ME C231B Assignment: Kalman Filtering (part 3)

Table of Contents

2.a

```
delta = 0.1;
A = [1,delta,0;0,1,delta;0,0,1];
E = zeros(3,3); E(3,1) = 1;
C = [1,0,0;0,0,1];
F = [0,1,0;0,0,1];
TS = -1;
W = diag([0.3,1,1]);
[K,L,H,G,nIter,estErrVar] = formSteadyStateKF(A,E,C,F,W,TS);
fprintf('The steady-state Kalman filter gain L is: \n');
disp(L);
fprintf('The steady-state Covariance is: \n');
disp(estErrVar);
```

2.d

```
delta = 0.1;
A = [1,delta,0;0,1,delta;0,0,1];
E = zeros(3,3); E(3,1) = 1;
C = [1,0,0;0,0,1];
F = [0,1,0;0,0,1];
W = diag([0.3,1,1]);

N = 200;
nX = 3;
nW = 3;
nY = 2;
arrayA = repmat(A,[1,1,N]);  % nX-by-nX-by-N
arrayE = repmat(E,[1,1,N]);  % nX-by-nW-by-N
arrayC = repmat(C,[1,1,N]);  % nY-by-nX-by-N
```

```
arrayF = repmat(F,[1,1,N]);  % nY-by-nW-by-N
arraySW = repmat(W,[1,1,N]); % nW-by-nW-by-N
Sx0 = eye(nX); % nX-by-nX
m0 = [0;0;0];      % nX-by-1
Sxii1 = Sx0;
xii1 = m0;
NumSimulation = 600;
errorlist_v2 = zeros(NumSimulation,1);
for step = 1:NumSimulation
    wSeq = randn(3,N); % nW-by-1-by-N
    wSeq(1,:) = wSeq(1,:)*0.3;
    x0 = [0;0;0];      % nX-by-1
    % arrayR =   % nR-by-1-by-N
    emptyB = [];  % this template file is for the case of no control
 signal
    emptyu = [];  % this template file is for the case of no control
 signal
    y = zeros(nY,N);
    xSeq = zeros(nX,N);
    xSeq(:,1) = x0;
    xiiSeq = zeros(nX,N);
    for i=0:N-1
        iMatlab = i+1;
        Ai = arrayA(:,:,iMatlab);
        Ei = arrayE(:,:,iMatlab);
        Ci = arrayC(:,:,iMatlab);
        Fi = arrayF(:,:,iMatlab);
        Swi = arraySW(:,:,iMatlab);
        wi = wSeq(:,iMatlab);
        xi = xSeq(:,iMatlab);
        % Get y(i) from system model, using x(i) and w(i)
        y(:,iMatlab) = Ci*xi + Fi*wi;
        % Get estimates of x(i+1|i) using u(i) and y(i) from KF
        [xi1i,Sxi1i,xii,Sxii,Syii1,Lk,Hk,Gk,wkk] = ...

 KF231B(xii1,Sxii1,Ai,emptyB,Ci,Ei,Fi,Swi,emptyu,y(:,iMatlab));
        % Save the state estimate xHat_{i|i}
        xiiSeq(:,iMatlab) = xii;
        % Other estimates can be saved too.  KF231B returns additional
 estimates
        % (for example, wkk), and saving those sequences could be
 useful too,
        % depending on the computational exercise.
        %
        % Get x(i+1) from system model, using x(i), u(i) and w(i)
        xSeq(:,iMatlab+1) = Ai*xi + Ei*wi;
        % Shift the error-variance estimate so that when loop-index i
 advances,
        % the initial condition for this variance is correct.
        Sxii1 = Sxi1i;
        xii1 = xi1i;
        error_v2 = xii1(2)-xi(2);
    end
    errorlist_v2(step) = abs(error_v2);
```

```
end
fprintf('Probability of greater than 0.75: %f',sum(errorlist_v2>0.73)/
NumSimulation);
```

3.a

The system is stable

```
nX = 4;
nY = 3;
nU = 2;
H = drss(nX,nY,nU);
if max(abs(eig(H.A)))>0.999; H.A = 0.99*H.A; end
isstable(H)
```

3.b

```
% Create function_handle which returns integrand (matrix-valued)
% as a function of OMEGA. This uses the command FREQRESP
% which calculates the frequency-response at a single, given
 frequency.
f = @(Omega) freqresp(H,Omega)*freqresp(H,Omega)';
% Call INTEGRAL, specifying limits of integration as 0 to 2*PI.  Make
% sure the command INTEGRAL knows that the integrand is not
% a scalar, but is array-valued (in our case, nY-by-nY)
Int = 1/(2*pi) * integral(f,0,2*pi,'ArrayValued',true')
% Note that although the integrand is a complex, hermitian
% matrix, the overall integral is purely real.  This can be proven
% but we will not focus on that here.  Some small numerical
% values for imaginary parts creep in due to roundoff.  Hence,
% take the real-part of the computed answer to clean up the result.
Int = real(Int)
```

3.c

```
sqrt(trace(Int))
norm(H,2)
```

3.d

```
% Decide on the duration (in time) of each input/output sequences.
% The formula has k -> INF, but we can only simulate for a finite
% duration. In general, choose a duration "long" compared to the
% time-constants of the system. Here, we simply pick a duration
% of 200 steps.
Nsteps = 200;

% So each instance (associated with a point in the sample space)
% of the u-sequence is an nU-by-200 array. The convention in
% Matlab’s simulation codes (lsim, Simulink, etc) is to
% represent this as a 200-by-nU array.
% Decide on the dimension of the sample-space. Take 2000 outcomes,
NsampleSpace = 2000;

% Hence, we need 2000, 200-by-nU arrays, as generated by RANDN.
% Create a 200-by-nU-by-2000 array with randn.
U = randn(Nsteps,nU,NsampleSpace);

% Initialize a nY-by-nY matrix to hold the expectation
% of the final value (of each simulation) of y*y^T.
E_y_yT = zeros(nY,nY);

% Our construction (from RANDN) had all outcomes equally likely,
% with probability equal to 1/NsampleSpace. So, create a
% constant value for each individual outcome's probability
p = 1/NsampleSpace;

% Use a FOR loop to compute the response for all the
% sample-space outcomes
for i=1:NsampleSpace
    % Use LSIM to compute each response.
    % Recall that U(:,:,i) is the 200-by-nU array (for input
    % sequence) associated with the i'th sample point.
    Y = lsim(H,U(:,:,i));
    % By convention (LSIM), the matrix Y will be Nsteps-by-nY.
    % The last value (our approximation to k->INF) is simply the
    % last row of Y.
    y_end  = Y(end,:)';
    E_y_yT = E_y_yT + p*(y_end*y_end');
end
% Compare to limiting expected value to the integral
E_y_yT
Int
```

3.e

```
covar(H,eye(nU));
Int % exactly matches output of COVAR
E_y_yT % approximately matches output of COVAR
```

4.c

```
efkIterationScript;
```

5.a

General intuition in assisting the verification of the realization formulation: State evolution:

```
xG(k+1) = AG*xG(k) + BG*q;

xP(k+1) = AP*xP(k) + BP*u; u = CG*xG(k) + DG*q;

xP(k+1) = AP*xP(k) + BP*CG*xG(k) + BP*DG*q;
```

```
xK(k+1) = AK*xK(k) + BK*u; u = CP*xP(k) + DP*(CG*xG(k) + DG*q);

xK(k+1) = AK*xK(k+1) + BK*CP*xP(k) + BK*DP*CG*xG(k) + BK*DP*DG*q;
Output:
e(k) = xP(k) - xhatP(k); xhatP(k) = M*CK*xK(k) + M*DK*u; u = CP*xP(k)
 + DP*CG*xG(k)

e(k) = xP(k) - M*CK*xK(k) - M*DK*CP*xP(k) - M*DK*DP*CG*xG(k);

e(k) = (I - M*DK*CP)*xP(k) -M*DK*DP*CG*xG(k) -M*CK*xK(k);
```

5.b

Please refer to function IC = kfBuildIC(G,P,K,M)

Published with MATLAB® R2017b