Implementing SPKF on ESC model

- Refactor SPKF to represent real BMS implementation better
  - □ Initialization routine (`initSPKF.m`), called once at startup
  - □ Update routine (`iterSPKF.m`), called every sample interval
  - □ "Wrapper" code, coordinates the entire simulation process
- Wrapper code begins (load data, store in local variables):

```
% Load cell model, cell-test data (incl. variable "DYNData" of which the field
% "script1" is of interest). It has sub-fields time, current, voltage, soc.
load CellModel            % loads "model" of cell
load('Cell_DYN_P25');     % loads data from cell test
T = 25; % Test temperature

time    = DYNData.script1.time(:);    deltat = time(2)-time(1);
time    = time-time(1); % start time at 0
current = DYNData.script1.current(:); % discharge > 0; charge < 0.
voltage = DYNData.script1.voltage(:);
soc     = DYNData.script1.soc(:);
```

Wrapper code (2)

- Wrapper code continues by reserving storage, initializing covariances, calling SPKF initialization routine

```
% Reserve storage for computed results, for plotting
sochat = zeros(size(soc));
socbound = zeros(size(soc));

% Covariance values
SigmaX0 = diag([1e-3 1e-3 1e-2]); % uncertainty of initial state
SigmaV = 2e-1; % uncertainty of voltage sensor, output equation
SigmaW = 1e1;  % uncertainty of current sensor, state equation

% Create ekfData structure and initialize variables using first
% voltage measurement and first temperature measurement
spkfData = initSPKF(voltage(1),T,SigmaX0,SigmaV,SigmaW,model);
```

Wrapper code (3)

- Wrapper code continues by entering main simulation loop
  - □ "Measure" sensor readings; update SPKF

```
% Now, enter loop for remainder of time, where we update the SPKF
% once per sample interval
hwait = waitbar(0,'Computing...');
for k = 1:length(voltage),
  vk = voltage(k); % "measure" voltage
  ik = current(k); % "measure" current
  Tk = T;          % "measure" temperature

  % Update SOC (and other model states)
  [sochat(k),socbound(k),spkfData] = iterSPKF(vk,ik,Tk,deltat,spkfData);
  % update waitbar periodically, but not too often (slow procedure)
  if mod(k,1000)==0, waitbar(k/length(current),hwait); end;
end
close(hwait);
```

Wrapper code (4)

- Wrapper code continues with plotting/analysis code

```
figure; plot(time/60,100*sochat,time/60,100*soc); hold on
plot([time/60; NaN; time/60],...
         [100*(sochat+socbound); NaN; 100*(sochat-socbound)]);
title('SOC estimation using SPKF'); xlabel('Time (min)'); ylabel('SOC (%)');
legend('Estimate','Truth','Bounds'); grid on

fprintf('RMS SOC estimation error = %g%%\n',sqrt(mean((100*(soc-sochat)).^2)));

figure; plot(time/60,100*(soc-sochat)); hold on
plot([time/60; NaN; time/60],[100*socbound; NaN; -100*socbound]);
title('SOC estimation errors using SPKF');
xlabel('Time (min)'); ylabel('SOC error (%)'); ylim([-4 4]);
legend('Estimation error','Bounds'); grid on

ind = find(abs(soc-sochat)>socbound);
fprintf('Percent of time error outside bounds = %g%%\n',...
        length(ind)/length(soc)*100);
```

SPKF initialization code (1)

- The SPKF initialization code begins with

```
function spkfData=initSPKF(v0,T0,SigmaX0,SigmaV,SigmaW,model)
  % Initial state description
  ir0   = 0;                          spkfData.irInd = 1;
  hk0   = 0;                          spkfData.hkInd = 2;
  SOC0  = SOCfromOCVtemp(v0,T0,model); spkfData.zkInd = 3;
  spkfData.xhat  = [ir0 hk0 SOC0]';    % initial state

  % Covariance values
  spkfData.SigmaX = SigmaX0;           spkfData.SigmaV = SigmaV;
  spkfData.SigmaW = SigmaW;            spkfData.Qbump = 5;
  spkfData.Snoise = real(chol(diag([SigmaW; SigmaV]),'lower'));

  % previous value of current (and its sign)
  spkfData.priorI = 0;                 spkfData.signIk = 0;
```

SPKF initialization code (2)

- The SPKF initialization code concludes with

```
  % SPKF specific parameters
  Nx = length(spkfData.xhat); spkfData.Nx = Nx; % state-vector length
  Ny = 1; spkfData.Ny = Ny; % measurement-vector length
  Nu = 1; spkfData.Nu = Nu; % input-vector length
  Nw = size(SigmaW,1); spkfData.Nw = Nw; % process-noise-vector length
  Nv = size(SigmaV,1); spkfData.Nv = Nv; % sensor-noise-vector length
  Na = Nx+Nw+Nv; spkfData.Na = Na;     % augmented-state-vector length

  h = sqrt(3); spkfData.h = h; % SPKF/CDKF tuning factor
  Weight1 = (h*h-Na)/(h*h); % weighting factors when computing mean
  Weight2 = 1/(2*h*h);      % and covariance
  spkfData.Wm = [Weight1; Weight2*ones(2*Na,1)]; % mean
  spkfData.Wc = spkfData.Wm;                     % covar

  % store model data structure too
  spkfData.model = model;
end
```

Summary

- Implementation of SPKF on ESC model refactors code
  - Initialization routine (`initSPKF.m`), called once at startup
  - Update routine (`iterSPKF.m`), called every sample interval
  - “Wrapper” code, coordinates the entire simulation process
- You have now seen the details of the initialization routine and the wrapper code
- Next lesson will present the update routine