Spencer Freeman

AOE 5784, Estimation and Filtering

12/17/2024

Final

HW 1 | Spencer Freeman | AOE 5784 | 9/20/24

1-9) $H_0 : \theta = 0$ $H_1 : \theta \neq 0$

$z_i = \theta + w_i$ $i = 1, \dots n$

$w_i \sim N(0, \sigma)$ $\underline{w} = [w_1 \dots w_n]^T$

$E[\underline{w}\underline{w}^T] = P$

① Optimal hypothesis test for false alarm prob. α:

$\boxed{\text{Two-sided locally most powerful test}}$

② $n = 2$, $P = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$, $\alpha = 1\%$

$$q(\underline{z}, \theta_1) \approx \left[\frac{dq}{d\theta} \Big|_{\underline{z},0} \right] \theta_1 + \frac{1}{2} \left[\frac{d^2 q}{d\theta^2} \Big|_{\underline{z},0} \right] \theta_1^2 + O(\theta_1^3)$$

$$q(\underline{z}, \theta_1) = \ln\left(\frac{p(\underline{z} | \theta = \theta_1)}{p(\underline{z} | \theta = 0)} \right)$$

$$\underline{z} = [z_1 \dots z_n]^T = \underline{e}\,\theta + \underline{w}$$

test: accept H_1 if $|\beta(\underline{z})| = \left| \left[\frac{dq}{d\theta_1} \Big|_{\underline{z},0} \right] \right| \geq \frac{q_0}{|\theta_1|} = \beta_0$;

otherwise, accept H_0

$$\alpha = \int_{-\infty}^{-\beta_0} p(\beta | \theta = 0)\, d\beta + \int_{\beta_0}^{\infty} p(\beta | \theta = 0)\, d\beta$$

$$p(\underline{z}\mid\theta=0) = p(\underline{e}(0)+\underline{w}) = p(\underline{w}) = \frac{1}{(2\pi)^n\sqrt{|P|}}\cdot e^{-\frac{1}{2}(\underline{z}-\underline{\mu_w})^T P^{-1}(\underline{z}-\underline{\mu_w})}$$

$$\underline{\mu_w} \to 0$$

$$p(\underline{z}\mid\theta=0) = \frac{1}{(2\pi)^n\sqrt{|P|}}\cdot e^{-\frac{1}{2}\underline{z}^T P^{-1}\underline{z}}$$

$$\underline{z}\sim N(\underline{e}\theta_1, P)$$

$$p(\underline{z}\mid\theta=\theta_1) = p(\underline{e}\theta_1+\underline{w}) = \frac{1}{(2\pi)^n\sqrt{|P|}}\cdot e^{-\frac{1}{2}(\underline{z}-\theta_1\underline{e})^T P^{-1}(\underline{z}-\theta_1\underline{e})}$$

$$p(\underline{z}\mid\theta=\theta_1) = \frac{1}{(2\pi)^n\sqrt{|P|}}\cdot e^{-\frac{1}{2}(\underline{z}-\theta_1\underline{e})^T P^{-1}(\underline{z}-\theta_1\underline{e})}$$

$$q(\underline{z},\theta_1) = \ln\left(e^{-\frac{1}{2}(\underline{z}-\theta_1\underline{e})^T P^{-1}(\underline{z}-\theta_1\underline{e})+\frac{1}{2}\underline{z}^T P^{-1}\underline{z}}\right)$$

$$q(\underline{z},\theta_1) = \frac{1}{2}\left(\underline{z}^T P^{-1}\underline{z} - (\underline{z}-\theta_1\underline{e})^T P^{-1}(\underline{z}-\theta_1\underline{e})\right).$$

$$\beta(\underline{z}) = \left.\frac{dq}{d\theta_1}\right|_{\underline{z},0} = -\frac{1}{2}\frac{d}{d\theta_1}\left[(\underline{z}-\theta_1\underline{e})^T P^{-1}(\underline{z}-\theta_1\underline{e})\right]$$

$$\beta(\underline{z}) = -\frac{1}{2}D(f\circ g)(\theta_1) = -\frac{1}{2}\left[Df(g(\theta_1))\right]\left[Dg(\theta_1)\right] \to \text{chain rule}$$

$$g(\theta_1) = \underline{z}-\theta_1\underline{e}, \quad f(g(\theta_1)) = g(\theta_1)^T P^{-1} g(\theta_1) \to \text{quadratic form}$$

$$Df(g(\theta_1)) = g(\theta_1)^T(P^{-1}+P^{-1^T}) \to \text{derivative of quadratic form}$$

$$Dg(\theta_1) = -\underline{e} \qquad P^{-1^T} = P^{-1}$$

$$\beta(\underline{z}) = -\frac{1}{2}(\underline{z}-\underbrace{\theta_1}_{\theta_1=0}\underline{e})^T(P^{-1}+P^{-1^T})(-\underline{e}) = \underline{z}^T P^{-1}\underline{e}$$

$p(\beta(\underline{z}) | \theta = 0)$, $\beta(\underline{z}) = \underline{z}^T P^{-1} \underline{e} = \underline{z}^T \underline{P_e}$

$$\underset{(\theta=0)}{\beta(\underline{z})} = \underline{w}^T \underline{P_e}$$

distribution of sum of jointly normal random variables

$$\underset{(\theta=0)}{\beta(\underline{z})} \sim N(\overset{0}{\underline{\mu_w}^T} \underline{P_e} , \underline{P_e}^T P \underline{P_e})$$

$$p(\beta(\underline{z}) | \theta = 0) = \frac{1}{\underline{P_e}^T P \underline{P_e} \sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{\beta}{\underline{P_e}^T P \underline{P_e}}\right)^2}$$

$$\alpha = \int_{-\infty}^{-\beta_0} p(\beta(\underline{z}) | \theta = 0) d\beta + \int_{\beta_0}^{\infty} p(\beta(\underline{z}) | \theta = 0) d\beta$$

$$\boxed{\beta_0 = -\text{norminv}(\alpha/2, \mu_\beta, \sigma_\beta), \quad \mu_\beta = 0, \quad \sigma_\beta = \sqrt{\underline{P_e}^T P \underline{P_e}}}$$

$P = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$

$\alpha = 0.01$

$\mu_\beta = 0$

$\sigma_\beta = 1.1547$

$\beta_0 = 2.9743$

$$\boxed{\text{accept } H_1 \text{ if } |\underline{z}^T P^{-1} \underline{e}| \geq \beta_0}$$

(8)

From BS 1-9:

$\beta = \underline{z}^T P^{-1} \underline{e}$, accept H_1 if $|\beta| \geq \beta_0$

$$\underbrace{q(\underline{z}, \theta_1)}_{\text{likelihood ratio}} = \frac{1}{2}\underline{z}^T P^{-1} \underline{z} - \frac{1}{2}(\underline{z} - \theta_1 \underline{e})^T P^{-1} (\underline{z} - \theta_1 \underline{e})$$

alternate:

$p(\underline{z}|H_1) = p[\underline{z} | \hat{\theta}_1(\underline{z})]$, $\hat{\theta}_1(\underline{z})$ is optimal estimate of θ_1, $\hat{\theta}_1(\underline{z})$ <u>maximizes</u> $p(\underline{z}|\theta_1)$

$$\frac{dq}{d\theta_1} = 0$$

$$q(\underline{z}, \theta_1) = \theta_1 \underline{z}^T P^{-1} \underline{e} - \frac{\theta_1^2}{2} \underline{e}^T P^{-1} \underline{e}$$

$$\frac{dq}{d\theta_1} = \underline{z}^T P^{-1} \underline{e} - \theta_1 \underline{e}^T P^{-1} \underline{e} = 0, \quad \hat{\theta}_1 = \frac{\underline{z}^T P^{-1} \underline{e}}{\underline{e}^T P^{-1} \underline{e}}$$

$$q(\underline{z}, \hat{\theta}) = \frac{\underline{z}^T P^{-1} \underline{e}}{\underline{e}^T P^{-1} \underline{e}} (\underline{z}^T P^{-1} \underline{e}) - \left(\frac{\underline{z}^T P^{-1} \underline{e}}{\underline{e}^T P^{-1} \underline{e}}\right)^2 \frac{\underline{e}^T P^{-1} \underline{e}}{2}$$

$$q(\underline{z}, \hat{\theta}_1) = \frac{(\underline{z}^T P^{-1} \underline{e})^2}{\underline{e}^T P^{-1} \underline{e}} - \frac{(\underline{z}^T P^{-1} \underline{e})^2}{\underline{e}^T P^{-1} \underline{e}} \frac{1}{2} = \frac{1}{2} \frac{(\underline{z}^T P^{-1} \underline{e})^2}{\underline{e}^T P^{-1} \underline{e}}$$

$$q(\underline{z}, \hat{\theta}_1) = \frac{1}{2} \frac{\beta^2}{\underline{e}^T P^{-1} \underline{e}}$$, accept H_1 if $q(\underline{z}, \hat{\theta}_1) > q_0$

$$q_0 = \frac{1}{2} \frac{\beta_0^2}{\underline{e}^T P^{-1} \underline{e}}$$

$$\boxed{(\underline{e}^T P^{-1} \underline{z})^2 \geq \beta_0^2 \leftrightarrow \beta^2 \geq \beta_0^2 \quad |\beta| \geq \beta_0}$$

HW5	Spencer Freeman	AOE 5784	11/20/24

BS 5-10) Prove that the state estimation errors are not white

$$E[\tilde{\underline{x}}(k+1|k+1)\tilde{\underline{x}}^T(k|k)] = [I - W(k+1)H(k+1)]F(k)P(k|k)$$

$$\tilde{\underline{x}}(k+1|k+1) = \underline{x}(k+1) - \hat{\underline{x}}(k+1|k+1)$$

$$\tilde{\underline{x}}(k|k) = \underline{x}(k) - \hat{\underline{x}}(k|k)$$

$$\tilde{\underline{x}}(k+1|k+1) = F(k)\underline{x}(k) + G(k)\underline{u}(k) + \Gamma(k)\underline{v}(k) - \ldots$$
$$[\hat{\underline{x}}(k+1|k) + W(k+1)\underline{\nu}(k+1)]$$

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

$$\tilde{\underline{x}}(k+1|k+1) = F(k)[\underline{x}(k) - \hat{\underline{x}}(k|k)] - W(k+1)\underline{\nu}(k+1) + \Gamma(k)\underline{v}(k)$$

$$\tilde{\underline{x}}(k+1|k+1) = F(k)\tilde{\underline{x}}(k|k) + \Gamma(k)\underline{v}(k) - \ldots$$
$$W(k+1)[H(k+1)\underline{x}(k+1) + \underline{w}(k+1) - H(k+1)\hat{\underline{x}}(k+1|k)]$$

$$= F(k)\tilde{\underline{x}}(k|k) + \Gamma(k)\underline{v}(k) - \ldots$$
$$W(k+1)[H(k+1)(F(k)(\underline{x}(k) - \hat{\underline{x}}(k|k)) + \Gamma(k)\underline{v}(k)) + \underline{w}(k+1)]$$

$$= [I - W(k+1)H(k+1)]F(k)\tilde{\underline{x}}(k|k) + [I - W(k+1)H(k+1)]\Gamma(k)\underline{v}(k) + \ldots$$
$$W(k+1)\underline{w}(k+1)$$

$$= [I - W(k+1)H(k+1)][F(k)\tilde{\underline{x}}(k|k) + \Gamma(k)\underline{v}(k)] - \underline{w}(k+1)$$

$$E[\tilde{\underline{x}}(k+1|k+1)\,\tilde{\underline{x}}^T(k|k)] = \dots$$

$$[I - W(k+1)H(k+1)]\big[F(k)\,E[\tilde{\underline{x}}(k|k)\,\tilde{\underline{x}}^T(k|k)] +$$

expectation is linear ←

$$\Gamma(k)\,E[\underline{v}(k)\,\tilde{\underline{x}}^T(k|k)]\big] +$$

$$W(k+1)\,E[\underline{w}(k+1)\,\tilde{\underline{x}}(k|k)]$$

$$E[\tilde{\underline{x}}(k+1|k+1)\,\tilde{\underline{x}}(k|k)] = \dots$$

$$[I - W(k+1)H(k+1)]\,F(k)\,P(k|k) + 0 + 0 \quad \checkmark$$

→ Process noise is <u>white</u>, therefore uncorrelated to current state

→ Measurement noise is uncorrelated with state

HW7 Problem 1 Final

Spencer Freeman

AOE 5784

12/17/2024

Results:

```
HW7-P1-Final
120 RK Steps
State Estimate:

fprinted =

   1.0e+02 *

   0.620701210018915
  -1.529444304253257
  -0.705857744405961
  -0.103775385949128

Partial Derivative wrt xk Estimate:

dfprinted_dxk =

   1.0e+02 *

   0.728877498626523  -1.472776463825067  -0.548136654443188  -1.753681011366440
  -1.802276348157174   3.647524833920750   1.359299650355381   4.334068375928841
  -0.870935186580203   1.754235467078160   0.646202266188082   2.077782611804557
  -0.167654300505863   0.348202994601513   0.134817316015435   0.406122909801801

Partial Derivative wrt vk Estimate:

dfprinted_dvk =

   1.0e+02 *

   1.033625821871495  -0.478443883796926   0.161916522846359
  -2.522534149758227   1.183721601826497  -0.385332453603791
  -1.193825976395165   0.538038300656368  -0.143777850356074
  -0.233714858740136   0.144761934545640  -0.037248695108266

60 RK Steps
State Estimate:

fprinted =

   1.0e+02 *

   0.620697482796144
  -1.529435074986165
  -0.705853308658435
  -0.103774505997242

Partial Derivative wrt xk Estimate:

dfprinted_dxk =

   1.0e+02 *

   0.623180322654275  -1.258552351010682  -0.468159647121948  -1.499667430804622
  -1.540539945582156   3.117091929068315   1.161314212345298   3.704913219247961
  -0.745300454509505   1.499428602715482   0.550897764793195   1.775392564654726
  -0.142571006994151   0.297577105024353   0.116036029385876   0.345874841165753

Partial Derivative wrt vk Estimate:

dfprinted_dvk =

   1.0e+02 *

   0.885599711582125  -0.407521904148829   0.139724403897416
```

```
 -2.155886828683541   1.008425181308094  -0.330347147756422
 -1.017332286172073   0.453506086768662  -0.116838467575323
 -0.198737287889482   0.128531955988640  -0.032161523410142
```

Error in state estimate is 10 times larger halving the number of steps. Error in derivatives is nearly doubled.
>>

Script hw7_prob1_final.m :

```
%% Fix the code to compute the required partial derivatives.
% Spencer Freeman, 12/17/2024
% AOE 5784, Estimation and Filtering
%
% This script solves number 1 of problem set 7
% --------------------------------------------------------------------------
clear;clc;close all

disp('HW7-P1-Final')

format long

% Test your code using the supplied test function "fscript_ts01.m".
% Test the function by numerically integrating from a random initial
% condition and using a random process noise vector. Integrate over a
% time span of 3 seconds, and use 120 4th-order Runge-Kutta numerical
% integration steps. Compare your results with the exact results, which
% you can computed as outlined in the initial comments section of
% "fscript_ts01.m". Test the results again, but this time use only 60
% 4th-order Runge-Kutta steps for the same inputs. Does the error for
% this second case change as you expect it to change in comparison with
% the error for the first numerical integration case?

tk = 0; % s
tkp1 = 3; % s

% xk = rand(4, 1);
% vk = rand(3, 1);
% uk = [];
xk = [-0.40; 0.85; -0.60; -1.65];
uk = [];
vk = [-0.77; 1.30; 1.65];
idervflag = true;
fscriptname = 'fscript_ts01';

[~, A, D] = fscript_ts01(tk,xk,uk,vk,idervflag);

[dfprinted_dxk_true, dfprinted_dvk_true] = c2d(A, D, (tkp1 - tk));
fprinted_true = dfprinted_dxk_true*xk + dfprinted_dvk_true*vk;

%% many steps
nRK = 120;

[fprinted, dfprinted_dxk, dfprinted_dvk] = ...
    ...
    c2dnonlinear(xk,uk,vk,tk,tkp1,nRK,fscriptname,idervflag);

disp(string(nRK) + " RK Steps")

disp('State Estimate:')
fprinted_true
fprinted
disp('Partial Derivative wrt xk Estimate:')
dfprinted_dxk_true
dfprinted_dxk
disp('Partial Derivative wrt vk Estimate:')
dfprinted_dvk_true
dfprinted_dvk


%% few steps
```

```
nRK = 60;

[fprinted, dfprinted_dxk, dfprinted_dvk] = ...
   ...
   c2dnonlinear(xk,uk,vk,tk,tkp1,nRK,fscriptname,idervflag);

disp(string(nRK) + " RK Steps")

disp('State Estimate:')
fprinted_true
fprinted
disp('Partial Derivative wrt xk Estimate:')
dfprinted_dxk_true
dfprinted_dxk
disp('Partial Derivative wrt vk Estimate:')
dfprinted_dvk_true
dfprinted_dvk

disp('Error in state estimate is 10 times larger halving the number of steps. Error in derivatives is nearly doubled.')
```

Function c2dnonlinear.m :

```
function [fprinted,dfprinted_dxk,dfprinted_dvk] = ...
         c2dnonlinear(xk,uk,vk,tk,tkp1,nRK,fscriptname,idervflag)
%
%  Copyright (c) 2002 Mark L. Psiaki.  All rights reserved.
%
%
%  This function derives a nonlinear discrete-time dynamics function
%  for use in a nonlinear difference equation via 4th-order
%  Runge-Kutta numerical integration of a nonlinear differential
%  equation.  If the nonlinear differential equation takes the
%  form:
%
%            xdot = fscript{t,x(t),uk,vk}
%
%  and if the initial condition is x(tk) = xk, then the solution
%  gets integrated forward from time tk to time tkp1 using nRK
%  4th-order Runge-Kutta numerical integration steps in order to
%  compute fprinted(k,xk,uk,vk) = x(tkp1).  This function can
%  be used in a nonlinear dynamics model of the form:
%
%      xkp1 = fprinted(k,xk,uk,vk)
%
%  which is the form defined in MAE 676 lecture for use in a nonlinear
%  extended Kalman filter.
%
%  This function also computes the first partial derivative of
%  fprinted(k,xk,uk,vk) with respect to xk, dfprinted_dxk, and with
%  respect to vk, dfprinted_dvk.
%
%
%  Inputs:
%
%   xk          The state vector at time tk, which is the initial
%               time of the sample interval.
%
%   uk          The control vector, which is held constant
%               during the sample interval from time tk to time
%               tkp1.
%
%   vk          The discrete-time process noise disturbance vector,
%               which is held constant during the sample interval
%               from time tk to time tkp1.
%
%   tk          The start time of the numerical integration
%               sample interval.
%
%   tkp1        The end time of the numerical integration
```

```
%               sample interval.
%
%   nRK         The number of Runge-Kutta numerical integration
%               steps to take during the sample interval.
%
%   fscriptname    The name of the Matlab .m-file that contains the
%               function which defines fscript{t,x(t),uk,vk}.
%               This must be a character string.  For example, if
%               the continuous-time differential equation model is
%               contained in the file rocketmodel.m with the function
%               name rocketmodel, then on input to the present
%               function fscriptname must equal 'rocketmodel',
%               and the first line of the file rocketmodel.m
%               must be:
%
%               function [fscript,dfscript_dx,dfscript_dvtil] = ...
%                       rocketmodel(t,x,u,vtil,idervflag)
%
%               The function must be written so that fscript
%               defines xdot as a function of t, x, u, and vtil
%               and so that dfscript_dx and dfscript_dvtil are the
%               matrix partial derivatives of fscript with respect
%               to x and vtil if idervflag = 1.  If idervflag = 0, then
%               these outputs must be empty arrays.
%
%   idervflag     A flag that tells whether (idervflag = 1) or not
%               (idervflag = 0) the partial derivatives
%               dfprinted_dxk and dfprinted_dvk must be calculated.
%               If idervflag = 0, then these outputs will be
%               empty arrays.
%
%  Outputs:
%
%   fprinted      The discrete-time dynamics vector function evaluated
%               at k, xk, uk, and vk.
%
%   dfprinted_dxk  The partial derivative of fprinted with respect to
%               xk.  This is a Jacobian matrix.  It is evaluated and
%               output only if idervflag = 1.  Otherwise, an
%               empty array is output.
%
%   dfprinted_dvk  The partial derivative of fprinted with respect to
%               vk.  This is a Jacobian matrix.  It is evaluated and
%               output only if idervflag = 1.  Otherwise, an
%               empty array is output.
%

%
%  Prepare for the Runge-Kutta numerical integration by setting up
%  the initial conditions and the time step.
%
  x = xk;
  if idervflag == 1
    nx = size(xk,1);
    nv = size(vk,1);
    F = eye(nx); % ANSWER
    Gamma = zeros(nx, nv); % ANSWER
  end
  t = tk;
  delt = (tkp1 - tk)/nRK;
%
%  This loop does one 4th-order Runge-Kutta numerical integration step
%  per iteration.  Integrate the state.  If partial derivatives are
%  to be calculated, then the partial derivative matrices simultaneously
%  with the state.
%

  for jj = 1:nRK
    if idervflag == 1
      [fscript,dfscript_dx,dfscript_dvtil] = ...
```

```
            feval(fscriptname,t,x,uk,vk,1);
      dFa = (dfscript_dx * F)*delt; % ANSWER
      dGammaa = (dfscript_dx * Gamma + dfscript_dvtil)*delt;  % ANSWER
    else
      fscript = feval(fscriptname,t,x,uk,vk,0);
    end
    dxa = fscript*delt;
%
    if idervflag == 1
      [fscript,dfscript_dx,dfscript_dvtil] = ...
            feval(fscriptname,(t + 0.5*delt),(x + 0.5*dxa),...
                uk,vk,1);
      dFb = (dfscript_dx * F)*delt; % ANSWER
      dGammab = (dfscript_dx * Gamma + dfscript_dvtil)*delt;  % ANSWER
    else
      fscript = feval(fscriptname,(t + 0.5*delt),(x + 0.5*dxa),...
                uk,vk,0);
    end
    dxb = fscript*delt;
%
    if idervflag == 1
      [fscript,dfscript_dx,dfscript_dvtil] = ...
            feval(fscriptname,(t + 0.5*delt),(x + 0.5*dxb),...
                uk,vk,1);
      dFc = (dfscript_dx * F)*delt; % ANSWER
      dGammac = (dfscript_dx * Gamma + dfscript_dvtil)*delt;  % ANSWER
    else
      fscript = feval(fscriptname,(t + 0.5*delt),(x + 0.5*dxb),...
                uk,vk,0);
    end
    dxc = fscript*delt;
%
    if idervflag == 1
      [fscript,dfscript_dx,dfscript_dvtil] = ...
            feval(fscriptname,(t + delt),(x + dxc),...
                uk,vk,1);
      dFd = (dfscript_dx * F)*delt; % ANSWER
      dGammad = (dfscript_dx * Gamma + dfscript_dvtil)*delt; % ANSWER
    else
      fscript = feval(fscriptname,(t + delt),(x + dxc),...
                uk,vk,0);
    end
    dxd = fscript*delt;
%
    x = x + (dxa + 2*(dxb + dxc) + dxd)*(1/6);
    if idervflag == 1
      F = F + (dFa + 2*(dFb + dFc) + dFd)*(1/6);
      Gamma = Gamma + ...
          (dGammaa + 2*(dGammab + dGammac) + dGammad)*(1/6);
    end
    t = t + delt;
  end
%
%  Assign the results to the appropriate outputs.
%
  fprinted = x;
  if idervflag == 1
    dfprinted_dxk = F;
    dfprinted_dvk = Gamma;
  else
    dfprinted_dxk = [];
    dfprinted_dvk = [];
  end
```

(3)

EKF:

$$\underline{x}(k+1) = \underline{f}(k, \underline{x}(k), \underline{u}(k), \underline{v}(k)) \approx$$

$$\underbrace{\underline{f}(k, \hat{\underline{x}}(k), \underline{u}(k), 0) + F(k)[\underline{x}(k) - \hat{\underline{x}}(k)] + \Gamma(k)\underline{v}(k)}_{\text{1st order Taylor series approximation}}$$

$$\underline{h}(k+1, \underline{x}(k+1)) \approx \underbrace{\underline{h}(k+1, \bar{\underline{x}}(k+1)) + H(k+1)[\underline{x}(k+1) - \bar{\underline{x}}(k+1)]}_{\text{1st order Taylor series approximation}}$$

SRIF Propagation:

Solve dynamics model equation for $x(k)$ in terms of $\underline{x}(k+1)$, $\underline{u}(k)$, + $\underline{v}(k)$. Substitute result into a posteriori $\underline{x}(k)$ SRI equation

$$\underline{z}_x(k) = \mathcal{R}_{xx}(k)\underline{x}(k) + \underline{w}_x(k) \quad \rightarrow \text{SRI equation}$$

$$\underline{x}(k) = F^{-1}(k)[\underline{x}(k+1) - \underline{f}(k, \hat{\underline{x}}(k), \underline{u}(k), 0) - \Gamma(k)\underline{v}(k)] + \hat{\underline{x}}(k)$$

$$0 = \mathcal{R}_{vv}(k)\underline{v}(k) + \underline{w}_v(k)$$

$$\begin{bmatrix} 0 \\ z_x + \mathcal{R}_{xx}(k)[F^{-1}(k)\underline{f}(k, \hat{\underline{x}}(k), \underline{u}(k), 0) - \hat{\underline{x}}(k)] \end{bmatrix} = \dots$$

$$\begin{bmatrix} \mathcal{R}_{vv}(k) & 0 \\ -\mathcal{R}_{xx}(k)F^{-1}(k)\Gamma(k) & \mathcal{R}_{xx}(k)F^{-1}(k) \end{bmatrix} \begin{bmatrix} \underline{v}(k) \\ \underline{x}(k+1) \end{bmatrix} = \begin{bmatrix} \underline{w}_v(k) \\ \underline{w}_x(k) \end{bmatrix}$$

QR Factorize:

$$\begin{bmatrix} \bar{R}_{vv}(k) & \bar{R}_{vx}(k+1) \\ 0 & \bar{R}_{xx}(k+1) \end{bmatrix} = T_a(k) \begin{bmatrix} R_{vv}(k) & 0 \\ -R_{xx}(k)F^{-1}(k)\Gamma(k) & R_{xx}(k)F^{-1}(k) \end{bmatrix}$$

Thus:

$$\begin{bmatrix} \bar{\xi}_v(k) \\ \bar{\xi}_x(k+1) \end{bmatrix} = T_a(k) \begin{bmatrix} 0 \\ \xi_x(k) + R_{xx}(k)[F^{-1}(k)\underline{f}(k, \hat{\underline{x}}(k), \underline{v}(k), 0) - \hat{\underline{x}}(k)] \end{bmatrix}$$

and:

$$\begin{bmatrix} \bar{\underline{w}}_v(k) \\ \bar{\underline{w}}_x(k+1) \end{bmatrix} = T_a(k) \begin{bmatrix} \underline{w}_v(k) \\ \underline{w}_x(k) \end{bmatrix}$$

left multiply original eqn by $T_a(k)$

$$\begin{bmatrix} \bar{\xi}_v(k) \\ \bar{\xi}_x(k+1) \end{bmatrix} = \begin{bmatrix} \bar{R}_{vv}(k) & \bar{R}_{vx}(k+1) \\ 0 & \bar{R}_{xx}(k+1) \end{bmatrix} \begin{bmatrix} \underline{v}(k) \\ \underline{x}(k+1) \end{bmatrix} + \begin{bmatrix} \bar{\underline{w}}_v(k) \\ \bar{\underline{w}}_x(k) \end{bmatrix}$$

$$\bar{\xi}_x(k+1) = \bar{R}_{xx}(k+1)\underline{x}(k+1) + \bar{\underline{w}}_x(k)$$

Measurement Update:

$$\underline{z}(k+1) = \underline{h}(k+1, \bar{\underline{x}}(k+1)) + H(k+1)[\underline{x}(k+1) - \bar{\underline{x}}(k+1)] + \underline{w}(k+1)$$

$$\underline{z}_a(k+1) = R_a^{-T}(k+1)\underline{z}(k+1)$$

$$\underline{z}_a(k+1) = R_a^{-T}[\underline{h}(k+1, \bar{\underline{x}}(k+1)) + H(k+1)[\underline{x}(k+1) - \bar{\underline{x}}(k+1)] + \underline{w}(k+1)]$$

$$\begin{bmatrix} \bar{\xi}_x(k+1) \\ \underline{z}_a(k+1) - R_a^{-T}[\underline{h}(k+1, \bar{\underline{x}}(k+1)) + H(k+1)\bar{\underline{x}}(k+1)] \end{bmatrix} = \begin{bmatrix} \bar{R}_{xx}(k+1) \\ R_a^{-T}H(k+1) \end{bmatrix} \underline{x}(k+1) + \dots \begin{bmatrix} \bar{\underline{w}}_x(k) \\ R_a^{-T}\underline{w}(k+1) \end{bmatrix}$$

$$\begin{bmatrix} \underline{\xi}_x(k+1) \\ \underline{z}_a(k+1) - \underline{h}_a(k+1, \bar{\underline{x}}(k+1)) - H_a(k+1)\bar{\underline{x}}(k+1) \end{bmatrix} = \ldots$$

$$\begin{bmatrix} \bar{R}_{xx}(k+1) \\ H_a(k+1) \end{bmatrix} \underline{x}(k+1) + \begin{bmatrix} \underline{w}_x(k) \\ \underline{w}_a(k+1) \end{bmatrix}$$

$$\underline{h}_a(k+1, \bar{\underline{x}}(k+1)) = R_a^{-T}\underline{h}(k+1, \bar{\underline{x}}(k+1))$$

$$H_a(k+1) = R_a^{-T} H(k+1)$$

$$\underline{w}_a(k+1) = R_a^{-T}\underline{w}(k+1)$$

QR Factorize:

$$\begin{bmatrix} R_{xx}(k+1) \\ 0 \end{bmatrix} = T_b(k+1)\begin{bmatrix} \bar{R}_{xx}(k+1) \\ H_a(k+1) \end{bmatrix}$$

Thus:

$$\begin{bmatrix} \underline{\xi}_x(k+1) \\ \underline{\xi}_r(k+1) \end{bmatrix} = T_b(k+1)\begin{bmatrix} \bar{\underline{\xi}}_x(k+1) \\ \underline{z}_a(k+1) - \underline{h}_a(k+1, \bar{\underline{x}}(k+1)) - H_a(k+1)\bar{\underline{x}}(k+1) \end{bmatrix}$$

And:

$$\begin{bmatrix} \underline{w}_x(k+1) \\ \underline{w}_r(k+1) \end{bmatrix} = T_b(k+1)\begin{bmatrix} \bar{\underline{w}}_x(k+1) \\ \underline{w}_a(k+1) \end{bmatrix}$$

left multiply by $T_b(k+1)$:

$$\begin{bmatrix} \underline{\xi}_x(k+1) \\ \underline{\xi}_r(k+1) \end{bmatrix} = \begin{bmatrix} R_{xx}(k+1) \\ 0 \end{bmatrix} \underline{x}(k+1) + \begin{bmatrix} \underline{w}_x(k+1) \\ \underline{w}_r(k+1) \end{bmatrix}$$

Recapitulating for Filtering:

① Propagate:

$$\begin{bmatrix} \bar{R}_{vv}(k) & \bar{R}_{vx}(k+1) \\ 0 & \bar{R}_{xx}(k+1) \end{bmatrix} = T_a(k) \overbrace{\begin{bmatrix} R_{vv}(k) & 0 \\ -R_{xx}(k)F^{-1}(k)\Gamma(k) & R_{xx}(k)F^{-1}(k) \end{bmatrix}}^{qr()}$$

$$\begin{bmatrix} \bar{\xi}_v(k) \\ \bar{\xi}_x(k+1) \end{bmatrix} = T_a(k) \begin{bmatrix} 0 \\ \xi_x(k) + R_{xx}(k)\left[F^{-1}(k)\,\underline{f}(k, \hat{\underline{x}}(k), \underline{u}(k), 0) - \hat{\underline{x}}(k)\right] \end{bmatrix}$$

② Measurement Update:

$$\begin{bmatrix} R_{xx}(k+1) \\ 0 \end{bmatrix} = T_b(k+1) \overbrace{\begin{bmatrix} \bar{R}_{xx}(k+1) \\ H_a(k+1) \end{bmatrix}}^{qr()}$$

$$\begin{bmatrix} \xi_x(k+1) \\ \xi_r(k+1) \end{bmatrix} = T_b(k+1) \begin{bmatrix} \bar{\xi}_x(k+1) \\ \underline{z}_a(k+1) - \underline{h}_a(k+1, \bar{\underline{x}}(k+1)) - H_a(k+1)\bar{\underline{x}}(k+1) \end{bmatrix}$$

where:

$$\hat{\underline{x}}(k) = R_{xx}^{-1}(k)\,\underline{\xi}_x(k) \qquad P(k+1) = R_{xx}^{-1}(k+1)R_{xx}^{-T}(k+1)$$

$$\bar{\underline{x}}(k+1) = \bar{R}_{xx}^{-1}(k+1)\,\bar{\underline{\xi}}_x(k+1)$$

$$\hat{\underline{x}}(k+1) = R_{xx}^{-1}(k+1)\,\underline{\xi}_x(k+1)$$

$$F(k) = \left.\frac{d\underline{f}}{d\underline{x}(k)}\right|_{[k, \hat{\underline{x}}(k), \underline{u}(k), 0]} \qquad \Gamma(k) = \left.\frac{d\underline{f}}{d\underline{v}(k)}\right|_{[k, \hat{\underline{x}}(k), \underline{u}(k), 0]}$$

$$H(k) = \left.\frac{d\underline{h}}{d\underline{x}(k)}\right|_{\underline{x}(k)} \qquad H_a(k) = \mathrm{chol}(R(k))^{-T} H(k)$$

$$\underline{h}_a(k) = \mathrm{chol}(R(k))^{-T}\,\underline{h}(k)$$

$$\underline{z}_a(k) = \mathrm{chol}(R(k))^{-T}\,\underline{z}(k)$$

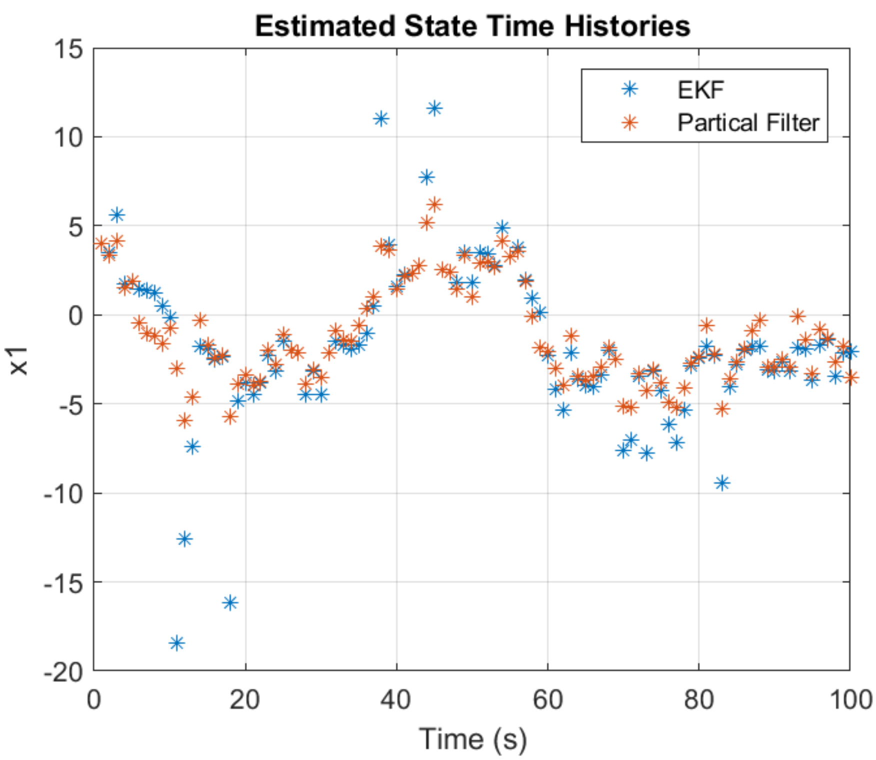
HW8 Problem 4 Final

Spencer Freeman

AOE 5784

12/17/2024

Results (based in part on random draw):



Script hw8_prob34_final.m (includes local functions defined at the bottom):

```
%% Solve the particle filtering example problem that was presented in lecture. Re-do Problem 3 using an extended Kalman filter
% Spencer Freeman, 12/17/2024
% AOE 5784, Estimation and Filtering
%
% This script solves number 4 of problem set 8
% -----------------------------------------------------------------------
clear;clc;close all

disp('HW8-P4_final')

%% load data
load('measdata_pfexample.mat')

n = length(zkhist); % samples
nx = length(xhat0);
nv = size(Q, 1);
thist = 1:n;

%% filter the data
[xhathist_pft,Phist,sigmahist,enuhist] = ...
   particle_filter(zkhist, xhat0, P0, Q, R);
```

```
[xhathist_ekf,Phist,sigmahist,enuhist] = ...
   efk(zkhist, xhat0, P0, Q, R);

%% plotting
close all

% time histories
names = "x1";

fig = figure;
fig.WindowStyle = 'Docked';
for i = 1:nx
   subplot(nx, 1, i)
   plot(thist, xhathist_ekf(:, i), '*'); hold on; grid on
   plot(thist, xhathist_pft(:, i), '*'); hold on; grid on
   ylabel(names(i))
   if i == 1
      title('Estimated State Time Histories')
      legend('EKF', 'Partical Filter')
   end % if
end % for
xlabel('Time (s)')
grid on


%% functions

% Particle Filter ----------------------------------------------------------
function [xhathist,Phist,sigmahist,enuhist] = ...
   particle_filter(zkhist, xhat0, P0, Q, R)

n = length(zkhist); % samples
nx = length(xhat0);
nv = size(Q, 1);
thist = 1:n;

t = 0; % s
xhat = xhat0; % initial state estimate
phat = P0; % initial state covariance
ev = 0;

ts = nan(1, n);
xhats_pft = nan(nx, n);
phats_pft = nan(nx * nx, n);
evs = nan(1, n);

Rinv = inv(R);

Ns = 400; % # of particles

w0 = 1/Ns * ones(1, Ns); % initial weights
chi0 = chol(P0)'*randn(nx, Ns) + xhat0; % initial particles

w = w0;
chi = chi0;
for i = 1:n

   ts(i) = t;
   xhats_pft(:, i) = xhat;
   phats_pft(:, i) = phat(:); % unwrap to column vector
   % evs(i) = ev;

   vss = chol(P0)'*randn(nv, Ns);

   z = zkhist(i);

   log_wtil = nan(1, Ns);
   for j = 1:Ns
      chi(:, j) = f_class_example(i, chi(:, j), vss(:, j)); % propagate to k+1
```

```
        log_wtil(j) = log(w(j)) - .5 * (z - h_class_example(chi(:, j)))' * Rinv * (z - h_class_example(chi(:, j)));
        % wtil(j) = w(j) * exp( -.5 * (z - h(chi(:, j)))' * Rinv * (z - h(chi(:, j))) );
    end % for
    log_wtil_max = max(log_wtil);
    wtiltil = exp(log_wtil - log_wtil_max);

    w = wtiltil / sum(wtiltil); % normalized weights

    xhat = sum(w .* chi, 2); % compute a posteriori state estimate
    phat = zeros(nx);
    for j = 1:Ns
        phat = phat + w(j) * (chi(:, j) - xhat)*(chi(:, j) - xhat)'; % compute a posteriori error covariance matrix
    end % for

    % resampling
    c = nan(1, Ns + 1);
    c(1) = 0;
    c(end) = 1 + 10^-10;
    for j = 2:Ns
        c(j) = sum(w(1:j - 1));
    end % for

    chi_new = nan(nx, Ns);
    for l = 1:Ns
        nl = rand;
        ind = find(nl >= c, 1, 'last');
        chi_new(:, l) = chi(:, ind);
    end % for

    chi = chi_new;
    w = w0;

end % for

% record the final filter outputs
ts(n) = t;
xhats_pft(:, n) = xhat;
phats_pft(:, n) = phat(:); % unwrap to column vector

xhathist = xhats_pft';
Phist = [];
sigmahist = [];
enuhist = [];


end % function



% Extended Kalman Filter --------------------------------------------------
function [xhathist,Phist,sigmahist,enuhist] = ...
    efk(zkhist, xhat0, P0, Q, R)

n = length(zkhist); % samples
nx = length(xhat0);
nv = size(Q, 1);
thist = 1:n;

% EKF
t = 0; % s
xhat = xhat0; % initial state estimate
phat = P0; % initial state covariance
ev = 0;

ts = nan(1, n);
vs = nan(1, n);
xhats = nan(nx, n);
phats = nan(nx * nx, n);
evs = nan(1, n);
```

```
for i = 1:(n - 1)

   ts(i) = t;
   xhats(:, i) = xhat;
   phats(:, i) = phat(:); % unwrap to column vector
   evs(i) = ev;

   % propagate
   % tkp1 = thist(i); % s

   % [fprinted, dfprinted_dxk, dfprinted_dvk] = ...
   %     c2dnonlinear(xhat, [], [0; 0], t, tkp1, nRK, fscriptname, true);

   xbar = f_class_example(i, xhat, 0);
   F = 2*sec(xhat)^2; % df / dxk
   GAMMA = 1;

   pbar = F * phat * F' + GAMMA * Q * GAMMA';
   % t = tkp1;

   % measurement update
   zbar = h_class_example(xbar);
   H = 1 + 2*xbar + 3*xbar^2; % dh /dx

   z = zkhist(i);
   v = z - zbar; % innovation
   S = H * pbar * H' + R; Sinv = inv(S);
   W = pbar * H' * Sinv;

   xhat = xbar + W * v;
   phat = pbar - W * S * W';

   ev = v' * Sinv * v; % estimation error statistic

end % for

% record the final filter outputs
ts(n) = t;
xhats(:, n) = xhat;
phats(:, n) = phat(:); % unwrap to column vector
evs(n) = ev;

xhathist = xhats';
Phist = reshape(phats, nx, nx, n);
sigmahist = [];
enuhist = [];



end % function


% nonlinear dynamics function class example -------------------------------
function xkp1 = f_class_example(k, x, v)
xkp1 = 2*atan(x) + .5*cos(pi*k/3) + v;
end % function

% nonlinear measurement function class example ----------------------------
function z = h_class_example(x)
z = x + x.^2 + x.^3;
end % function
```

HW8 Problem 7 Final

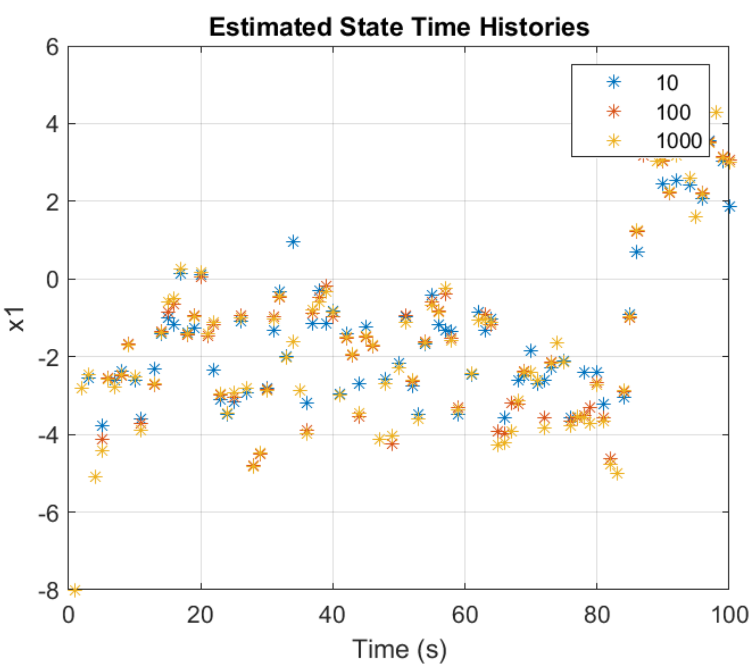
Spencer Freeman

AOE 5784

12/17/2024

Results:

```
HW8-P7_final
10 Particles:

xhathist_10_end =
  1.866917416183279

Phist_10_end =
   1.218938466632424e-12

100 Particles:

xhathist_100_end =
  3.049432998161409

Phist_100_end =
   3.690248926342467e-10

1000 Particles:

xhathist_1000_end =
  3.007122639338923

Phist_1000_end =
   8.297955080856043e-04
```



Script hw8_prob7_final.m (includes local functions defined at the bottom):

```
%% Do fixed-interval particle smoothing on the particle filtering problem of Problem 3
% Spencer Freeman, 12/17/2024
% AOE 5784, Estimation and Filtering
%
% This script solves number 7 of problem set 8
% ----------------------------------------------------------------------
clear;clc;close all
```

```
disp('HW8-P7_final')

format long

%% import data
load('measdata_pfexample02.mat')

n = length(zkhist); % samples
nx = length(xhat0);
nv = size(Q, 1);
thist = 1:n;

%% filter data
Ns = 10;
[xhathist_10,Phist_10_end,sigmahist,enuhist] = ...
   particle_smoother(zkhist, xhat0, P0, Q, R, Ns);

disp('10 Particles:')
xhathist_10_end = xhathist_10(end)
Phist_10_end

Ns = 100;
[xhathist_100,Phist_100_end,sigmahist,enuhist] = ...
   particle_smoother(zkhist, xhat0, P0, Q, R, Ns);

disp('100 Particles:')
xhathist_100_end = xhathist_100(end)
Phist_100_end

Ns = 1000;
[xhathist_1000,Phist_1000_end,sigmahist,enuhist] = ...
   particle_smoother(zkhist, xhat0, P0, Q, R, Ns);

disp('1000 Particles:')
xhathist_1000_end = xhathist_1000(end)
Phist_1000_end

%% plotting
close all

% time histories
names = ["x1"];

fig = figure;
fig.WindowStyle = 'Docked';
for i = 1:nx
   subplot(nx, 1, i)
   plot(thist, xhathist_10(:, i), '*'); hold on; grid on
   plot(thist, xhathist_100(:, i), '*'); hold on; grid on
   plot(thist, xhathist_1000(:, i), '*'); hold on; grid on
   % plot(thist, xhathist_ukf(:, i), '*'); hold on; grid on
   ylabel(names(i))
   if i == 1
      title('Estimated State Time Histories')
      legend('10', '100', '1000')
   end % if
end % for
xlabel('Time (s)')
grid on


% particle smoothing filter ----------------------------------------------
function [xhathist,Phist_end,sigmahist,enuhist] = ...
   particle_smoother(zkhist, xhat0, P0, Q, R, Ns)

n = length(zkhist); % samples
nx = length(xhat0);
nv = size(Q, 1);
```

```
t = 0; % s
xhat = xhat0; % initial state estimate
phat = P0; % initial state covariance
ev = 0;

ts = nan(1, n);
xhats = nan(nx, n);
phats = nan(nx * nx, n);
evs = nan(1, n);

Rinv = inv(R);
Svj = chol(Q)';

for k = 1:n

   ts(k) = t;
   xhats(:, k) = xhat;
   phats(:, k) = phat(:); % unwrap to column vector
   % evs(i) = ev;

   wtil = nan(1, Ns);
   chis = nan(nx, Ns);
   for i = 1:Ns

      chi = chol(P0)'*randn(nx, 1) + xhat0; % initial particle
      for j = 1:k

         vss = Svj * randn(nv, 1);
         chi = f_class_example(j, chi, vss); % propagate
         dz = zkhist(j) - h_class_example(chi);

      end

      wtil(i) = exp( -.5*sum(dz.*Rinv*dz) );
      chis(:, i) = chi; % chi(k)

   end
   w = wtil / sum(wtil);

   xhat = sum(w .* chis, 2); % compute a posteriori state estimate
   phat = zeros(nx);
   for i = 1:Ns
      phat = phat + w(i) * (chis(:, i) - xhat)*(chis(:, i) - xhat)'; % compute a posteriori error covariance matrix
   end % for


end % for

% record the final filter outputs
ts(n) = t;
xhats(:, n) = xhat;
phats(:, n) = phat(:); % unwrap to column vector

xhathist = xhats';
Phist_end = phat;
sigmahist = [];
enuhist = [];


end % function


% nonlinear dynamics function class example -------------------------------
function xkp1 = f_class_example(k, x, v)
xkp1 = 2*atan(x) + .5*cos(pi*k/3) + v;
end % function

% nonlinear measurement function class example ----------------------------
function z = h_class_example(x)
z = x + x.^2 + x.^3;
```

```
end % function
```