinear State Space Form Derivation from Equations of Motion

After implementing the second order ODEs, we have to manipulate it to be in a get the state space representation of it.

```
% Turn to a Single Order Set of ODEs as State Space Representation
[V, S, tmp] = reduceDifferentialOrder(Eq, vars);
[M_eqns, f_eqns] = massMatrixForm(V, S);
StSp = simplify(M_eqns \ f_eqns);
```

```
% Rename State Variables to x
SysOrder = numel(S);
syms x [SysOrder 1] real
StSp = subs(StSp, S, x)
```

```
StSp =
```

$$\begin{pmatrix} x_3 \\ x_4 \\ \dfrac{\mathrm{I}\,U - \mathrm{I}\,b\,x_3 + U\,l^2\,m + l^3\,m^2\,x_4\sin(x_2) - b\,l^2\,m\,x_3 + \dfrac{g\,l^2\,m^2\sin(2\,x_2)}{2} + \mathrm{I}\,l\,m\,x_4\sin(x_2)}{\sigma_1} \\ -\dfrac{l\,m\,(U\cos(x_2) - b\,x_3\cos(x_2) + g\,m\sin(x_2) + M\,g\sin(x_2) + l\,m\,x_4\cos(x_2)\sin(x_2))}{\sigma_1} \end{pmatrix}$$

where

$$\sigma_1 = -l^2\,m^2\cos(x_2)^2 + l^2\,m^2 + M\,l^2\,m + \mathrm{I}\,m + \mathrm{I}\,M$$

Part 2. Linearization

Part 2.1 Substitute Real Values

Then the real values are substituted for the symbolic variables.

```
SS = subs(StSp, [  M,   m,   b,   l,     I,    g], ...
                [0.5, 0.2, 0.1, 0.3, 0.006, 9.81]);

% Number of Decimal Points Precision
precision = 3;
SS = vpa(SS, precision)
```

```
SS =
```

$$\begin{pmatrix} x_3 \\ x_4 \\ -\dfrac{1.0\,(0.024\,U - 0.0024\,x_3 + 0.0177\sin(2.0\,x_2) + 0.00144\,x_4\sin(x_2))}{0.0036\cos(x_2)^2 - 0.0168} \\ \dfrac{0.06\,(6.87\sin(x_2) + U\cos(x_2) - 0.1\,x_3\cos(x_2) + 0.06\,x_4\cos(x_2)\sin(x_2))}{0.0036\cos(x_2)^2 - 0.0168} \end{pmatrix}$$

Part 2.2 Linearize Model and Export as Linear State Space Format

Using the first order Taylors Series Expantion, the system in linearized around a working point that is considered as all zeros.

```
% Working/Equilibrium Point
x_eq = [0; 0; 0; 0];
U_eq = 0;
```

```
% A and B Matrices Creation
A = jacobian(SS, x);
B = jacobian(SS, U);

% Substitute Equivalent Values for A and B
A = subs(A, [x; U], [x_eq; U_eq]); A = double(A);
B = subs(B, [x; U], [x_eq; U_eq]); B = double(B);

C = [1, 0, 0, 0
     0, 1, 0, 0];
D = [0; 0];

% System Order
n = size(A, 1);

% Final State Space Form
states = {'X', 'Phi', 'dX', 'dPhi'};
inputs = {'U'};
outputs = {'Phi'};
Sys = ss(A, B, C, D, 'statename', states, 'inputname', inputs, 'outputname',
outputs)
```

```
Sys =

  A =
              X      Phi       dX     dPhi
   X          0        0        1        0
   Phi        0        0        0        1
   dX         0    2.675  -0.1818        0
   dPhi       0   -31.21   0.4545        0

  B =
             U
   X         0
   Phi       0
   dX    1.818
   dPhi -4.545

  C =
             X   Phi    dX  dPhi
   Phi(1)    1     0     0     0
   Phi(2)    0     1     0     0

  D =
          U
   Phi(1)  0
   Phi(2)  0

Continuous-time state-space model.
Model Properties
```

Part 2.3 Creation of Some Realizations

To create the realizations, a orthonormal matrix is created as a random modal matrix and a new system is created based on that.

```
RealizationNo = 4;
```

```matlab
for R = 1:RealizationNo
    % Create a Random Orthonormal Modal Matrix
    Q = orth(rand(n));

    % Create a New Realization
    Anew = inv(Q) * A * Q;
    Bnew = inv(Q) * B;
    Cnew = C * Q;
    Dnew = D;
    SysNew = ss(Anew, Bnew, Cnew, Dnew);

    % Print information about the new realization
    disp('=====================================================================')
    fprintf('System Realization %d:\n', R);

    % Print A, B, C, and D separately
    fprintf('A:\n');
    disp(SysNew.A);

    fprintf('\nB:\n');
    disp(SysNew.B);

    fprintf('\nC:\n');
    disp(SysNew.C);

    fprintf('\nD:\n');
    disp(SysNew.D);

    fprintf('\n');
end
```

```
=====================================================================
System Realization 1:
A:
   -8.4494  -10.4088    1.4311    8.3062
   13.7196   14.6139   -1.7025  -12.7620
   -2.3586   -2.2426   -0.2654    1.2116
    6.8499    8.2883   -0.7922   -6.0808
B:
    1.6301
   -3.7035
   -0.3172
   -2.7373
C:
   -0.3915    0.0808    0.8063   -0.4359
   -0.5705   -0.6369    0.0650    0.5145
D:
     0
     0
=====================================================================
System Realization 2:
A:
   -7.4434    2.1060   13.4740    1.6341
    2.9927   -1.2237   -5.5875   -0.8422
```

```
   -7.1558    1.8623   11.0217    2.2884
   11.7153   -1.9981  -19.1882   -2.5364
B:
    1.6825
   -2.1951
    1.7907
   -3.6210
C:
   -0.5294    0.4518   -0.2831   -0.6599
   -0.5132    0.1002    0.8441    0.1183
D:
     0
     0
======================================================================
System Realization 3:
A:
   -6.6023   -6.9590    5.6393   13.2169
    6.1525    5.6735   -4.5680  -12.2740
   -8.6393   -8.1623    5.9790   14.6760
    2.4815    3.2305   -2.6945   -5.2320
B:
    1.6785
   -3.0154
    2.8972
   -1.9140
C:
   -0.3496    0.5908    0.7080   -0.1656
   -0.4144   -0.4081    0.3117    0.7513
D:
     0
     0
======================================================================
System Realization 4:
A:
   -0.3253   -1.8100   -0.0688    0.2002
    4.1428    8.2906    2.0884   -3.3833
   -1.1723   -0.9263   -0.7315   -0.0881
   11.8915   25.3858    5.9112   -7.4156
B:
   -0.7529
   -0.9126
   -0.5426
   -4.7194
C:
   -0.5655    0.0684    0.8217   -0.0175
   -0.4137   -0.8490   -0.2086    0.2541
D:
     0
     0
```

Part 3. Control

Part 3.1 Controllability Check

```
% Controllability Matrix Creation
CtrbMat = ctrb(Sys);

% Controllability Check
if rank(CtrbMat) == n
    disp(' >> The system is controllable.');
    disp('Controllability Matrix:');
```

```
    disp(CtrbMat);
else
    disp(' >> The system is not controllable.');
    disp('Controllability Matrix:');
    disp(CtrbMat);
end
```

```
 >> The system is controllable.
Controllability Matrix:
         0    1.8182   -0.3306  -12.1011
         0   -4.5455    0.8264  141.7299
    1.8182   -0.3306  -12.1011    4.4113
   -4.5455    0.8264  141.7299  -31.2969
```

Part 3.2 Observability Check

```
% Observability Matrix Creation
ObsvMat = obsv(Sys);

% Observability Check
if rank(ObsvMat) == n
    disp('The system is observable.');
    disp('Observability Matrix:');
    disp(ObsvMat);
else
    disp('The system is not observable.');
    disp('Observability Matrix:');
    disp(ObsvMat);
end
```

```
The system is observable.
Observability Matrix:
    1.0000         0         0         0
         0    1.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000
         0    2.6755   -0.1818         0
         0  -31.2136    0.4545         0
         0   -0.4864    0.0331    2.6755
         0    1.2161   -0.0826  -31.2136
```

Part 3.3 Controller and Observer Design

```
% State Feedback Controller Design
ContollerPoles = [-1-2j, -1+2j, -2-1j, -2+1j];
K = place(A, B, ContollerPoles);

% State Feedback Controller Design
ObserverPoles = [-1-2j, -1+2j, -2-1j, -2+1j];
L = place(A', C', ObserverPoles)';

% Display Gain Matrices
```

```
disp(['State Feedback Gain (K): ' mat2str(K, 2)]), disp(['State Observer Gain (L):
' mat2str(L, 2)])
```

```
State Feedback Gain (K): [0.56 3.1 0.57 -1.1]
State Observer Gain (L): [2.9 0.81;-0.7 3;3.3 5.4;-1.8 -27]
```

```
% Calculate the Augmented System Char Matrices
Aaug = [  A,       -B*K
        L*C, A-B*K-L*C];

Baug = [B
        B];

Caug = [C -D*K];

Daug=D;

% System Object Creation
SysAug = ss(Aaug, Baug, Caug, Daug);

% Time vector for simulation
t = 0:0.01:20;

% Initial States
% x0 = [pi, 0.1, 0, 0, 0.2, 0, 0, 0];
x0 = rand(1, 8)*3;

% Simulate the closed-loop system
[y, t, x] = lsim(SysAug, zeros(size(t)), t, x0);

% Plot the output
figure;
subplot(2, 2, [1, 2]);
plot(t, y, 'LineWidth', 2);
title('Closed-Loop System Output');
xlabel('Time');
ylabel('Output');
legend({'Output $X$', 'Output $\phi$'}, 'Interpreter', 'latex');
grid on

% Plot all states
subplot(2, 2, 3);
plot(t, x);
title('System States');
xlabel('Time');
ylabel('States');
legend({'$X$', '$\phi$', '$\dot{X} $', '$\dot{\phi}$', ...
        'Est. $X$', 'Est. $\phi$', 'Est. $\dot{X} $', 'Est. $\dot{\phi}$'}, ...
        'Interpreter', 'latex', 'NumColumns', 2);
ylim([-4, 8])
grid on
```

```
% Estimation Error
subplot(2, 2, 4);
Err = x(:, 4) - x(:, 5:end);
plot(t, Err);
title('Estimation Errors');
xlabel('Time');
ylabel('States');
legend({'$\hat{X}$', '$\hat{\phi}$', '$\hat{\dot{X}}$', '$\hat{\dot{\phi}}$'},...
        'Interpreter', 'latex', ...
        'NumColumns', 2);
ylim([-4, 8])
grid on

% Adjust subplot layout
sgtitle('Closed-Loop System Simulation');
```