Set Based Estimator Testing

Setup

```
clear
close all
```

Simulation Settings

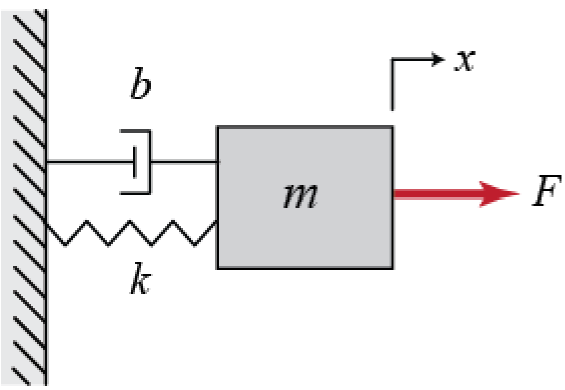
```
tf = 0.2;
dt = 0.01;
tspan = 0:dt:tf;
```

Input

```
omegaInput = 5;
inputFunction = @(t) sin(omegaInput * t);
U = inputFunction(tspan);
```

System Setup

```
sysType = "simpleDTsys"; %"springMassDamper" "simpleDTsys"

switch sysType
    case {"springMassDamper"}
```



$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F(t)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

```
        m = 1;
        k = 1;
        b = 1;
        A_ct = [0, 1;
            -k/m, -b/m];
        B_ct = [0;
            -1/m];
```

```
        C = [1, 0];
        D = [0];
        sys_ct = ss(A_ct,B_ct,C,D)

        A = expm(A_ct * dt);
        B = A * B_ct; %assume small and just disturbance anyway
        sys = ss(A,B,C,D,dt);

    case {"simpleDTsys"}
%         A = diag([0.5 -0.5])
%         * [
%             0.9371    0.8491
%             0.8295    0.3725];
%         A = [
%             0.5 -0.5;
%             -0.5 -0.5]
        A = 0.7 .* [
            1   -1
            1   1
            ];
        B = [
            1
            0];
        C = eye(2);
        D = ones(2,1);
end
 [numStates, numInputs] = size(B);
 numOutputs = size(C,1);
```

Estimator Setup

Nominal

$$\widehat{x}_k = (A + LC)\widehat{x}_{k-1} + Bu_{k-1} + Ly_{k-1}$$

```
lambda_obsv = [-1, -1.5];
L = place(A',C',lambda_obsv).';
```

Simulation

```
% Setup
x_0 = 10 * rand(2,1);
x_hat_0 = x_0 + 0.1 * rand(2,1);

% Initialization
x = x_0;
x_hat = x_hat_0;

% Evolution
for k = 1:length(tspan)
    % Inputs and Measurements
```

```
    u = U(:,k);
    y = C * x + D * u;
    y_hat = C * x_hat;

    % State Equations
%     dx = A * x + B * u;
%     dx_hat = A * x_hat + B * u + L*(y - y_hat);

    % Euler Update
%     x = x + dx * dt;
%     x_hat = x_hat + dx_hat * dt;
    x = A * x + B * u;
    x_hat = A * x_hat + B * u + L * (y - y_hat);

    % Store Values
%     X_data(:,k) = x;
%     X_hat_data(:,k) = x_hat;
%     Y_data(:,k) = y;
%     Y_hat_data(:,k) = y_hat;
end
```

Set Based

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}_w \mathbf{w}_{k-1}, \qquad \mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}_v \mathbf{v}_k, \qquad (31)$$

$$\hat{X}_k = (\mathbf{A}\hat{X}_{k-1} + \mathbf{B}_w W) \cap_{\mathbf{C}} (\mathbf{y}_k - \mathbf{D}_v V),$$

with $\hat{X}_0 = X_0 \cap_{\mathbf{C}} (\mathbf{y}_0 - \mathbf{D}_v V)$. In genera

assume that essentially the bound is what we call the x_hat... (the idea of enclosue vs exact definition)

$$\mathcal{O}_k \supset (\mathbf{A}\mathcal{O}_{k-1} + \mathbf{B}_w W) \cap_{\mathbf{C}} (\mathbf{y}_k - \mathbf{D}_v V),$$

with $\mathcal{O}_0 \supset X_0 \cap_{\mathbf{C}} (\mathbf{y}_0 - \mathbf{D}_v V)$. ; $\mathcal{O}_k \supset \hat{X}_k$

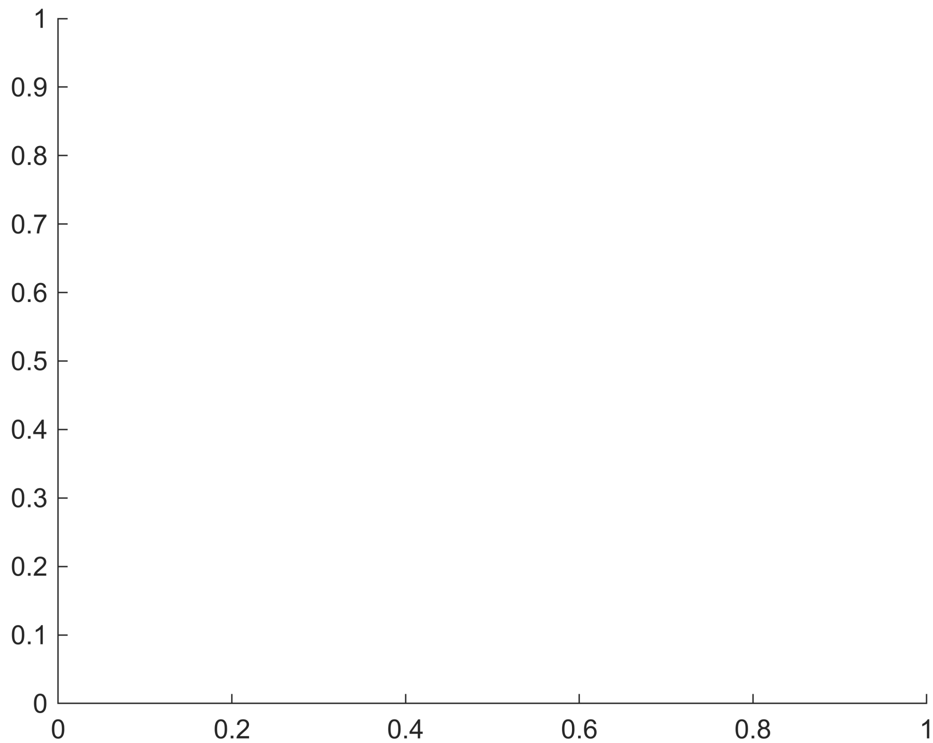
the intersection definition (using the generalizedIntersection() function adds a constraint demension each timestep:

$$Z \cap_{\mathbf{R}} Y = \left\{ [\mathbf{G}_z\ \mathbf{0}], \mathbf{c}_z, \begin{bmatrix} \mathbf{A}_z & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_y \\ \mathbf{R}\mathbf{G}_z & -\mathbf{G}_y \end{bmatrix}, \begin{bmatrix} \mathbf{b}_z \\ \mathbf{b}_y \\ \mathbf{c}_y - \mathbf{R}\mathbf{c}_z \end{bmatrix} \right\}$$

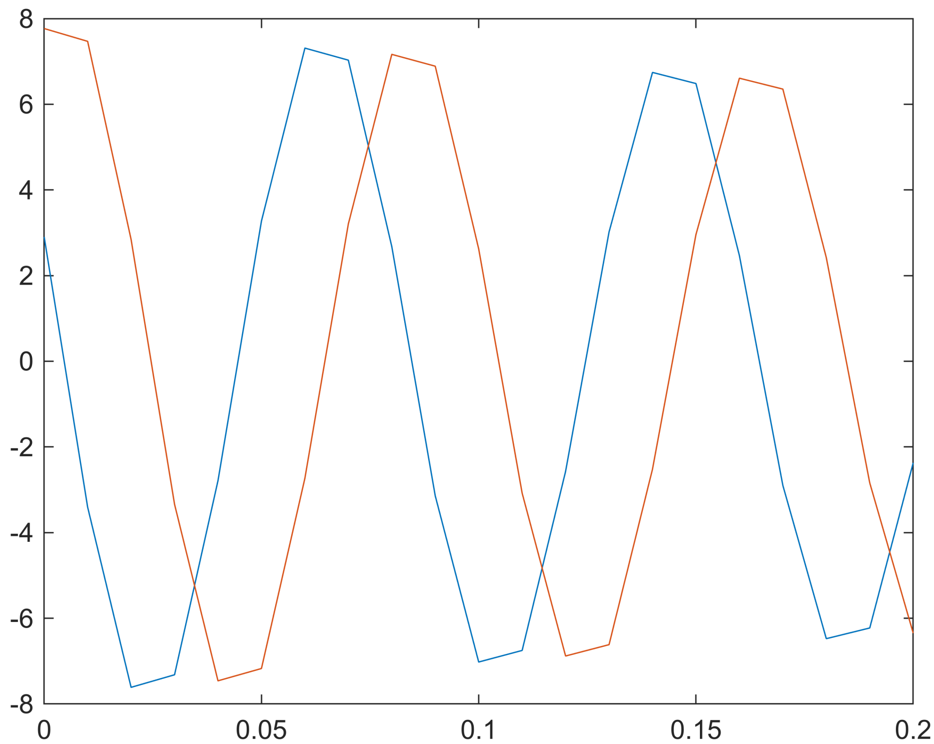
```
% error bound scale
w_0 = 0.0001;
v_0 = 0.1 .* ones(numInputs,1);
```

```
x_hat_0 = 0.1;

% error zonotopes
W_0 = conZono(0, w_0, [], []);
V_0 = conZono(zeros(size(numOutputs,1)), v_0, [], []);
X_hat_0 = conZono(x_hat, x_hat_0 * eye(size(x_0,1)), [], []);

% plot setup
fig = figure;
ax = axes;


% Evolution
x = x_0;
x_hat = x_0 .* (1 + 0.1 * (2*rand(size(x_0))-1));
X_hat = X_hat_0;
W = W_0;
V = V_0;
for k = 1:length(tspan)
    % Inputs and Measurements
    u = w_0 .* (2 * rand(size(w_0)) - 1);
    y = C * x + D * u + v_0 .* (2 * rand(size(v_0)) - 1);

    % System Evolution
    x = A * x + B * u;
    X_hat = generalizedIntersection(A * X_hat + B * W, (D * V + -y), C);

    % Save Data
    X_data(:,k) = x;
    Y_data(:,k) = y;
    U_data(:,k) = u;
    X_hat_data{k} = X_hat;



    % Plot
    hold off
    X_hat.plot
    hold on
%     plot(x(2),x(1))
%     axis([-1 1 -1 1])
    M(k) = getframe(gcf);
end
close gcf
```

```
figure
movie(M)
```




```
figure
plot([tspan' tspan'], X_data')
```



test

```
X_hat_0
```

```
X_hat_0 =
  conZono with properties:

        c: [2×1 double]
        G: [2×2 double]
        A: [0×2 double]
        b: []
        n: 2
       nG: 2
       nC: 0
    order: 1
```

```
X_hat_0.c
```

```
ans = 2×1
    8.2572
    3.2404
```

```
X_hat_0.G
```

```
ans = 2×2
    0.1000         0
         0    0.1000
```

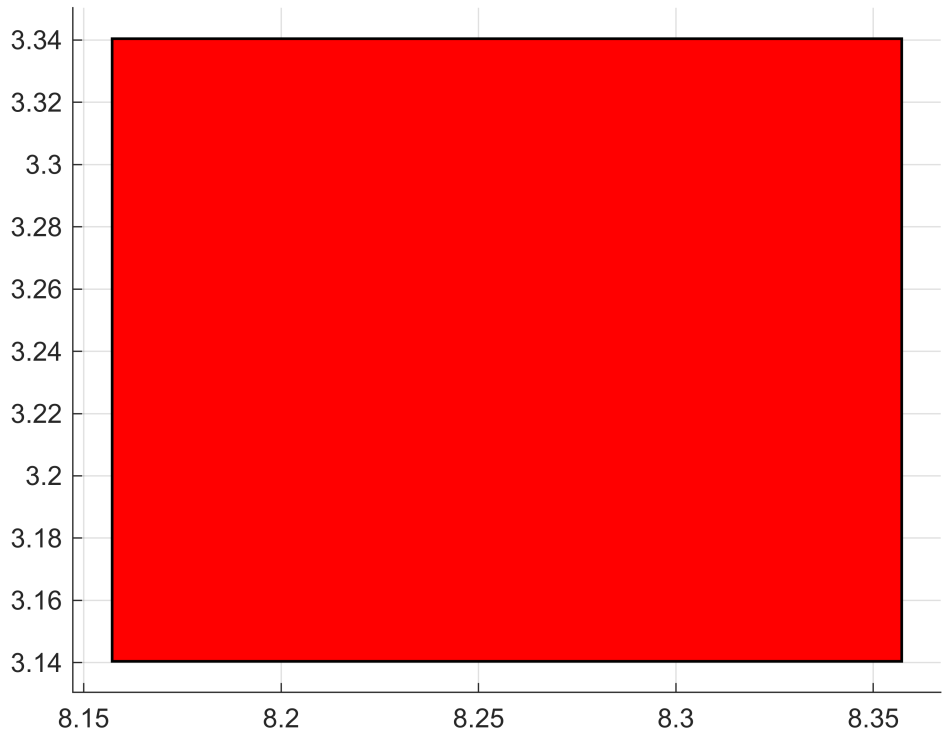
```
X_hat_0.A
```

```
ans =

  0×2 empty double matrix
```

```
X_hat_0.bD
```

```
ans =

     []
```

```
X_hat_0.plot
```




```
X_hat_data{1}
```

```
ans = 
  conZono with properties:

        c: [2×1 double]
        G: [2×4 double]
        A: [2×4 double]
        b: [2×1 double]
        n: 2
       nG: 4
       nC: 2
    order: 1
```

```
X_hat_data{1}.c
```

```
ans = 2×1
    3.5118
    8.0483
```

```
X_hat_data{1}.G
```

```
ans = 2×4
    0.0700   -0.0700    0.0001         0
    0.0700    0.0700         0         0
```

```
X_hat_data{1}.A
```

```
ans = 2×4
    0.0700   -0.0700    0.0001   -0.1000
    0.0700    0.0700         0   -0.1000
```

```
X_hat_data{1}.b
```

```
ans = 2×1
  -11.0464
  -11.4375
```

```
X_hat_data{1}.plot
```