AOE 5784 Exam #2 Solutions

Fall 2024

1) Problem 4-7 in Bar-Shalom (15 pts):

i.)

$$\underline{x}(k) = \left(\prod_{j=0}^{k-\ell-1} F\right)\underline{x}(\ell)$$

$$+ \sum_{i=\ell}^{k-1}\left\{\prod_{j=0}^{k-i-2} F\right\}\left[G\underline{u}(i) + \Gamma v(i)\right]$$

Added to make two parts consistent even though Bar-Shalom is not consistent

or

$$\underline{x}(k) = \left\{F^{(k-\ell)}\right\}\underline{x}(\ell) + \sum_{i=\ell}^{k-1}\left\{F^{(k-i-1)}\right\}\left[G\underline{u}(i) + \Gamma v(i)\right]$$

ii.) $$\underline{x}(k) - \bar{\underline{x}}(k) = \left\{F^{(k-\ell)}\right\}\left(\underline{x}(\ell) - \bar{\underline{x}}(\ell)\right)$$

$$+ \sum_{i=\ell}^{k-1}\left\{F^{(k-i-1)}\right\}\Gamma\left[\underline{v}(i) - \bar{\underline{v}}(i)\right]$$

$$P(k) = E\left\{\left[\underline{x}(k) - \bar{\underline{x}}(k)\right]\left[\underline{x}(k) - \bar{\underline{x}}(k)\right]^T\right\}$$

$$= \left\{F^{(k-\ell)}\right\}P(\ell)\left\{F^{(k-\ell)}\right\}^T$$

$$+ \sum_{i=\ell}^{k-1}\left\{F^{(k-i-1)}\right\}\Gamma Q \Gamma^T\left\{F^{(k-i-1)}\right\}^T$$

because $$E\left\{\left[\underline{x}(\ell) - \bar{\underline{x}}(\ell)\right]\left[\underline{v}(\ell) - \bar{\underline{v}}(\ell)\right]^T\right\} = 0$$

and because

$$E\{[\underline{v}(j)-\bar{\underline{v}}(j)][\underline{v}(\ell)-\bar{\underline{v}}(\ell)]^T\} = \delta_{j\ell}\, Q$$

2) Problem Set 5, #2 (15 Pts):

$$\hat{\underline{x}}(k/k{+}1)-\hat{\underline{x}}(k) = F^{-1}(k)\Big[\hat{\underline{x}}(k{+}1) - G(k)\underline{u}(k) - \Gamma(k)\hat{\underline{v}}(k)\Big] - \hat{\underline{x}}(k)$$

but $\hat{\underline{v}}(k) = Q(k)\Gamma^T(k)\bar{P}^{-1}(k{+}1)\left[\hat{\underline{x}}(k{+}1)-\bar{\underline{x}}(k{+}1)\right]$

from MAP derivation of KF and recognizing that $\bar{P}(k{+}1) = F(k)P(k)F^T(k) + \Gamma(k)Q(k)\Gamma^T(k)$

but $\hat{\underline{x}}(k{+}1)-\bar{\underline{x}}(k{+}1) = W(k{+}1)\underline{\nu}(k{+}1)$
$= \bar{P}(k{+}1)H^T(k{+}1)S^{-1}(k{+}1)\underline{\nu}(k{+}1)$

So $\hat{\underline{v}}(k) = Q(k)\Gamma^T(k)H^T(k{+}1)S^{-1}(k{+}1)\underline{\nu}(k{+}1)$

and

$$\hat{\underline{x}}(k/k{+}1)-\hat{\underline{x}}(k) = F^{-1}(k)\Big[\bar{\underline{x}}(k{+}1) + \bar{P}(k{+}1)H^T(k{+}1)S^{-1}(k{+}1)\underline{\nu}(k{+}1) - G(k)\underline{u}(k) - \Gamma(k)Q(k)\Gamma^T(k)H^T(k{+}1)S^{-1}(k{+}1)\underline{\nu}(k{+}1)\Big] - \hat{\underline{x}}(k)$$

but $\bar{\underline{x}}(k{+}1) = F(k)\hat{\underline{x}}(k) + G(k)\underline{u}(k)$

Therefore $F^{-1}(k)\left[\underline{x}(k+1) - G(k)\underline{u}(k)\right]$
$- \underline{x}(k) = 0$

So

$$\underline{\hat{x}}(k|k+1) - \underline{\hat{x}}(k) = F^{-1}(k)\left[\bar{P}(k+1) - \Gamma(k)Q(k)\Gamma^T(k)\right] \cdot$$
$$H^T(k+1)\,S^{-1}(k+1)\,\underline{\nu}(k+1)$$

but $\bar{P}(k+1) - \Gamma(k)Q(k)\Gamma^T(k) = F(k)P(k)F^T(k)$

So

$$\underline{\hat{x}}(k|k+1) - \underline{\hat{x}}(k) = P(k)F^T(k)H^T(k+1)S^{-1}(k+1)\underline{\nu}(k+1)$$

$$\underline{z}(k+1) - H(k+1)\underline{\hat{x}}(k+1)$$
$$= \underline{z}(k+1) - H(k+1)\left[\underline{\bar{x}}(k+1) + \bar{P}(k+1)H^T(k+1)S^{-1}(k+1)\underline{\nu}(k+1)\right]$$
$$= \underline{\nu}(k+1) - H(k+1)\bar{P}(k+1)H^T(k+1)S^{-1}(k+1)\underline{\nu}(k+1)$$
$$= \left[S(k+1) - H(k+1)\bar{P}(k+1)H^T(k+1)\right]S^{-1}(k+1)\,\underline{\nu}(k+1)$$

But $S(k+1) = H(k+1)\bar{P}(k+1)H^T(k+1) + R(k+1)$

So

$$\underline{z}(k+1) - H(k+1)\underline{\hat{x}}(k+1) = R(k+1)\,S^{-1}(k+1)\,\underline{\nu}(k+1)$$

Now $J_b[\underline{v}(k), \hat{x}(k+1), k]$

$$= J_a\{\hat{x}(k/k+1), \hat{v}(k), \hat{x}(k+1), k\}$$

$$= \frac{1}{2}[\hat{x}(k/k+1) - \hat{x}(k)]^T P^{-1}(k)[\hat{x}(k/k+1) - \hat{x}(k)]$$

$$+ \frac{1}{2}\hat{v}^T(k)\, Q^{-1}(k)\, \hat{v}(k)$$

$$+ \frac{1}{2}[\underline{z}(k+1) - H(k+1)\hat{x}(k+1)]^T R^{-1}(k+1)[z(k+1) - H(k+1)\hat{x}(k+1)]$$

or, using the 3 expressions that have been derived:

$$J_b\{\hat{v}(k), \hat{x}(k+1), k\} =$$

$$\frac{1}{2}\underline{\nu}^T(k+1)\{S^{-1}(k+1)H(k+1)F(k)P(k)P^{-1}(k)P(k)F^T(k)H^T(k+1)S^{-1}(k+1)$$

$$+ S^{-1}(k+1)H(k+1)\Gamma(k)Q(k)Q^{-1}(k)Q(k)\Gamma^T(k)H^T(k+1)S^{-1}(k+1)$$

$$+ S^{-1}(k+1)R(k+1)R^{-1}(k+1)R(k+1)S^{-1}(k+1)\}\underline{\nu}(k+1)$$

$$= \frac{1}{2}\underline{\nu}^T(k+1)S^{-1}(k+1)\{H(k+1)[F(k)P(k)F^T(k) + \Gamma(k)Q(k)\Gamma^T(k)]\cdot H^T(k+1)$$

$$+ R(k+1)\}S^{-1}(k+1)\underline{\nu}(k+1)$$

Note $S(k+1) = S^T(k+1)$ and $R(k+1) = R^T(k+1)$

or

$$J_6\left[\underline{x}(k), \underline{\delta}(k+1), k\right] =$$

$$\frac{1}{2}\underline{\nu}^T(k+1)\, S^{-1}(k+1)\left\{ H(k+1)\ \bar{P}(k+1)\, H^T(k+1) + R(k+1)\right\} S^{-1}(k+1)\,\underline{\nu}(k+1)$$

$$= \frac{1}{2}\underline{\nu}^T(k+1)\, S^{-1}(k+1)\, S(k+1)\, S^{-1}(k+1)\,\underline{\nu}(k+1)$$

$$= \frac{1}{2}\underline{\nu}^T(k+1)\ S^{-1}(k+1)\,\underline{\nu}(k+1)$$

3) Problem Set 5, #6, with modified $Q = 6$, $R = 0.05$ (20 pts):

See the print-outs that follow

```
function [xtruehist,zhist] = kf_truthmodel(F,Gamma,H,Q,R,xhat0,...
                                          P0,kmax)
%
%  Copyright (c) 2005 Mark L. Psiaki.  All rights reserved.
%
%  This function performs a truth-model Monte-Carlo simulation for
%  the discrete-time stochastic system model:
%
%     x(k+1)  = F*x(k) + Gamma*v(k)
%     z(k)    = H*x(k) + w(k)
%
%  Where v(k) and w(k) are uncorrelated, zero-mean, white-noise
%  Gaussian random processes with covariances E[v(k)*v(k)'] = Q and
%  E[w(k)*w(k)'] = R.  The simulation starts from a true x(0)
%  that is drawn from the Gaussian distribution with mean xhat0
%  and covariance P0.  The simulation starts at time k = 0 and
%  lasts until time k = kmax.
%
%
%  Inputs:
%
%    F           The (nx)x(nx) state transition matrix for the time-
%                invariant linear system.
%
%    Gamma       The (nx)x(nv) process noise gain matrix for the time-
%                invariant linear system.
%
%    H           The (nz)x(nx) output gain matrix for the time-
%                invariant linear system.
%
%    Q           The (nv)x(nv) symmetric positive definite process noise
%                covariance matrix.
%
%    R           The (nz)x(nz) symmetric positive definite measurement
%                noise covariance matrix.
%
%    xhat0       The (nx)x1 initial state estimate.
%
%    P0          The (nx)x(nx) symmetric positive definite initial state
%                estimation error covariance matrix.
%
%    kmax        The maximum discrete-time index of the simulation.
%
%  Outputs:
%
%    xtruehist   = [x(0)';x(1)';x(2)';...;x(kmax)'], the (kmax+1)-by-(nx)
%                array that stores the truth time history for the state
%                vector.  Note that Matlab does not allow zero indices
%                in its arrays.  Thus, x(0) = xtruehist(1,:)',
%                x(1) = xtruehist(2,:)', etc.
%
%    zhist       = [z(1)';z(2)';z(3)';...;z(kmax)'], the (kmax)-by-(nz)
%                array that stores the measurement time history.
%                z(1) = zhist(1,:)', z(2) = zhist(2,:)', etc.  Note
%                that the state vector xtruehist(j+1,:)' and
%                the measurement vector zhist(j,:)' correspond
%                to the same time.
```

```
%

%
%  Get problem dimensions and set up the output arrays.
%
   [nx,nv] = size(Gamma);
   nz = size(H,1);
   xtruehist = zeros((kmax+1),nx);
   zhist = zeros(kmax,nz);
%
%  Calculate the appropriate matrix square root of P0 for use in
%  randomly generating an initial state.  P0 = P0_rt*(P0_rt');
%
   P0_rt = chol(P0)';
%
%  Generate the initial truth state with the aid of a random number
%  generator and store it.
%
   x0 = xhat0 + P0_rt*randn(nx,1);
   xtruehist(1,:) = x0';
%
%  Calculate the appropriate matrix square roots of Q and R for use in
%  randomly generating the process and measurement noise vectors.
%  Q = Q_rt*(Q_rt') and R = R_rt*(R_rt').
%
   Q_rt = chol(Q)';
   R_rt = chol(R)';
%
%  Iterate through the kmax samples, propagating the state with
%  randomly generated process noise of the correct covariance and
%  corrupting the measurements with randomly generated measurement
%  noise of the correct covariance.
%
   xk = x0;
   for k = 1:kmax
      xkm1 = xk;
      vkm1 = Q_rt*randn(nv,1);
      xk = F*xkm1 + Gamma*vkm1;
      wk = R_rt*randn(nz,1);
      zk = H*xk + wk;
      kp1 = k + 1;
      xtruehist(kp1,:) = xk';
      zhist(k,:) = zk';
   end
```

```
function [xhathist,Phist] = kf_solution(F,Gamma,H,Q,R,xhat0,...
                                        P0,zhist)
%
%  Copyright (c) 2005 Mark L. Psiaki.  All rights reserved.
%
%  This function performs Kalman filter state estimation for
%  the dynamic system model:
%
%     x(k+1) = F*x(k) + Gamma*v(k)
%     z(k)   = H*x(k) + w(k)
%
%  Where v(k) and w(k) are uncorrelated, zero-mean, white-noise
%  Gaussian random processes with covariances E[v(k)*v(k)'] = Q and
%  E[w(k)*w(k)'] = R.  The Kalman filter starts from the a posteriori
%  estimate and its covariance at sample 0, xhat0 and P0, and it
%  performs dynamic propagation and measurement update for
%  samples k = 1 though k = kmax.
%
%
%  Inputs:
%
%    F            The (nx)x(nx) state transition matrix for the time-
%                 invariant linear system.
%
%    Gamma        The (nx)x(nv) process noise gain matrix for the time-
%                 invariant linear system.
%
%    H            The (nz)x(nx) output gain matrix for the time-
%                 invariant linear system.
%
%    Q            The (nv)x(nv) symmetric positive definite process noise
%                 covariance matrix.
%
%    R            The (nz)x(nz) symmetric positive definite measurement
%                 noise covariance matrix.
%
%    xhat0        The (nx)x1 initial state estimate.
%
%    P0           The (nx)x(nx) symmetric positive definite initial state
%                 estimation error covariance matrix.
%
%    zhist        = [z(1)';z(2)';z(3)';...;z(kmax)'], the (kmax)-by-(nz)
%                 array that stores the measurement time history.
%                 z(1) = zhist(1,:)', z(2) = zhist(2,:)', etc.  Note
%                 that the state estimate xhathist(k+1,:)' and
%                 the measurement vector zhist(k,:)' correspond
%                 to the same time.
%
%  Outputs:
%
%    xhathist     = [xhat(0)';xhat(1)';xhat(2)';...;xhat(kmax)'], the
%                 (kmax+1)-by-(nx) array that stores the estimated time
%                 history for the state vector.  Note that Matlab
%                 does not allow zero indices in its arrays.  Thus,
%                 xhat(0) = xhathist(1,:)', xhat(1) = xhathist(2,:)', etc.
%
%    Phist        The nx-by-nx-by-(kmax+1) array that stores the
```

```
%               estimation error covariance time history as
%               computed by the Kalman filter.  Phist(:,:,k+1) is
%               the nx-by-nx estimation error covariance matrix
%               for the error in xhat(k) = xhathist(k+1,:)'.
%

%
%  Get problem dimensions and set up the output arrays.
%
   [nx,nv] = size(Gamma);
   kmax = size(zhist,1);
   kmaxp1 = kmax + 1;
   xhathist = zeros(kmaxp1,nx);
   Phist = zeros(nx,nx,kmaxp1);
%
%  Initialize quantities for use in the main loop and
%  store the first a posteriori estimate and its
%  error covariance matrix.
%
   xhatk = xhat0;
   Pk = P0;
   xhathist(1,:) = xhatk';
   Phist(:,:,1) = Pk;
%
%  Iterate through the kmax samples, first doing the propagation,
%  then doing the measurement update.
%
   for k = 0:(kmax - 1)
%
%  Propagation from sample k to sample k + 1.
%
      xbarkp1 = F*xhatk
      Pbarkp1 = F*Pk*(F') + Gamma*Q*(Gamma');
%
%  Measurement update at sample k + 1.
%
      zbarkp1 = H*xbarkp1;
      kp1 = k + 1;
      zkp1 = zhist(kp1,:)';
      nukp1 = zkp1 - zbarkp1;
      Skp1 = H*Pbarkp1*(H') + R;
      Wkp1 = (Pbarkp1*(H'))/Skp1;
      xhatkp1 = xbarkp1 + Wkp1*nukp1;
      Pkp1 = Pbarkp1 - Wkp1*Skp1*(Wkp1');
%
%  Store results.
%
      kp2 = kp1 + 1;
      xhathist(kp2,:) = xhatkp1';
      Phist(:,:,kp2) = Pkp1;
%
%  Prepare for next sample.
%
      xhatk = xhatkp1;
      Pk = Pkp1;
   end
```

Sheet 10 of 22

```
% kf_mcruns.m
%
%  Copyright (c) 2005 Mark L. Psiaki.  All rights reserved.
%
%  This Matlab script sets up a series of Monte-Carlo runs of
%  a truth model simulation and the associated Kalman filter
%  and computes evaluation results for the Kalman filter.
%
%  This script is for solving the 4th problem on the 2nd Prelim
%  exam during the Fall 2005 semester.  This is a solution to
%  Problem number 6 of Problem Set 5, but with modified Q and
%  R values.
%
%
%  Define problem matrices:
%
   Fk     = [  0.81671934103521,   0.08791146849849;...
              -3.47061412053765,   0.70624978972000];     % for all k
   Gammak = [  0.00464254201630;...
               0.08791146849849];                         % for all k
   Hk     = [  2.00000000000000,   0.30000000000000];     % for all k
%
   Qk     =    6.00000000000000;                          % for all k
   Rk     =    0.05000000000000;                          % for all k
%
   xhat0   = [  0.20000000000000;...
               -2.50000000000000];
   P0      = [  0.25000000000000,   0.08000000000000;...
                0.08000000000000,   0.50000000000000];
%
%  Set up arrays that will be used to compile Monte-Carlo
%  simulation results that are required to answer the problem
%  questions.
%
   xtilmean_10 = zeros(2,1);
   Pxtilxtil_10 = zeros(2,2);
   xtilmean_35 = zeros(2,1);
   Pxtilxtil_35 = zeros(2,2);
%
%  Use 50 samples in the truth model simulations and in the filter
%  runs.
%
   kmax = 50;
%
%  Run 50 Monte Carlo Simulations and store results.
%
   N_mc = 50;
   for n_mc = 1:N_mc
%
%  Do the truth model simulation for this Monte-Carlo case.
%
      [xtruehist,zhist] = kf_truthmodel(Fk,Gammak,Hk,Qk,Rk,xhat0,...
                                        P0,kmax);
%
%  Do the filter run for this Monte-Carlo case.
%
      [xhathist,Phist] = kf_solution(Fk,Gammak,Hk,Qk,Rk,xhat0,...
```

```
                                      P0,zhist);
%
%  Compute the estimation errors at samples 10 and 35.
%
      xtil_10 = (xtruehist(11,:) - xhathist(11,:))';
      xtil_35 = (xtruehist(36,:) - xhathist(36,:))';
%
%  Build up the mean errors and their covariances.
%
      xtilmean_10 = xtilmean_10 + xtil_10;
      Pxtilxtil_10 = Pxtilxtil_10 + xtil_10*(xtil_10');
      xtilmean_35 = xtilmean_35 + xtil_35;
      Pxtilxtil_35 = Pxtilxtil_35 + xtil_35*(xtil_35');
   end
%
%  Finish computing the mean errors and their covariances and display
%  them.  Also, display the Kalman filter's computed covariances at
%  these same two sample times.
%
   format long
   xtilmean_10 = xtilmean_10*(1/N_mc)
   Pxtilxtil_10 = Pxtilxtil_10*(1/N_mc)
   P10 = Phist(:,:,11)
   xtilmean_35 = xtilmean_35*(1/N_mc)
   Pxtilxtil_35 = Pxtilxtil_35*(1/N_mc)
   P35 = Phist(:,:,36)
```

```
Results from 50 Monte-Carlo Runs

xtilmean_10 =     0.00442619769592
                 -0.02849725242452

Pxtilxtil_10      0.00211630118767  -0.00053277520037
                 -0.00053277520037   0.07994766353256

P10 =             0.00169527846036   0.00065651099373
                  0.00065651099373   0.08593137727297

xtilmean_35 =     0.00441392040158
                  0.07758629367261

Pxtilxtil_35 =    0.00117678670648   0.00018966268363
                  0.00018966268363   0.07772490248148

P35 =             0.00162221660582   0.00066232408352
                  0.00066232408352   0.08454563074970


Results from 1000 Monte-Carlo Runs

xtilmean_10 =     0.00077323416023
                  0.00831733827821

Pxtilxtil_10 =    0.00169366222721   0.00049294907869
                  0.00049294907869   0.08221428146817

P10 =             0.00169527846036   0.00065651099373
                  0.00065651099373   0.08593137727297

xtilmean_35 =     0.00044194859522
                  0.00245897781691

Pxtilxtil_35 =    0.00159901821770   0.00065499400893
                  0.00065499400893   0.08130446225054

P35 =             0.00162221660582   0.00066232408352
                  0.00066232408352   0.08454563074970
```

Note how xtilmean_10, (Pxtilxtil_10 - P10), xtilmean_35, and (Pxtilxtil_35 - P35) are all smaller for the case with 1000 Monte-Carlo runs.

4) Problem 5-12 in Bar-Shalom (15 pts):

i.) $\underline{x}_{aug} = \begin{bmatrix} \underline{x} \\ \bar{w} \end{bmatrix}$ where $\underline{x}$ is the original 2×1 state vector

$$\underline{x}_{aug}(k+1) = \begin{bmatrix} 1 & T & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \underline{x}_{aug}(k) + \begin{bmatrix} T^2/2 \\ T \\ 0 \end{bmatrix} \underline{v}(k)$$

$$z(k) = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \underline{x}_{aug}(k) + \tilde{w}(k)$$

where wtilde(k) = w(k) - wbar(k) is the new zero-mean noise

so

$$F = \begin{bmatrix} 1 & T & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \Gamma = \begin{bmatrix} T^2/2 \\ T \\ 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

ii) One cannot estimate the sensor bias independently of the first element of $\underline{x}$ because the system is unobservable

$$\mathcal{O} = \begin{bmatrix} H \\ HF \\ HF^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & T & 1 \\ 1 & 2T & 1 \end{bmatrix}$$

rank $\mathcal{O}$ = $2 < 3$ because its first and third columns are identical.

5) Problem Set 6, #2 (20pts)

Results from first problem

```
xhat_final_regkf = 1.0e+005 * [ -2.21356459581035
                                -0.05440916380803
                                -0.00076759405563]

xhat_final_srif  = 1.0e+005 * [-2.21356459581035
                               -0.05440916380803
                               -0.00076759405563]

xhat_diff_regminussrif = 1.0e-010 * [ 0.87311491370201
                                     -0.31832314562052
                                      0.13173462320992]

P_final_regkf = [ 8.63663488922525  -0.73909536880184  -6.41653503360249
                 -0.73909536877414   1.22922644919996  -0.59557477155930
                 -6.41653503360239  -0.59557477155931   8.60892764914233]

P_final_srif  = [ 8.63663488920783  -0.73909536878839  -6.41653503361189
                 -0.73909536878839   1.22922644920663  -0.59557477155836
                 -6.41653503361189  -0.59557477155836   8.60892764914985]

P_diff_regminussrif = ...
      1.0e-010 * [ 0.17418955167159  -0.13455347946945   0.09408473999883
                   0.14241607892984  -0.06670441976553  -0.00938027433506
                   0.09507949982890  -0.00952238288221  -0.07522871214860]

abs(P_diff_regminussrif)./(abs(P_final_srif) + eps^6) = ...
      1.0e-010 * [ 0.020168683046826   0.182051579744067   0.014662857680350
                   0.192689719005147   0.054265363236027   0.015749952454359
                   0.014817888366672   0.015988559853357   0.008738453291105]
```

Results from second problem

```
xhat_final_regkf =  1.0e+004 * [-3.95586142622457
                                 0.03092158589606
                                 0.00325300128618]

xhat_final_srif =   1.0e+004 * [-3.95586151100119
                                 0.03092162247507
                                 0.00325301290478]

xhat_diff_regminussrif = 1.0e-003 * [ 0.84776620496996
                                     -0.36579009508841
                                     -0.11618600371577]

P_final_regkf = [ 6.57944434558310  -0.66645304555193  -5.24653825447921
                 -0.66645304549624   0.72580886645960  -0.78516468742299
                 -5.24653825447615  -0.78516468742291   6.81686762932198]

P_final_srif =  [ 6.57944540726625  -0.66645316023850  -5.24653908678925
                 -0.66645316023850   0.72580898379376  -0.78516480734902
```

Sheet 15 of 22

```
                 -5.24653908678925  -0.78516480734902   6.81686870148728]

P_diff_regminussrif = ...
      1.0e-005 * [-0.10616831467303   0.01146865642454   0.08323100457730
                   0.01147422565850  -0.01173341551697   0.01199260252216
                   0.08323131011068   0.01199261104867  -0.10721653076473]

abs(P_diff_regminussrif)./(abs(P_final_srif) + eps^6) = ...
      1.0e-006 * [ 0.161363622769450   0.172084958235261   0.158639825607847
                   0.172168523507245   0.161659827571163   0.152739939563148
                   0.158640407960089   0.152740048158311   0.157281202645634]
```

Note how the difference between the state estimates of the regular Kalman filter and the SRIF is about 1.e+07 times larger for the second problem than for the first problem. The covariance discrepancies are larger by about 1.e+05 and the relative discrepancies in the covariance matrices are larger by about 1.e+04. This happens because the accuracy of the regular Kalman filter degrades on the second problem.

```
% kf_script.m
%
%  Copyright (c) 2005 Mark L. Psiaki.  All rights reserved.
%
%  This Matlab script sets up and solves two Kalman filtering
%  problems.  It solves each problem twice, once using a
%  regular Kalman filter, and once using an SRIF implementation.
%  It then compares results.
%
%  This script is for solving the 6th problem on the 2nd Prelim
%  exam during the Fall 2005 semester.  This is the solution to
%  Problem number 3 of Problem Set 6.
%
%
%  Set up the first Kalman filtering problem.
%
   clear
   kf_example03a
%
%  Solve it using the standard Kalman filter.
%
   [xhathist_regkf,Phist_regkf] = kf_solution(Fk,Gammak,Hk,Qk,Rk,xhat0,...
                                              P0,zhist);
%
%  Re-solve it using the standard SRIF.
%
   [xhathist_srif,Phist_srif] = srif_solution(Fk,Gammak,Hk,Qk,Rk,xhat0,...
                                              P0,zhist);
%
%  Compare the terminal state estimates and covariances.
%
   kmax = size(zhist,1);
   kmaxp1 = kmax + 1;
   xhat_final_regkf = xhathist_regkf(kmaxp1,:)'
   xhat_final_srif = xhathist_srif(kmaxp1,:)'
   xhat_diff_regminussrif = xhat_final_regkf - xhat_final_srif
   P_final_regkf = Phist_regkf(:,:,kmaxp1)
   P_final_srif = Phist_srif(:,:,kmaxp1)
   P_diff_regminussrif = P_final_regkf - P_final_srif
%
%
%  Repeat for the second problem.
%
   clear
   kf_example03b
%
%  Solve it using the standard Kalman filter.
%
   [xhathist_regkf,Phist_regkf] = kf_solution(Fk,Gammak,Hk,Qk,Rk,xhat0,...
                                              P0,zhist);
%
%  Re-solve it using the standard SRIF.
%
   [xhathist_srif,Phist_srif] = srif_solution(Fk,Gammak,Hk,Qk,Rk,xhat0,...
                                              P0,zhist);
%
%  Compare the terminal state estimates and covariances.
```

see Problem 3 solution for this function

```
%
  kmax = size(zhist,1);
  kmaxp1 = kmax + 1;
  xhat_final_regkf = xhathist_regkf(kmaxp1,:)'
  xhat_final_srif = xhathist_srif(kmaxp1,:)'
  xhat_diff_regminussrif = xhat_final_regkf - xhat_final_srif
  P_final_regkf = Phist_regkf(:,:,kmaxp1)
  P_final_srif = Phist_srif(:,:,kmaxp1)
  P_diff_regminussrif = P_final_regkf - P_final_srif
```

```
function [xhathist,Phist] = srif_solution(F,Gamma,H,Q,R,xhat0,...
                                          P0,zhist)
%
%  Copyright (c) 2005 Mark L. Psiaki.  All rights reserved.
%
%  This function performs Kalman filter state estimation for
%  the dynamic system model:
%
%     x(k+1) = F*x(k) + Gamma*v(k)
%     z(k)   = H*x(k) + w(k)
%
%  Where v(k) and w(k) are uncorrelated, zero-mean, white-noise
%  Gaussian random processes with covariances E[v(k)*v(k)'] = Q and
%  E[w(k)*w(k)'] = R.  The Kalman filter starts from the a posteriori
%  estimate and its covariance at sample 0, xhat0 and P0, and it
%  performs dynamic propagation and measurement update for
%  samples k = 1 though k = kmax.
%
%  This particular implementation uses SRIF techniques.
%
%
%  Inputs:
%
%    F            The (nx)x(nx) state transition matrix for the time-
%                 invariant linear system.
%
%    Gamma        The (nx)x(nv) process noise gain matrix for the time-
%                 invariant linear system.
%
%    H            The (nz)x(nx) output gain matrix for the time-
%                 invariant linear system.
%
%    Q            The (nv)x(nv) symmetric positive definite process noise
%                 covariance matrix.
%
%    R            The (nz)x(nz) symmetric positive definite measurement
%                 noise covariance matrix.
%
%    xhat0        The (nx)x1 initial state estimate.
%
%    P0           The (nx)x(nx) symmetric positive definite initial state
%                 estimation error covariance matrix.
%
%    zhist        = [z(1)';z(2)';z(3)';...;z(kmax)'], the (kmax)-by-(nz)
%                 array that stores the measurement time history.
%                 z(1) = zhist(1,:)', z(2) = zhist(2,:)', etc.  Note
%                 that the state estimate xhathist(k+1,:)' and
%                 the measurement vector zhist(k,:)' correspond
%                 to the same time.
%
%  Outputs:
%
%    xhathist     = [xhat(0)';xhat(1)';xhat(2)';...;xhat(kmax)'], the
%                 (kmax+1)-by-(nx) array that stores the estimated time
%                 history for the state vector.  Note that Matlab
%                 does not allow zero indices in its arrays.  Thus,
%                 xhat(0) = xhathist(1,:)', xhat(1) = xhathist(2,:)', etc.
```

```
%
%    Phist      The nx-by-nx-by-(kmax+1) array that stores the
%               estimation error covariance time history as
%               computed by the Kalman filter.  Phist(:,:,k+1) is
%               the nx-by-nx estimation error covariance matrix
%               for the error in xhat(k) = xhathist(k+1,:)'.
%

%
%  Get problem dimensions and set up the output arrays.
%
   [nx,nv] = size(Gamma);
   kmax = size(zhist,1);
   kmaxp1 = kmax + 1;
   xhathist = zeros(kmaxp1,nx);
   Phist = zeros(nx,nx,kmaxp1);
%
%  Determine the square-root information matrix for
%  the process noise, and transform the measurements
%  to have an error with an identity covariance.
%
   Rwwk = inv(chol(Q)');
   Ra = chol(R);
   Rainvtr = inv(Ra');
   Ha = Rainvtr*H;
   zahist = zhist*(Rainvtr');
%
%  Initialize quantities for use in the main loop and
%  store the first a posteriori estimate and its
%  error covariance matrix.
%
   Rxx0 = inv(chol(P0)');
   zx0 = Rxx0*xhat0;
   Rxxk = Rxx0;
   zxk = zx0;
   xhathist(1,:) = xhat0';
   Phist(:,:,1) = P0;
%
%  Compute inv(F) and inv(F)*Gamma for later use.
%
   Finv = inv(F);
   FinvGamma = Finv*Gamma;
%
%  Iterate through the kmax samples, first doing the propagation,
%  then doing the measurement update.
%
   for k = 0:(kmax - 1)
%
%  Propagation from sample k to sample k + 1.
%
      Rbig = [              Rwwk, zeros(nv,nx);...
             (-Rxxk*FinvGamma),    Rxxk*Finv];
      [Taktr,Rdum] = qr(Rbig);
      Tak = Taktr';
      zdum = Tak*[zeros(nv,1);zxk];
      idumxvec = [(nv+1):(nv+nx)]';
      Rbarxxkp1 = Rdum(idumxvec,idumxvec);
```

```
        zbarxkp1 = zdum(idumxvec,1);
%
%   Measurement update at sample k + 1.
%
        [Tbkp1tr,Rdum] = qr([Rbarxxkp1;Ha]);
        Tbkp1 = Tbkp1tr';
        kp1 = k + 1;
        zdum = Tbkp1*[zbarxkp1;zahist(kp1,:)'];
        idumxvec = [1:nx]';
        Rxxkp1 = Rdum(idumxvec,idumxvec);
        zxkp1 = zdum(idumxvec,1);
%
%   Compute and store the state estimate and its
%   estimation error covariance at sample k+1.
%
        Rxxkp1inv = inv(Rxxkp1);
        xhatkp1 = Rxxkp1inv*zxkp1;
        Pkp1 = Rxxkp1inv*(Rxxkp1inv');
        kp2 = kp1 + 1;
        xhathist(kp2,:) = xhatkp1';
        Phist(:,:,kp2) = Pkp1;
%
%   Prepare for next sample.
%
        zxk = zxkp1;
        Rxxk = Rxxkp1;
    end
```

6) Problem Set 6, #3 (15 pts):

$$\underline{x}^*(k) = F^{-1}(k)\left[\underline{x}^*(k+1) - G(k)\underline{u}(k) - \Gamma(k)\underline{v}^*(k)\right]$$

but $\quad \bar{\underline{x}}(k+1) = F(k)\hat{\underline{x}}(k) + G(k)\underline{u}(k)$

so $\quad -F^{-1}(k)G(k)\underline{u}(k) = \hat{\underline{x}}(k) - F^{-1}(k)\bar{\underline{x}}(k+1)$

also $\quad \underline{v}^*(k) = Q(k)\Gamma^T(k)\bar{P}^{-1}(k+1)\left[\underline{x}^*(k+1) - \bar{\underline{x}}(k+1)\right]$

substituting into the first equation yields

$$\underline{x}^*(k) = \hat{\underline{x}}(k) + F^{-1}(k)\Big[\underline{x}^*(k+1) - \bar{\underline{x}}(k+1)$$

$$- \Gamma(k)Q(k)\Gamma^T(k)\bar{P}^{-1}(k+1)\left\{\underline{x}^*(k+1) - \bar{\underline{x}}(k+1)\right\}\Big]$$

or, re-arranging terms

$$\underline{x}^*(k) = \hat{\underline{x}}(k) + F^{-1}(k)\left[I - \Gamma(k)Q(k)\Gamma^T(k)\bar{P}^{-1}(k+1)\right]\cdot\left[\underline{x}^*(k+1) - \bar{\underline{x}}(k+1)\right]$$

replacing I by $\bar{P}(k+1)\bar{P}^{-1}(k+1)$ and factoring out $\bar{P}^{-1}(k+1)$ yields

$$\underline{x}^*(k) = \hat{\underline{x}}(k) + F^{-1}(k)\left[\bar{P}(k+1) - \Gamma(k)Q(k)\Gamma^T(k)\right]\cdot\bar{P}^{-1}(k+1)\left[\underline{x}^*(k+1) - \bar{\underline{x}}(k+1)\right]$$

but $$\bar{P}(k+1) - \Gamma(k)\,Q(k)\,\Gamma^T(k) = F(k)\,P(k)\,F^T(k)$$

so

$$\underline{x}^*(k) = \hat{\underline{x}}(k) + F^{-1}(k)\left[F(k)\,P(k)\,F^T(k)\right] \cdot \bar{P}^{-1}(k+1)\left[\underline{x}^*(k+1) - \bar{\underline{x}}(k+1)\right]$$

or

$$\underline{x}^*(k) = \hat{\underline{x}}(k) + P(k)\,F^T(k)\,\bar{P}^{-1}(k+1)\left[\underline{x}^*(k+1) - \bar{\underline{x}}(k+1)\right]$$

as was to be proved.