Table of Contents

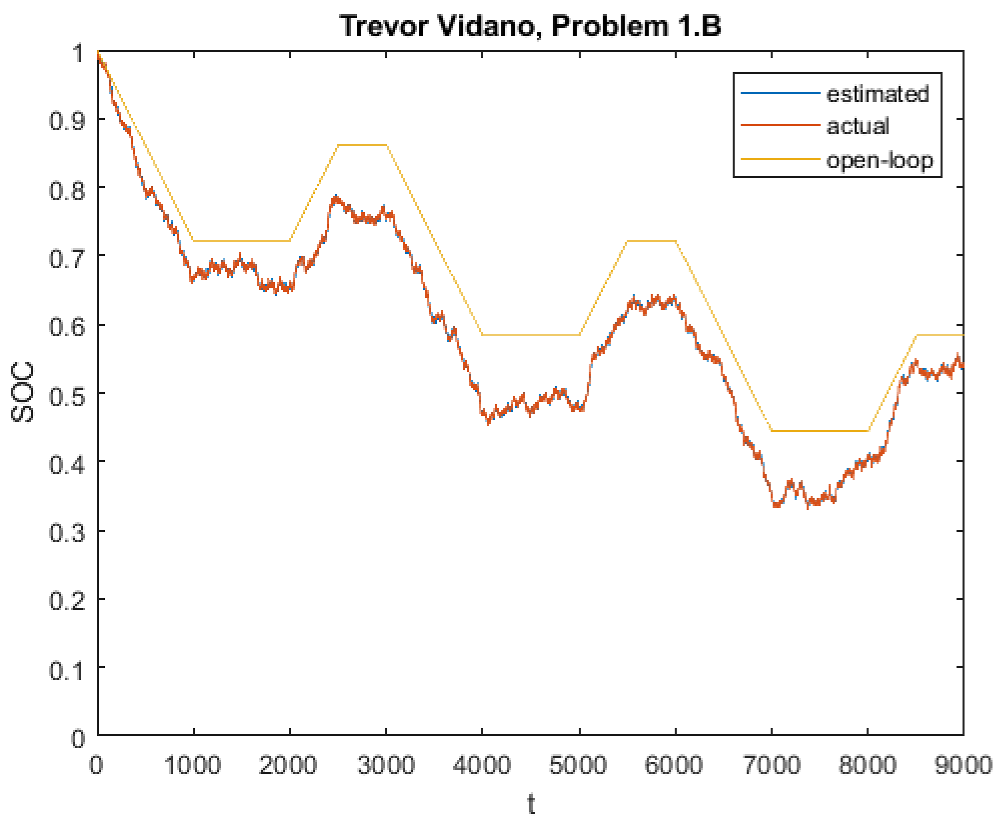
Main MATLAB script for HW 3 in MAE 298: Estimation

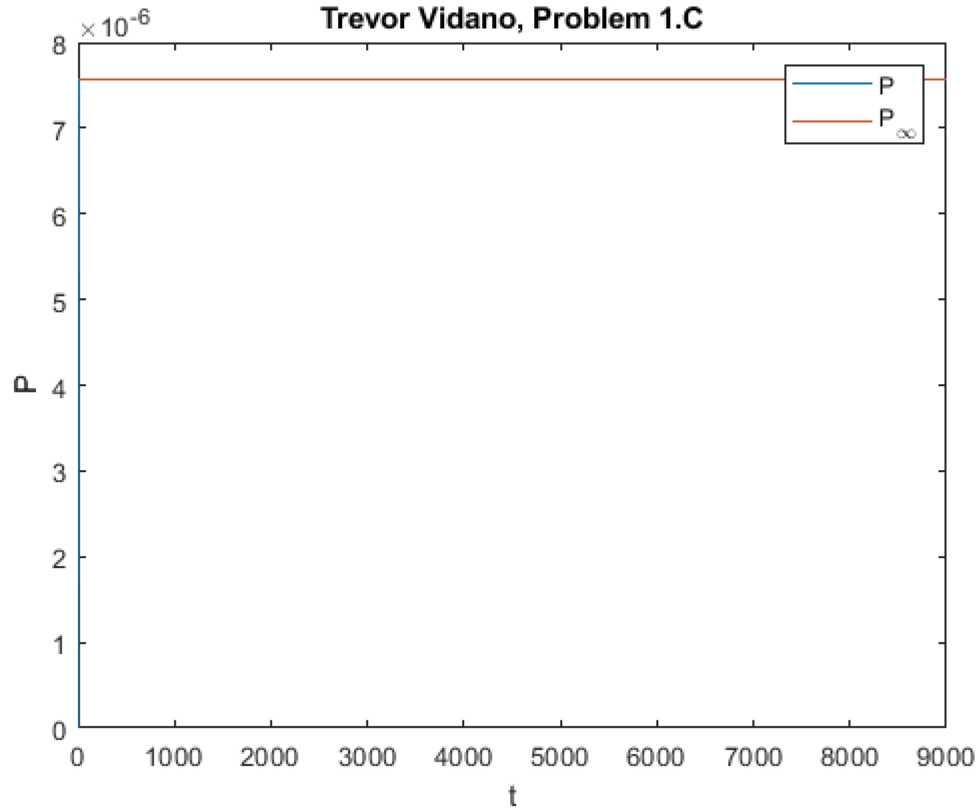
author: Trevor Vidano course: MAE 298: Estimation Spring Quarter 2021 MATLAB implementation of kalman filter and extended kalman filter on battery equivalent circuit model.

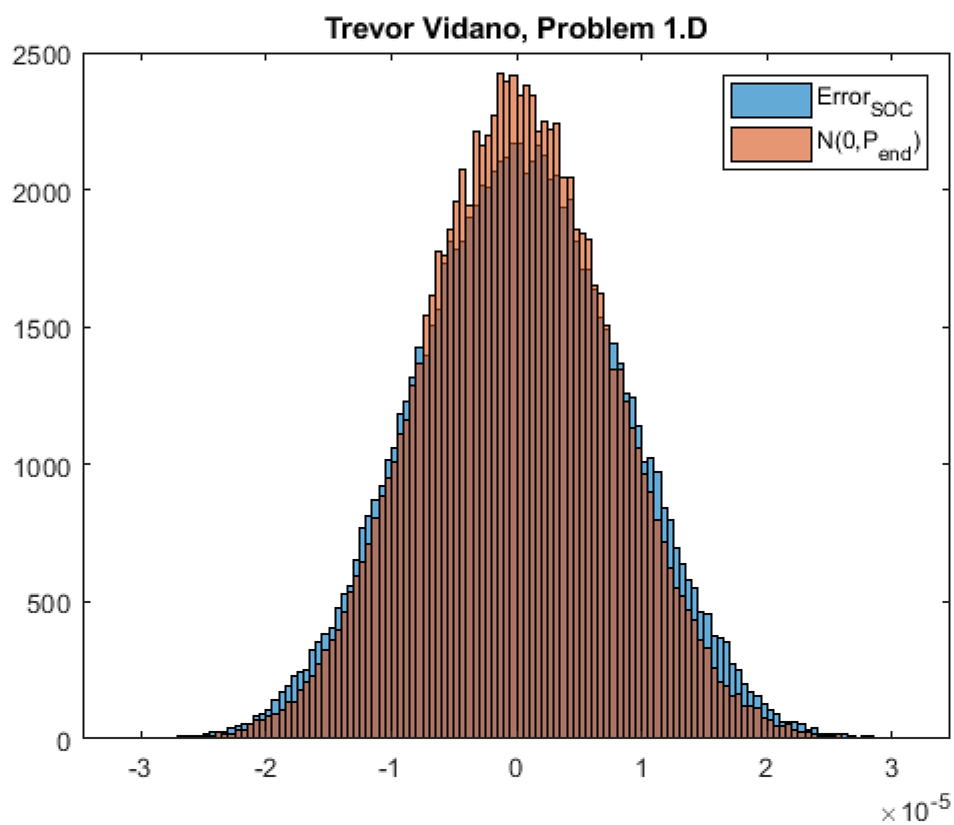
```
clearvars; close all;
% Battery parameters
C_bat= 5*3600;  % A-s
R0 = 0.01; % ohm
Rc = 0.015; % ohm
Cc = 2400; % F
alpha = 0.65; % V
Vocv0 = 3.435; % V

Q = 2.5E-7; % Process noise covariance
R = 1.0E-4; % Sensor noise covariance
Ts = 0.1;  % sampling period

%%%%%%%%%%%%%%%%%%%%%%% State Space %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% Continuous time State Space Matrices
A_c = [-1/Cc/Rc,0;0,0];
B_c = [1/Cc;-1/C_bat];
C_c = [-1,alpha];
D_c = -R0;

% Discrete time State Space Matrics
F = expm(A_c*Ts);
G = [Rc - Rc*exp(-Ts/Cc/Rc); -Ts/C_bat];
H = C_c;
M = D_c;

% Helper functions:
%{
kalman_gain = @(P_priork,H_k,R_k) ...
    P_priork*H_k'*inv(H_k*P_priork*H_k' + R_k);
cov_prior = @(F_k0,P_postk0,Q_k0) ;
cov_post = @(H_k,K_k,P_priork,R_k,size) ...
    (eye(size) - K_k*H_k)*P_priork*(eye(size) - K-k*H_k)' +
 K_k*R_k*K_k';
prior_est = @(F_k0,x_postk0,G_k0,u_k0) F_k0*x_postk0 + G_k0*u_k0;
post_est = @(x_priork,K_k,y_k,H_k) x_priork + K_k*(y_k -
 H_k*x_priork);
```

```
%}
%%%%%%%%%%%%%%%%%%%%%%% Simulation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
linearIV = matfile('IV_data_linear.mat');
u = linearIV.I';
t = linearIV.t';
SOC_act = linearIV.SOC_act';
V_measured = linearIV.V';

k_end = t(end)/Ts;

% scalar initilization:
SOC_post = 1;
SOC_open = SOC_post;
Vc = 0;
P_post = 0;
% matrix initialization:
% x_post(:,1) = [0;1];
% x_open(:,1) = x_post(:,1);
% P_post{1} = eye(2)*0;
for k = 2:k_end
%     % Full system estimation:
%     % Model Predict:
%     P_prior{k} = F*P_post{k-1}*F' + [0,0;0,Q];
%     K(:,k) = P_prior{k}*H'*inv(H*P_prior{k}*H' + R);
%     x_prior(:,k) = F*x_post(:,k-1) + G*u(:,k-1);
%     % Measurement Update:
%     y(:,k) = V_measured(k) - Vocv0;
%     x_post(:,k) = x_prior(:,k) + K(:,k)*(y(:,k) - (H*x_prior(:,k) +
 M*u(:,k)));
% %     P_post{k} = (eye(2) - K(:,k)*H)*P_prior{k}*(eye(2) -
 K(:,k)*H)' + ...
% %                 K(:,k)*R*K(:,k)';
%     P_post{k} = P_prior{k} - P_prior{k}*H'*inv(H*P_prior{k}*H' +
 R)*H*P_prior{k};
%     x_open(:,k) = F*x_open(:,k-1) + G*u(k-1);
%     P(k) = P_post{k}(4);

    % Estimation of only SOC:
    % Model Predict:
    P_prior(k) = 1*P_post(k-1)*1' + Q;
    K(k) = P_prior(k)*alpha/(alpha*P_prior(k)*alpha + R);
    SOC_prior(k) = 1*SOC_post(k-1) - Ts/C_bat*u(k-1);
    Vc(k) = Vc(k-1)*exp(-Ts/Cc/Rc) + Rc*(1 - exp(-Ts/Cc/Rc))*u(k-1);
    % Measurement Update:
    y(k) = V_measured(k) - Vocv0 + Vc(k);
    SOC_post(k) = SOC_prior(k) + K(k)*(y(k) - (alpha*SOC_prior(k) -
 R0*u(k)));
    P_post(k) = (1 - K(k)*alpha)^2*P_prior(k) + K(k)^2*R;

    % Open loop estimate:
    SOC_open(k) = SOC_open(k-1) - Ts/C_bat*u(k-1);
end
```

```
% Compare Differences to gaussian process:
gauss = normrnd(0,P_post(end),[1,length(P_post)]);
SOC_errors = SOC_post - SOC_act;
SOC_errors = SOC_errors*(max(gauss)/max(SOC_errors));

%%%%%%%%%%%%%%%%%%%%%%% Generate Plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
figure();
plot(t,SOC_post,t,SOC_act,t,SOC_open); xlabel('t'); ylabel('SOC');
legend('estimated','actual','open-loop');
title('Trevor Vidano, Problem 1.B');
ylim([0,1.0]);
figure();
plot(t,P_post,[t(1),t(end)],[7.568e-6,7.568e-6]); xlabel('t');
 ylabel('P');
legend('P','P_{\infty}');
title('Trevor Vidano, Problem 1.C');
figure();
histogram(SOC_errors); hold on;
histogram(gauss);
legend('Error_{SOC}','N(0,P_{end})');
title('Trevor Vidano, Problem 1.D');
```



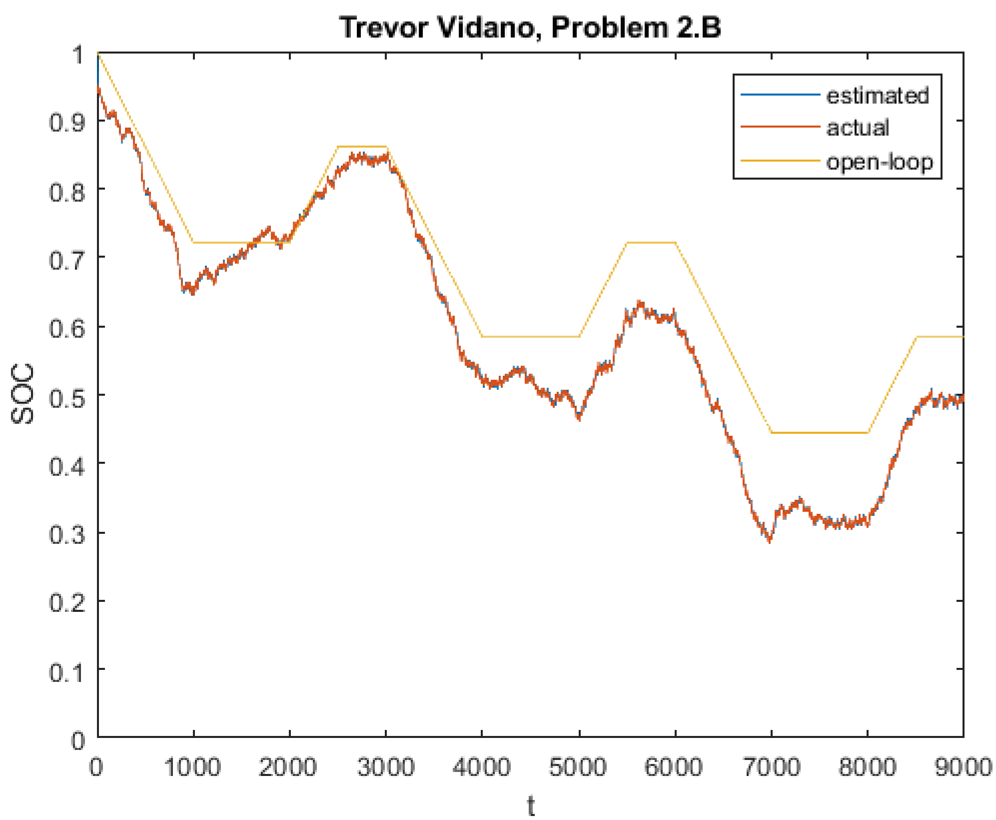

Trevor Vidano, Problem 1.C
×10^-6
8
7
6
5
4
3
2
1
0
P
P
P∞
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
t

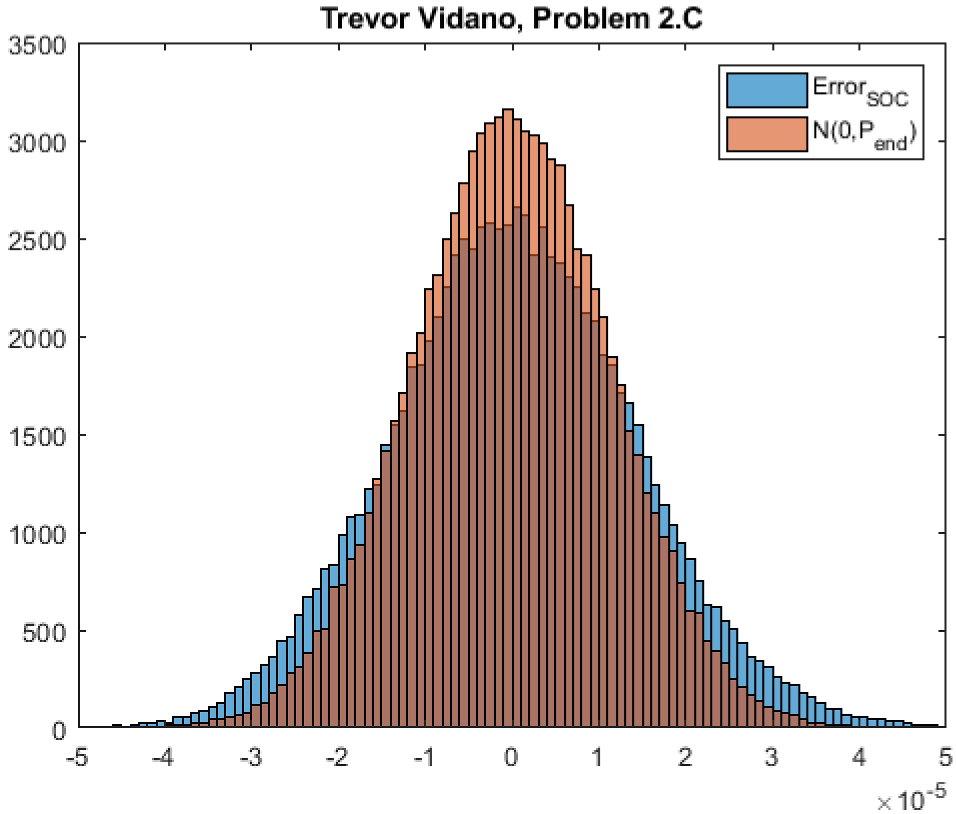

Trevor Vidano, Problem 1.D
Error_SOC
N(0,P_end)
2500
2000
1500
1000
500
0
-3
-2
-1
0
1
2
3
×10^-5

Problem 2: EKF

```
clearvars;
C_bat= 5*3600;  % A-s
R0 = 0.01; % ohm
Rc = 0.015; % ohm
Cc = 2400; % F
alpha = 0.65; % V
Vocv0 = 3.435; % V

nonlinIV = matfile('IV_data_nonlinear.mat');
u_non = nonlinIV.I';
t_non = nonlinIV.t';
SOC_trueNon = nonlinIV.SOC_act';
V_measNon = nonlinIV.V';

VocTable = matfile('OCV_table');
soc_intpts_OCV = VocTable.soc_intpts_OCV;
OCV_intpts = VocTable.OCV_intpts;
dVocTable = matfile('OCV_slope_table');
soc_intpts_OCV_slope = dVocTable.soc_intpts_OCV_slope;
OCV_slope_intpts = dVocTable.OCV_slope_intpts;

Q = 2.5E-7; % Process noise covariance
R = 1.0E-4; % Sensor noise covariance
Ts = 0.1;  % sampling period
k_end = t_non(end)/Ts;
x_post = 1.0;
P_ekf_post = 0.0;
Vc_non = 0.0;
x_open = x_post;
for k = 2:k_end
    % Model Prediction:
    x_prior(k) = x_post(k-1) - Ts/C_bat*u_non(k-1);
    P_ekf_prior(k) = P_ekf_post(k-1) + Q;
    Vc_non(k) = Vc_non(k-1)*exp(-Ts/Cc/Rc) + Rc*(1 - exp(-Ts/Cc/
Rc))*u_non(k-1);

    x_open(k) = x_open(k-1) - Ts/C_bat*u_non(k-1);

    % Measurement Update:
    C_prime(k) =
 interp1(soc_intpts_OCV_slope,OCV_slope_intpts,x_prior(k));
    L(k) = P_ekf_prior(k)*C_prime(k)/(C_prime(k)^2*P_ekf_prior(k) +
 R);
    y_non(k) = V_measNon(k) + Vc_non(k);
    Voc(k) = interp1(soc_intpts_OCV,OCV_intpts,x_prior(k));
    x_post(k) = x_prior(k) + L(k)*(y_non(k) - (Voc(k) -
 R0*u_non(k-1)));
    P_ekf_post(k) = P_ekf_prior(k) - L(k)*C_prime(k)'*P_ekf_prior(k);
end
% Compare Differences to gaussian process:
gauss_non = normrnd(0,P_ekf_post(end),[1,length(P_ekf_post)]);
```

```
x_errors = x_post - SOC_trueNon;
x_errors = x_errors*(min(gauss_non)/min(x_errors));


%%%%%%%%%%%%%%%%%%%%%%% Generate Plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
figure();
plot(t_non,x_post,t_non,SOC_trueNon,t_non,x_open); xlabel('t');
 ylabel('SOC');
legend('estimated','actual','open-loop');
title('Trevor Vidano, Problem 2.B');
ylim([0,1.0]);
figure();
histogram(x_errors); hold on;
histogram(gauss_non); hold off;
xlim([-.5e-4,.5e-4]);
legend('Error_{SOC}','N(0,P_{end})');
title('Trevor Vidano, Problem 2.C');
```

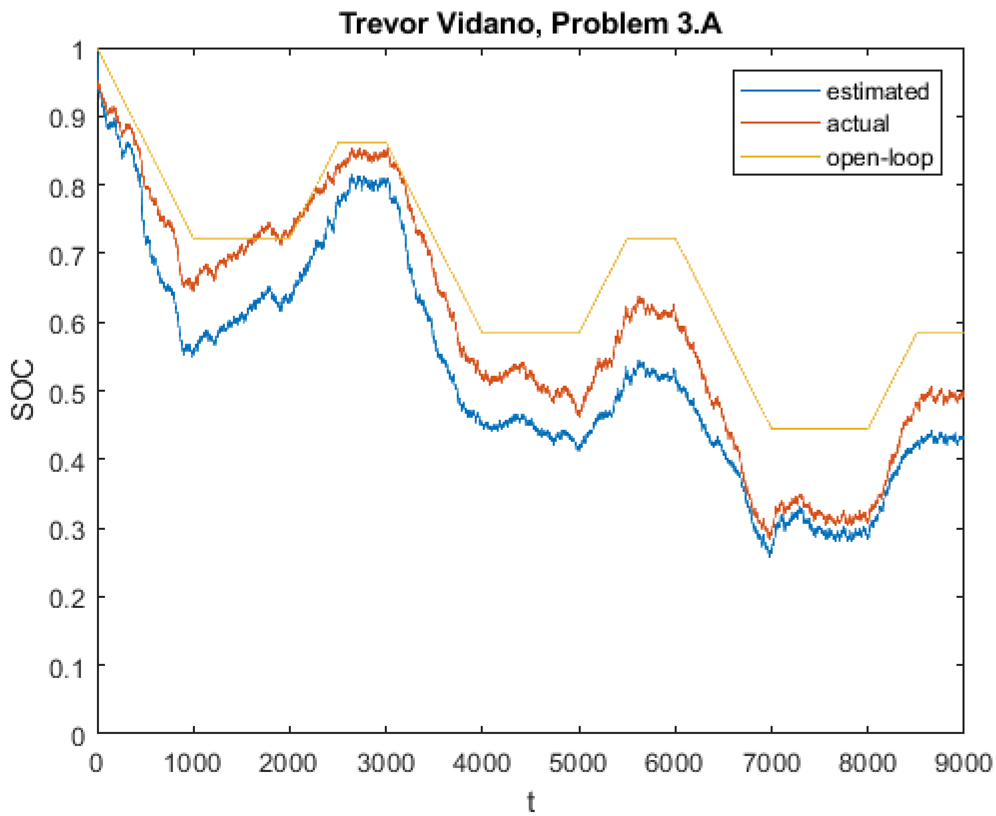


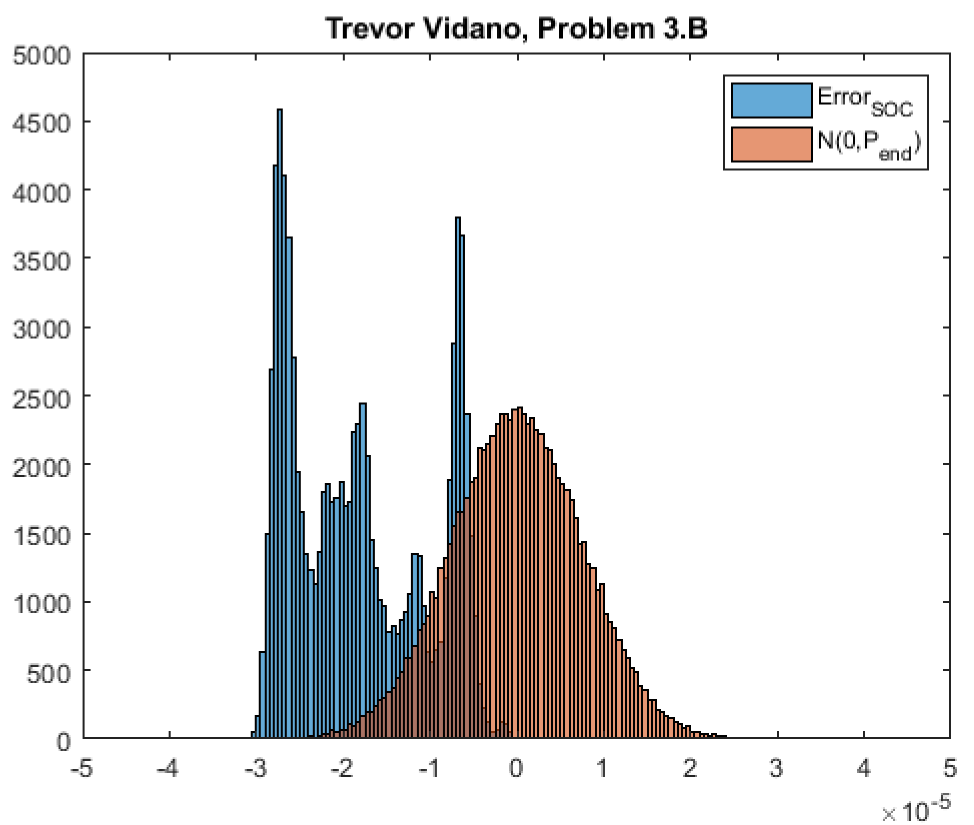


Problem 3: EKF

```
C_bat= 5*3600;  % A-s
R0 = 0.01; % ohm
Rc = 0.015; % ohm
Cc = 2400; % F
alpha = 0.65; % V
Vocv0 = 3.435; % V

nonlinIV = matfile('IV_data_nonlinear.mat');
u_non = nonlinIV.I';
t_non = nonlinIV.t';
SOC_trueNon = nonlinIV.SOC_act';
V_measNon = nonlinIV.V';

VocTable = matfile('OCV_table');
soc_intpts_OCV = VocTable.soc_intpts_OCV;
OCV_intpts = VocTable.OCV_intpts;
dVocTable = matfile('OCV_slope_table');
soc_intpts_OCV_slope = dVocTable.soc_intpts_OCV_slope;
OCV_slope_intpts = dVocTable.OCV_slope_intpts;

Q = 2.5E-7; % Process noise covariance
R = 1.0E-4; % Sensor noise covariance
Ts = 0.1;  % sampling period
```

```
k_end = t_non(end)/Ts;
x_post = 1.0;
P_ekf_post = 0.0;
Vc_non = 0.0;
x_open = x_post;
for k = 2:k_end
    % Model Prediction:
    x_prior(k) = x_post(k-1) - Ts/C_bat*u_non(k-1);
    P_ekf_prior(k) = P_ekf_post(k-1) + Q;
    Vc_non(k) = Vc_non(k-1)*exp(-Ts/Cc/Rc) + Rc*(1 - exp(-Ts/Cc/
Rc))*u_non(k-1);

    x_open(k) = x_open(k-1) - Ts/C_bat*u_non(k-1);

    % Measurement Update:
    L(k) = P_ekf_prior(k)*alpha/(alpha^2*P_ekf_prior(k) + R);
    y_non(k) = V_measNon(k) - Vocv0 + Vc_non(k);
    Voc(k) = interp1(soc_intpts_OCV,OCV_intpts,x_prior(k));
    x_post(k) = x_prior(k) + L(k)*(y_non(k) - (alpha*x_prior(k) -
 R0*u_non(k-1)));
    P_ekf_post(k) = (1 - L(k)*alpha)^2*P_ekf_prior(k) + L(k)^2*R;
end
% Compare Differences to gaussian process:
gauss_non = normrnd(0,P_ekf_post(end),[1,length(P_ekf_post)]);
x_errors = x_post - SOC_trueNon;
x_errors = x_errors*(min(gauss_non)/min(x_errors));

%%%%%%%%%%%%%%%%%%%%%%%%% Generate Plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
figure();
plot(t_non,x_post,t_non,SOC_trueNon,t_non,x_open); xlabel('t');
 ylabel('SOC');
legend('estimated','actual','open-loop');
title('Trevor Vidano, Problem 3.A');
ylim([0,1.0]);
figure();
histogram(x_errors); hold on;
histogram(gauss_non); hold off;
xlim([-.5e-4,.5e-4]);
legend('Error_{SOC}','N(0,P_{end})');
title('Trevor Vidano, Problem 3.B');
```


Trevor Vidano, Problem 3.A
estimated
actual
open-loop
SOC
t
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Trevor Vidano, Problem 3.B
Error$_{SOC}$
N(0,P$_{end}$)
$\times 10^{-5}$
-5
-4
-3
-2
-1
0
1
2
3
4
5
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

Published with MATLAB® R2020b