ASEN 5070 Final Exam

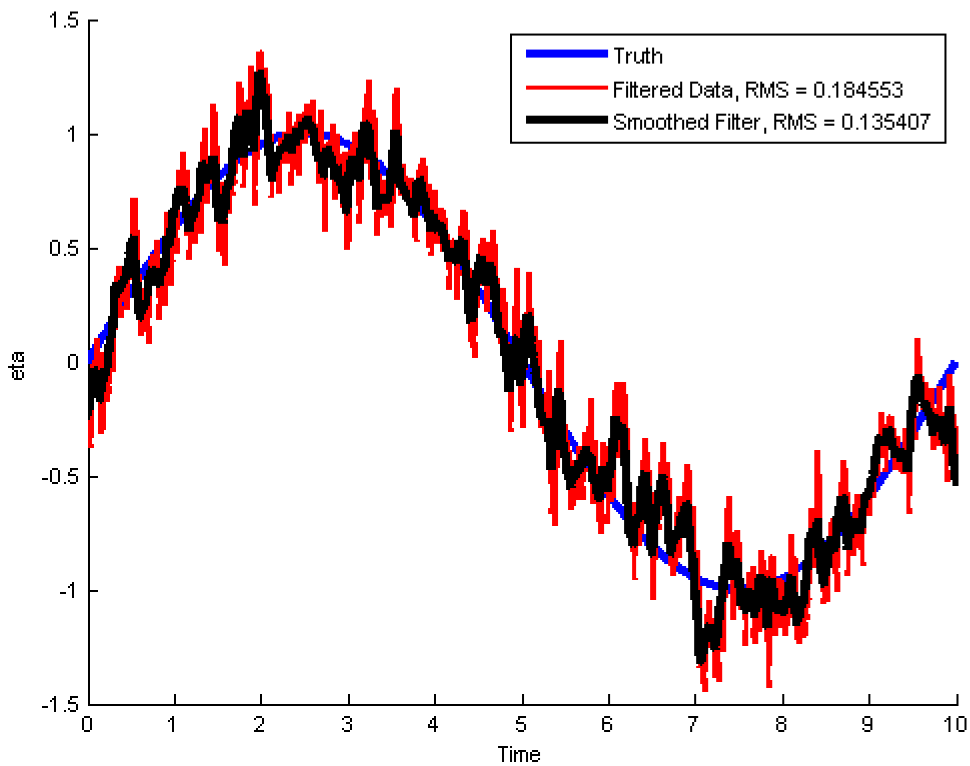
John Clouse

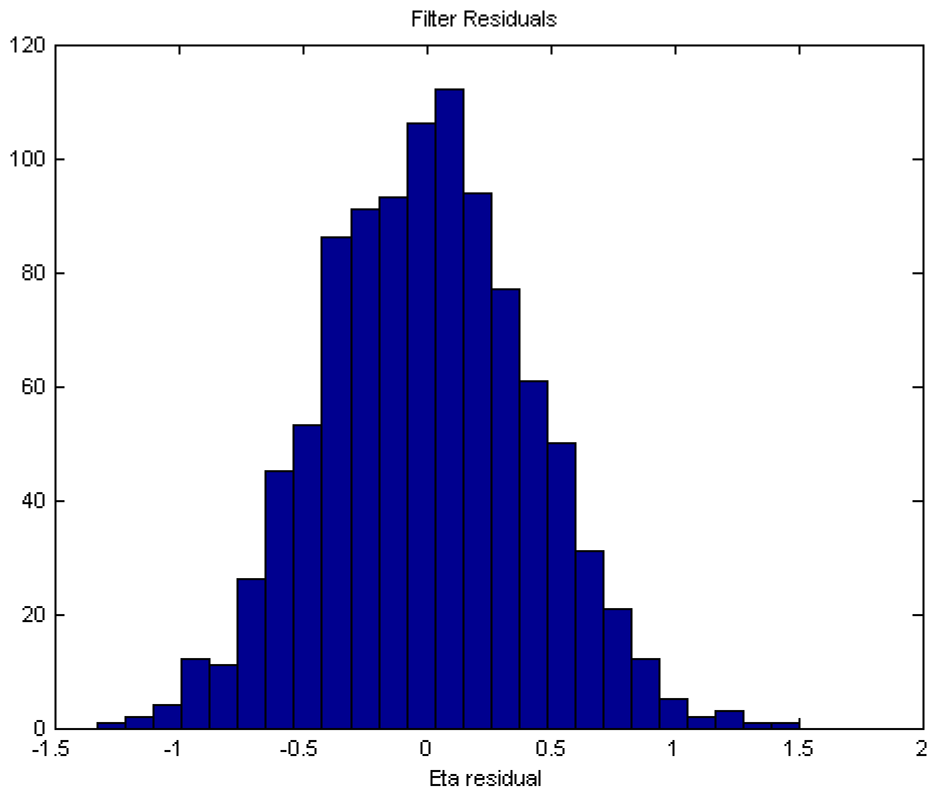
Problem 1

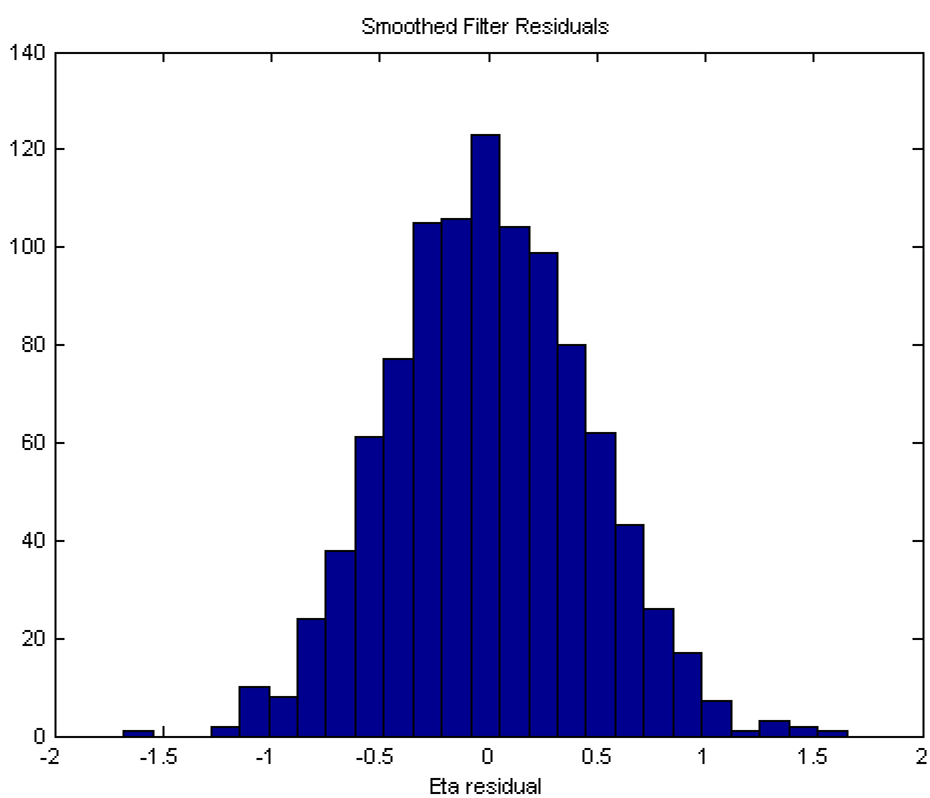
RMS non-smoothed: 0.184553

RMS smoothed: 0.135407

The histogram of the smoothed results is more Gaussian.




Filter Residuals
120
100
80
60
40
20
0
-1.5
-1
-0.5
0
0.5
1
1.5
2
Eta residual

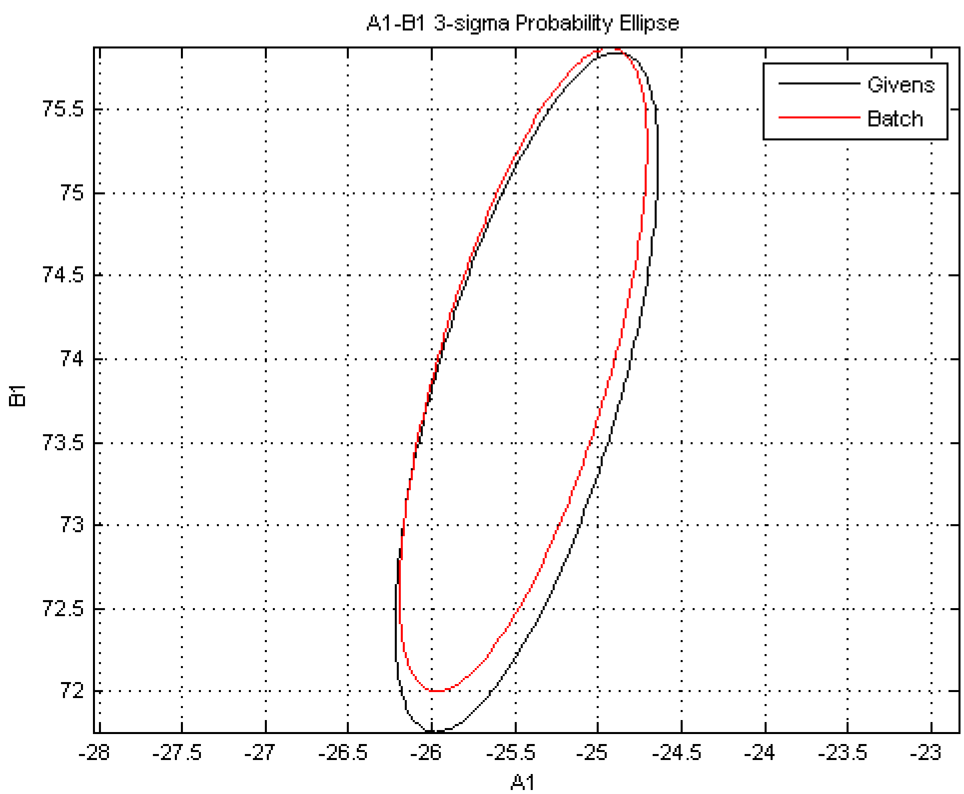

Smoothed Filter Residuals
140
120
100
80
60
40
20
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
Eta residual

Problem 2

a)

X_est_Givens =	std_Givens =
-29.813422329056046	0.392536741587372
0.675708079081207	0.015505826421920
-0.000753693605452	0.000085199406076
0.000000159940177	0.000000139435309
0.000000000067455	0.000000000069609
-25.430570609626745	0.263745050875215
8.003182482083073	0.076466468594406
-13.012860522773781	0.048674557278335
7.948520928074068	0.049855582263566
73.799764999857416	0.680434501336939
-0.362037944501615	0.107102189890218
-12.998727514670593	0.050489587657782
-1.068726601580486	0.051415507758401

b)



c)

X_est_batch =	std_batch =
-29.699450238323188	0.364170480004120
0.671710556400109	0.014661467935057
-0.000735334925939	0.000080686119601
0.000000133026177	0.000000132113834
0.000000000079780	0.000000000065971
-25.451430452206690	0.247904425806435
7.977723424969040	0.071328460758129
-13.018007034361684	0.045305198145460
7.937756185907434	0.044699935025174
73.933962255828277	0.644020992993889
-0.351550158819573	0.101162202187059
-12.999805774111280	0.045918251376173
-1.064354324586364	0.044717730442813

```
>> rel_diff = (X_est_Givens - X_est_batch)./X_est_batch

rel_diff =

   0.003837515166722
   0.005951257789549
   0.024966418519794
   0.202321078640794
  -0.154485619724847
  -0.000819594113546
   0.003191268455654
  -0.000395337901901
   0.001356144219414
  -0.001815096227448
   0.029832970968518
  -0.000082944273124
   0.004107914904956
```

The solutions differ slightly due to the condition of P. The Givens algorithm works on orthogonal transformations of the square root of P (with half of P's condition), while batch works on P itself. Even while batch uses Cholesky decomposition, the Givens algorithm will suffer from less error due to machine precision.

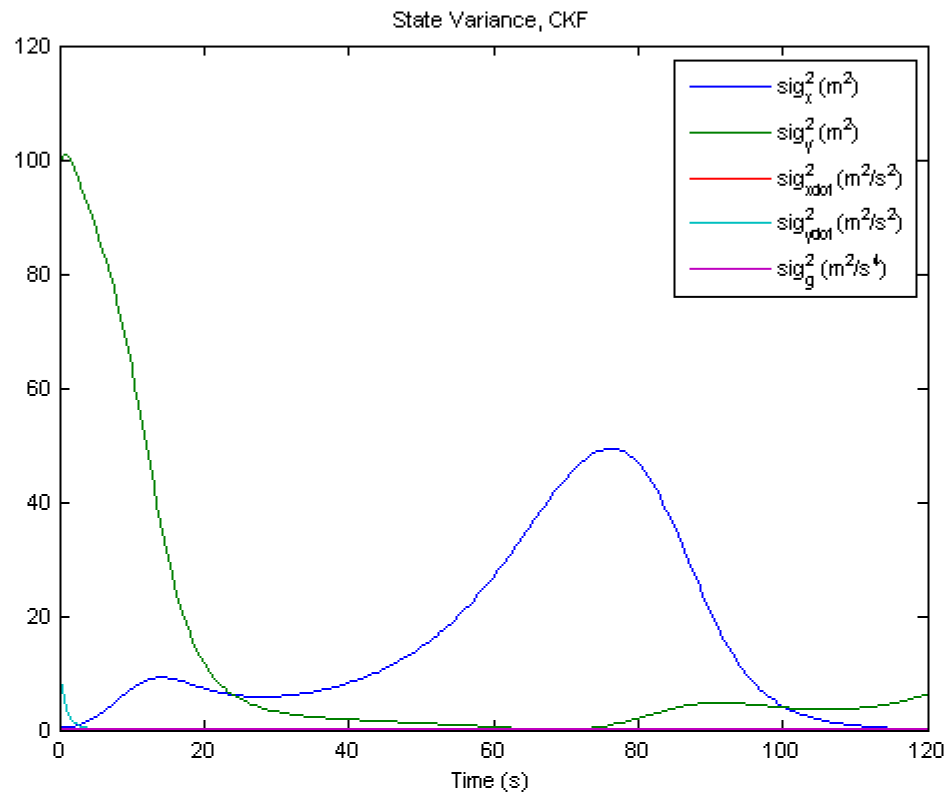
Problem 3

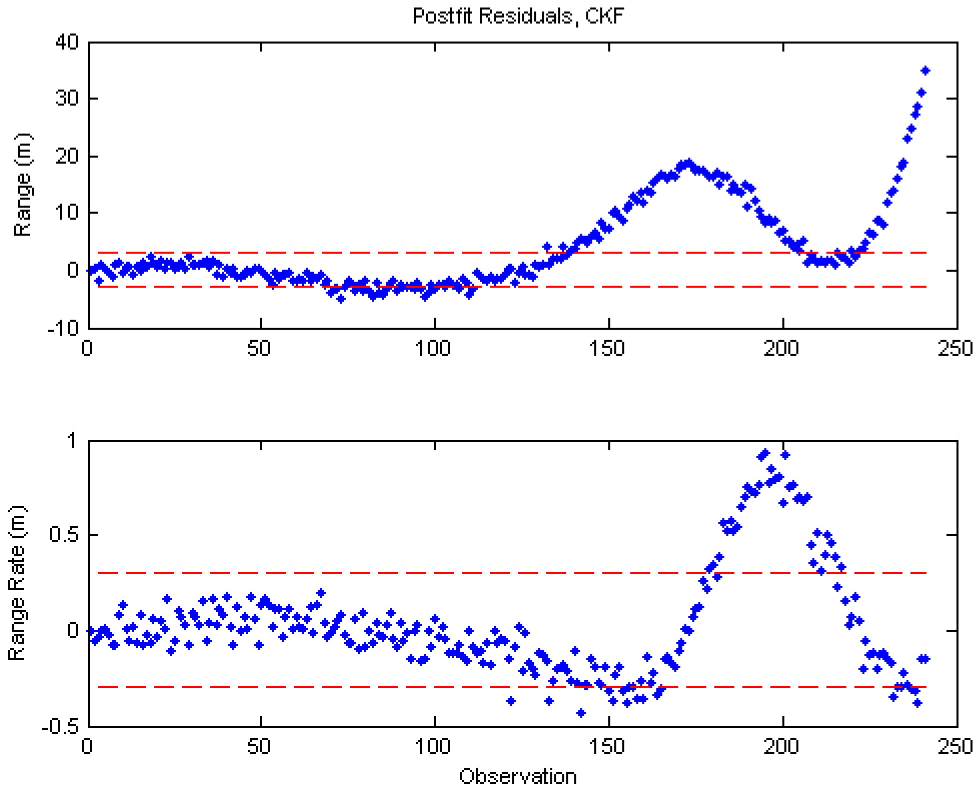
a)

```
final_est_state_CKF =

   1.0e+04 *

   3.583741841897519
   0.177509360885704
   0.030798171501356
  -0.056121375377933
   0.000974015522315
```





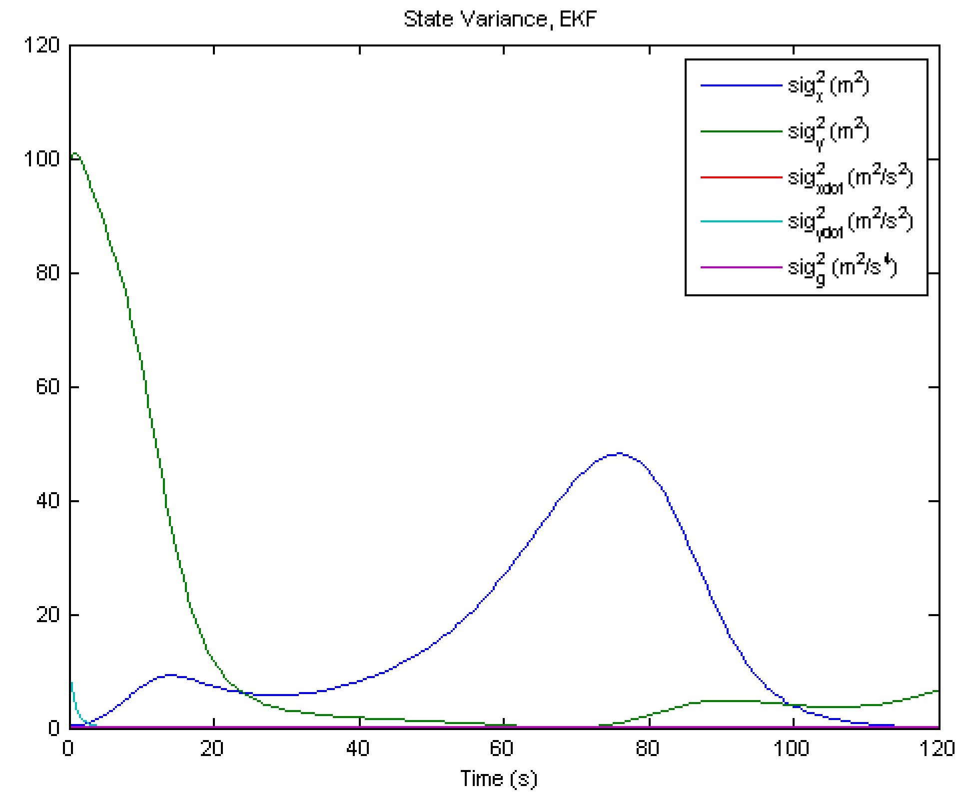
b)

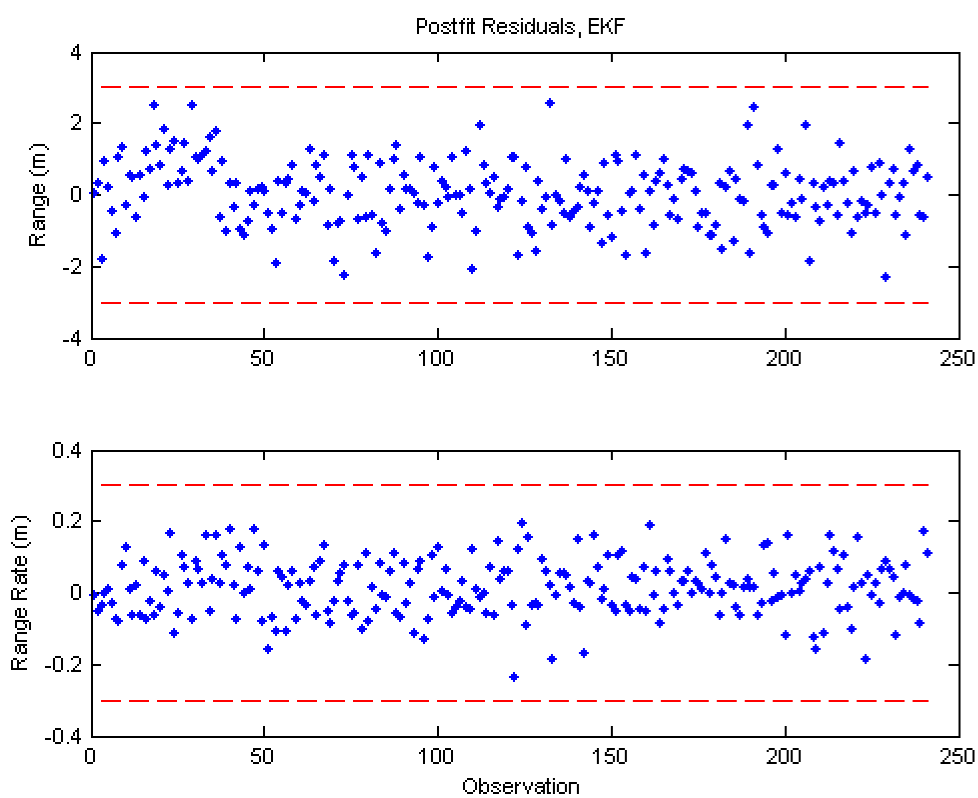
I chose to switch to EKF after 50 observations. The postfit residuals looked converged up to that point in the CKF, and started diverging after that. The covariance was also relatively low at that point (25 seconds).

```
final_est_state_EKF =

   1.0e+04 *

   3.592236384436523
   0.122517117640428
   0.029696318022659
  -0.057787927848967
   0.000980000489809
```


State Variance, EKF
sig$_x^2$ (m^2)
sig$_y^2$ (m^2)
sig$_{xdot}^2$ (m^2/s^2)
sig$_{ydot}^2$ (m^2/s^2)
sig$_g^2$ (m^2/s^4)
Time (s)


Postfit Residuals, EKF
Range (m)
Range Rate (m)
Observation

c)

The estimated states differ because the EKF's reference trajectory stays closer to the true trajectory, keeping linearity errors low. CKF's reference trajectory resembles truth less and less as time goes on , as seen in the postfit residuals. Its deviation gets large because the linearity assumption between reference and truth breaks down.

The covariances are nearly the same. Their (small) differences arise from the H matrix being calculated by reference-trajectory components. They are small differences because the range and range-rate measurements aren't affected much over the course of the observations:

```
>> (CKF_final_calc_range - EKF_final_calc_range)/EKF_final_calc_range

ans =

    -4.540464556983306e-06
```

Finally the postfit residuals for EKF are better than CKF (3-sigma of measurement error). This is because the *a priori* state deviation is zero during the processing of each observation for the EKF. The CKF *a priori* state deviation gets farther from the reference trajectory as time goes on.

Final Exam Problem 1

Table of Contents

Initialize

```
clearvars -except function_list pub_opt P_joseph_store
global function_list;
function_list = {};
close all

obs_data = load('hw11.dat');

T = 10;
truth = sin(obs_data(:,1) * 2*pi/T);
main_plot = figure;
hold on
% plot(obs_data(:,1),obs_data(:,2))
plot(obs_data(:,1),truth,'LineWidth',3)

eta0_ap = 0;
P0_ap = 1;
R = 1;
Q = 1;
num_obs = length(obs_data(:,1));
eta_est_store = zeros(num_obs,1);

eta_est = eta0_ap;
P = P0_ap;

tc_vec = [1/0.045];
sig_vec = [2.49];

best_RMS = -1;
for idx = 1:length(tc_vec)
time_const = tc_vec(idx);
beta = 1/time_const;
sigma = sig_vec(idx);
eta_est_store_inner = zeros(num_obs,1);
RMS_accum = 0;
for ii = 1:num_obs
    % STM
    if ii == 1 %measurement at t = 0
        m = 1;
    else
        m = exp(-beta*(obs_data(ii,1)-obs_data(ii-1,1)));
```

```
    end
    STM = m;

    % Time Update
    eta_ap = STM*eta_est;
    gamma = sqrt(sigma*sigma/2/beta*(1-m*m));
    P_ap = STM*P*STM + gamma*Q*gamma;

    % Kalman gain
    K = P_ap/(P_ap+1); % valid for this 1D case

    % Measurement Update
    Y = obs_data(ii,2);
    eta_est = eta_ap +K*(Y-eta_ap); %H~ == 1 in this case.
    P = K;

    eta_est_store_inner(ii) = eta_est;
    RMS_accum = RMS_accum + (truth(ii)-eta_est)*(truth(ii)-eta_est);

    % Stores
    P_store(ii) = P;
    STM_store(ii) = STM;
    gamma_store(ii) = gamma;
end
RMS = sqrt(RMS_accum/num_obs);
fprintf(sprintf('RMS for tau=%f, sigma=%f: %f\n',time_const, sigma, RMS));
if best_RMS == -1
    best_RMS = RMS;
    eta_est_store = eta_est_store_inner;
else
    if RMS < best_RMS
        eta_est_store = eta_est_store_inner;
    end
    best_RMS = min(best_RMS,RMS);
end
end
plot(obs_data(:,1),eta_est_store,'r','LineWidth',2)
```

Smoothing

```
smoothed_store = zeros(num_obs,1);
smoothed_store(end) = eta_est_store(end);
for ii = num_obs-1:-1:1
    P = P_store(ii);
    STM = STM_store(ii+1);
    gamma_store(ii+1);
    eta_est = eta_est_store(ii);
    eta_last = smoothed_store(ii+1);
    S = P*STM/(STM*P*STM + gamma*Q*gamma);
    eta_est_new = eta_est + S*(eta_last-STM*eta_est);
    smoothed_store(ii) = eta_est_new;
end
figure(main_plot);
```

```
plot(obs_data(:,1),smoothed_store,'k','LineWidth',3)
RMS_smooth = sqrt(sum((truth-smoothed_store).*(truth-smoothed_store))...
    /num_obs);
legend('Truth', sprintf('Filtered Data, RMS = %f',best_RMS),...
    sprintf('Smoothed Filter, RMS = %f',RMS_smooth))
xlabel('Time'), ylabel('eta')
fprintf(sprintf('Smoothed RMS: %f\n',RMS_smooth));
fprintf('The histogram of the smoothed results is more Gaussian.\n')
figure
hist(obs_data(:,2)-eta_est_store,25);
title('Filter Residuals')
xlabel('Eta residual')
figure
hist(obs_data(:,2)-smoothed_store,25);
title('Smoothed Filter Residuals')
xlabel('Eta residual')
```

Published with MATLAB® R2013b

Final Exam Problem 2

Table of Contents

Initialize

```
clearvars -except function_list pub_opt
global function_list;
function_list = {};
close all

w1 = 2*pi/709;
w2 = 2*pi/383;
w3 = 2*pi/107;
w4 = 2*pi/13;

H = @(t) [1 t t*t t*t*t t*t*t*t cos(w1*t) cos(w2*t) cos(w3*t) cos(w4*t) ...
    sin(w1*t) sin(w2*t) sin(w3*t) sin(w4*t)];

num_state = 13;
obs_data = load('P2_obs.txt');
y = obs_data(:,2);
t = obs_data(:,1);
num_obs = length(y);
H_mat = zeros(num_obs,num_state);
for ii = 1:num_obs
    % t = 0:99
    H_mat(ii,:) = H(t(ii));

end

x_ap = zeros(num_state,1);
P_ap = eye(num_state)*100;
L = chol(P_ap,'lower');
R_ap = eye(num_state)/(L);
b_ap = R_ap*x_ap;

W = 1; % 1/(measurements w/ unit variance)
W_sqrt = sqrt(W);

big_mat = [R_ap; H_mat];
[rows, cols] = size(big_mat);
Q = eye(rows);
R = big_mat;
I = eye(rows);
for ii = 1:cols
```

```
    ii
    for jj = ii+1:rows
        G = I;
%         theta = atan2(R(jj,ii),R(jj-1,ii));
%         S = sin(theta);
        temp = sqrt(R(ii,ii)*R(ii,ii)+R(jj,ii)*R(jj,ii));
        S = R(jj,ii)/temp;
%         C = cos(theta);
        C = R(ii,ii)/temp;
        G(ii,ii) = C;
        G(jj,ii) = -S;
        G(ii,jj) = S;
        G(jj,jj) = C;

        R(ii,:) = [C S]*[R(ii,:);R(jj,:)];
        R(jj,:) = [-S C]*[R(ii,:);R(jj,:)];
        Q(ii,:) = [C S]*[Q(ii,:);Q(jj,:)];
        Q(jj,:) = [-S C]*[Q(ii,:);Q(jj,:)];

%         R = G*R;
%         Q = G*Q;
    end
end
R = R(1:num_state,1:num_state);

b_e = Q*W_sqrt*[b_ap;y];
b = b_e(1:num_state);

x_est_Givens = R\b;
P_Givens = eye(num_state)/R/R';
P_G_diag = diag(P_Givens);
% A1 is index 6, B1 is 10
A1 = 6; B1 = 10;
P_A1_B1 = [P_G_diag(A1) P_Givens(A1,B1); P_Givens(A1,B1) P_G_diag(B1)];
[evec,ev]=eig(P_A1_B1);
ell_a=3*sqrt(ev(2,2)); % larger eigenvalue
ell_b=3*sqrt(ev(1,1));
angle=atan2(evec(2,2),evec(1,2)); % Using 2nd eigenvector
figure
plot_ellipse(ell_a,ell_b,angle,x_est_Givens(A1),x_est_Givens(B1));
title('A1-B1 3-sigma Probability Ellipse')
xlabel('A1'),ylabel('B1')
```

Batch

Begin batch

```
chol_P0 = chol(P_ap,'lower');
P0_inv = eye(13)/(chol_P0')/(chol_P0);
info_mat = P0_inv;
norm_mat = P0_inv*x_ap;
for ii = 1:num_obs
    % No integration step with constants. STM = I
```

```
    %H
    H_ = H(t(ii));

    % Accumulate information matrix
    info_mat = info_mat + H_'*W*H_;

    % Accumulate normal matrix
    norm_mat = norm_mat + H_'*W*y(ii);

end
x_est_batch = cholesky_linear_solver(info_mat,norm_mat);
chol_info = chol(info_mat,'lower');
P_batch = eye(13)/(chol_info')/(chol_info);
% P_batch = eye(num_state)/info_mat;
P_b_diag = diag(P_batch);

% Ellipsoid
P_A1_B1 = [P_b_diag(A1) P_batch(A1,B1); P_batch(A1,B1) P_b_diag(B1)];
[evec,ev]=eig(P_A1_B1);
ell_a=3*sqrt(ev(2,2)); % larger eigenvalue
ell_b=3*sqrt(ev(1,1));
angle=atan2(evec(2,2),evec(1,2)); % Using 2nd eigenvector
hold on
plot_ellipse(ell_a,ell_b,angle,x_est_batch(A1),x_est_batch(B1),'r');
legend('Givens','Batch')
```

Published with MATLAB® R2013b

Final Exam Problem 3

Table of Contents

Initialize

```
clearvars -except function_list pub_opt
global function_list;
function_list = {};
close all
```

Initial conditions

```
site = [20500;0];
X0_ap = [0 0 300 600 9.8]';
x0_ap = [0 0 0 0 0]';
num_state = length(x0_ap);
P0_ap = eye(num_state);
P0_ap(1,1) = 100;
P0_ap(2,2) = 100;
P0_ap(3,3) = 10;
P0_ap(4,4) = 10;
P0_ap(5,5) = 1e-5;

sig_range = 1;
sig_range_rate = 0.1;
R = [sig_range*sig_range 0; 0 sig_range_rate*sig_range_rate];

Q = zeros(num_state);
Q(1,1) = 1e-4;
Q(2,2) = 1e-4;
Q(3,3) = 1e-3;
Q(4,4) = 1e-3;

b = 1e-4;

obs_data = load('P3_obs.txt');
time = obs_data(:,1);
Y = obs_data(:,2:3);
```

```
num_obs = length(time);
```

Helpful Functions

STM from one obs to another

```
STM_gen = @(X,dt) ...
    [1 0 dt 0 0;...
    0 1 0 dt 0;...
    0 0 1-b 0 0;...
    0 0 0 1-b -dt;...
    0 0 0 0 1];

r_calc = @(X) ...
    sqrt((X(1)-site(1))*(X(1)-site(1)) + (X(2)-site(2))*(X(2)-site(2)));
rr_calc = @(X,rho) ...
    ((X(1)-site(1))*X(3) + (X(2)-site(2))*X(4))/rho;

H1 = @(X,rho) [(X(1)-site(1))/rho, (X(2)-site(2))/rho, 0, 0, 0];
H2 = @(X,rho) [X(3)/rho*((X(1)-site(1))*(X(1)-site(1))/(rho*rho)+1),...
    X(4)/rho*((X(2)-site(2))*(X(2)-site(2))/(rho*rho)+1),...
    (X(1)-site(1))/rho, (X(2)-site(2))/rho, 0];
H_calc = @(X,rho) [H1(X,rho);H2(X,rho)];
```

CKF

```
ref_state_CKF = zeros(num_state,num_obs);
x_est_store_CKF = zeros(num_state,num_obs);
P_diag_store_CKF = zeros(num_state,num_obs);
y_store_CKF = zeros(2,num_obs);
post_res_store_CKF = zeros(2,num_obs);
I = eye(5);
last_time = 0;
for ii = 1:num_obs
    % Time update
    if ii == 1
        % t = 0 here
        x_ap = x0_ap;
        P_ap = P0_ap;
        X = X0_ap;
        ref_state_CKF(:,ii) = X;
    else
        dt = time(ii)-last_time;
        STM = STM_gen(X,dt);
        x_ap = STM*x_est;
        P_ap = STM*P*STM' + Q;
        X = STM*X;
        ref_state_CKF(:,ii) = X;
    end

    rho = r_calc(X);

    y = Y(ii,:)' - [rho; rr_calc(X,rho)];
```

```
    y_store_CKF(:,ii) = y;
    H = H_calc(X,rho);
    K = P_ap*H'/(H*P_ap*H' + R);

    % Measurement Update
    x_est = x_ap + K*(y-H*x_ap);
    P = (I-K*H)*P_ap;
    x_est_store_CKF(:,ii) = x_est;
    P_diag_store_CKF(:,ii) = diag(P);
    post_res_store_CKF(:,ii) = y-H*x_est;
    last_time = time(ii);
end
final_est_state_CKF = X + x_est
```

CKF Plots

```
figure
subplot(2,1,1)
plot(1:num_obs,y_store_CKF(1,:),'.');
title('Prefit Residuals, CKF')
ylabel('Range (m)')
subplot(2,1,2)
plot(1:num_obs,y_store_CKF(2,:),'.');
ylabel('Range Rate (m)'), xlabel('Observation')

figure
subplot(2,1,1)
plot(1:num_obs,post_res_store_CKF(1,:),'.');
hold on
plot(1:num_obs,ones(1,num_obs)*3*sig_range,'r--');
plot(1:num_obs,ones(1,num_obs)*-3*sig_range,'r--');
title('Postfit Residuals, CKF')
ylabel('Range (m)')
subplot(2,1,2)
plot(1:num_obs,post_res_store_CKF(2,:),'.');
hold on
plot(1:num_obs,ones(1,num_obs)*3*sig_range_rate,'r--');
plot(1:num_obs,ones(1,num_obs)*-3*sig_range_rate,'r--');
ylabel('Range Rate (m)'), xlabel('Observation')

figure
plot(ref_state_CKF(1,:),ref_state_CKF(2,:));
hold on
plot(site(1),site(2),'r.','MarkerSize',10)
axis equal
title('Reference Trajectory, CKF')
xlabel('X (m)'), ylabel('Y (m)')
figure
plot(x_est_store_CKF(1,:),x_est_store_CKF(2,:));
title('Estimated Deviation, CKF')
axis equal
xlabel('x (m)'), ylabel('y (m)')
```

```
figure
plot(time,P_diag_store_CKF);
title('State Variance, CKF')
xlabel('Time (s)')
legend('sig_x^2 (m^2)','sig_y^2 (m^2)','sig_{xdot}^2 (m^2/s^2)',...
    'sig_{ydot}^2 (m^2/s^2)','sig_g^2 (m^2/s^4)')
```

EKF

```
ref_state_EKF = zeros(num_state,num_obs);
X_store_EKF = zeros(num_state,num_obs);
P_diag_store_EKF = zeros(num_state,num_obs);
y_store_EKF = zeros(2,num_obs);
post_res_store_EKF = zeros(2,num_obs);
I = eye(5);
last_time = 0;
CKF_obs = 50; %Observations up to here will use CKF
for ii = 1:num_obs
    % Time update
    if ii > CKF_obs
        X = X + x_est;
    end
    if ii == 1
        % t = 0 here
        x_ap = x0_ap;
        P_ap = P0_ap;
        X = X0_ap;
        ref_state_CKF(:,ii) = X;
    else
        dt = time(ii)-last_time;
        STM = STM_gen(X,dt);
        x_ap = STM*x_est;
        P_ap = STM*P*STM' + Q;
        X = STM*X;
        ref_state_CKF(:,ii) = X;
    end

    rho = r_calc(X);

    y = Y(ii,:)' - [rho; rr_calc(X,rho)];
    y_store_CKF(:,ii) = y;
    H = H_calc(X,rho);
    K = P_ap*H'/(H*P_ap*H' + R);

    % Measurement Update
    if ii <= CKF_obs
        x_est = x_ap + K*(y-H*x_ap);
        post_res_store_EKF(:,ii) = y-H*x_est;
        X_store_EKF(:,ii) = X + x_est;
    else
        x_est = K*y;
        X = X + x_est;
        post_res_store_EKF(:,ii) = y-H*x_est;
```

```
        X_store_EKF(:,ii) = X;
    end
    P = (I-K*H)*P_ap;
    P_diag_store_EKF(:,ii) = diag(P);
    last_time = time(ii);
end
final_est_state_EKF = X
```

Plots

```
figure
subplot(2,1,1)
plot(1:num_obs,post_res_store_EKF(1,:),'.');
hold on
plot(1:num_obs,ones(1,num_obs)*3*sig_range,'r--');
plot(1:num_obs,ones(1,num_obs)*-3*sig_range,'r--');
title('Postfit Residuals, EKF')
ylabel('Range (m)')
subplot(2,1,2)
plot(1:num_obs,post_res_store_EKF(2,:),'.');
hold on
plot(1:num_obs,ones(1,num_obs)*3*sig_range_rate,'r--');
plot(1:num_obs,ones(1,num_obs)*-3*sig_range_rate,'r--');
ylabel('Range Rate (m)'), xlabel('Observation')

figure
plot(X_store_EKF(1,:),X_store_EKF(2,:));
hold on
plot(site(1),site(2),'r.','MarkerSize',10)
axis equal
title('Estimated Trajectory, EKF')
xlabel('X (m)'), ylabel('Y (m)')

figure
plot(time,P_diag_store_EKF);
title('State Variance, EKF')
xlabel('Time (s)')
legend('sig_x^2 (m^2)','sig_y^2 (m^2)','sig_{xdot}^2 (m^2/s^2)',...
    'sig_{ydot}^2 (m^2/s^2)','sig_g^2 (m^2/s^4)')
```

Published with MATLAB® R2013b

```
function x = cholesky_linear_solver(Y,N)
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

vlen = length(N);
z = zeros(vlen,1);
x = zeros(vlen,1);
U = chol(Y); % Upper
UT = U';
for ii = 1:vlen
    tmp = N(ii);
    for jj = 1:(ii-1)
        tmp = tmp - UT(ii,jj)*z(jj);
    end
    z(ii) = tmp/UT(ii,ii);
end

for ii = vlen:-1:1
    tmp = z(ii);
    for jj = (ii+1):vlen
        tmp = tmp - U(ii,jj)*x(jj);
    end
    x(ii) = tmp/U(ii,ii);
end
```

Published with MATLAB® R2013b

```
function C=plot_ellipse(a,b,angle,x,y,color)
fcnPrintQueue(mfilename('fullpath')); % Add this code to code app
if nargin > 5
    c = color;
else
    c = 'k';
end

tmp=(0:pi/100:2*pi)';
x_ell_t=a * cos(tmp);
y_ell_t=b * sin(tmp);

x_ell = x_ell_t * cos(angle) - y_ell_t * sin(angle) + x;
y_ell = x_ell_t * sin(angle) + y_ell_t * cos(angle) + y;
C=[cos(angle) -sin(angle); sin(angle) cos(angle)];

plot(x_ell,y_ell,c), axis equal
```

Published with MATLAB® R2013b