ESTIMATION OF SOC AND SOH ALGORITHM

This program assumes that the observed battery voltage V is a list of voltage measurements taken at regular intervals. The program uses the Kalman filter algorithm to estimate the state of charge based on these voltage measurements.

The state of charge is estimated using a two-dimensional state vector x, where the first element represents the state of charge and the second element represents the rate of change of the state of charge. The program also uses a two-dimensional covariance matrix P to represent the uncertainty in the state estimate.

The program defines the state transition model A as a two-by-two matrix that describes how the state evolves over time. The program also defines the measurement model C has a one-by-two matrix that maps the state to the observed voltage measurement. The program uses a one-dimensional measurement noise matrix R to represent the uncertainty in the voltage measurement.

The program also defines a two-dimensional process noise matrix Q to represent the uncertainty in the state transition. Finally, the program estimates the state of charge by iterating through the voltage measurements and updating the state estimate using the Kalman filter algorithm.

Estimate the state of charge by using Kalman filter algorithm:

Step-1: initialize the state vector ‘x’ with initial soc value and the covariance matrix ‘p’.

$$X= [SOC, R0, R1, R2] ^T$$

$$P= \text{diag} [1\ 0\ ;0\ 1]$$

Step-2: Measure the battery voltage ‘v’ and the battery current ‘I’.

$$V(t)= ocv(soc)-I(t)R0-v1(t)-v2(t)$$

$$I(t) = ocv(soc)-v1(t)-v2(t)-v(t)/R0$$

Step-3: Use the open circuit voltage (ocv) vs soc curve to estimate the ocv.

Step-4: predict the state vector x(k+1) and its covariance matrix p(k+1) based on the current state vector x(t) and its covariance matrix p(t)

$$X(k+1) = A\ x(k)+Bu(k)=E(k)$$

$$P(k+1) = A\ P(k-1)\ A^T+ Q(k)$$

Step-5: Calculate the Kalman gain k(k+1) based on the predicted state vector and covariance matrix and measurement noise covariance matrix.

$$K(k+1) = p(k+1)\ C^T\ (CP(k+1)\ C^T+R(k+1))\ ^-1$$

Step-6: Update the state vector x(k+1) and its covariance matrix P(k+1) based on the Kalman gain.

$$X^(k/k) = X^(k/k-1)+K(y(k)-CX^(k/K-1))$$

$$P^(k/k) =(I-K(k)C)\ P(k)$$

Step-7: Estimate the battery SOC based on the update state vector x(k+1).

Step-8: Repeat steps 2-6 for subsequent time steps.

SOC program:

```
% Battery model parameters
R0 = 0.1;          % Battery internal resistance (Ohms)
C = 1000;          % Battery capacity (Ah)
alpha = 0.9;       % SoH decay factor

% Measurement noise covariance
Q = diag([0.01, 0.01]);  % Process noise covariance
R = diag([0.05, 0.05]);  % Measurement noise covariance

% Initial estimates and covariances
x_est = [0.5; 0.9];         % Initial estimates of SoC and SoH
P = diag([0.1, 0.1]);       % Initial covariance matrix

% Simulated current and voltage measurements
time = 0:0.1:10;
current = sin(time);        % Simulated current profile
voltage = R0 * current + C * alpha * (1 - exp(-time/alpha)) + randn(size(time))*sqrt(R(2,2));

% Kalman filter estimation
for i = 1:length(time)
    % Prediction step
    x_prd = x_est;                    % State prediction
    P_prd = P + Q;                    % Covariance prediction
    % Update step
    H = [x_prd(1), 0; 0, x_prd(2)];       % Measurement matrix
    z = [voltage(i); current(i)];         % Measurement vector
    S = H * P_prd * H' + R;               % Innovation covariance
    K = P_prd * H' / S;                   % Kalman gain
    x_est = x_prd + K * (z - H * x_prd);  % State estimation
    P = (eye(2) - K * H) * P_prd;         % Covariance estimation
```

```
    % Store the estimated SoC and SoH
    estimated_soc(i) = x_est(1);
    estimated_soh(i) = x_est(2);
end

% Plot the results
figure;
subplot(2,1,1);
plot(time, voltage, 'b', 'LineWidth', 1.5);
hold on;
grid on;
plot(time, R0 * current + C * estimated_soh, 'r--', 'LineWidth', 1.5);
ylabel('Voltage (V)');
legend('Measured Voltage', 'Estimated Voltage');
title('Battery Voltage');

subplot(2,1,2);
plot(time, estimated_soc, 'b', 'LineWidth', 1.5);
hold on;
grid on;
plot(time, ones(size(time))*0.5, 'r--', 'LineWidth', 1.5);
ylabel('State of Charge (SoC)');
xlabel('Time (s)');
legend('Estimated SoC', 'True SoC');
title('Battery State of Charge');
```

Output Graphs: