SPKF iteration code, load model

- Implementation of SPKF on ESC model refactors code
  - "Wrapper" code, coordinates the entire simulation process
  - Initialization routine (`initSPKF.m`), called once at startup
  - Update routine (`iterSPKF.m`), called every sample interval, which starts with:

```
function [zk,zkbnd,spkfData] = iterSPKF(vk,ik,Tk,deltat,spkfData)
  model = spkfData.model;
  % Load the cell model parameters
  Q  = getParamESC('QParam',Tk,model);
  G  = getParamESC('GParam',Tk,model);
  M  = getParamESC('MParam',Tk,model);
  M0 = getParamESC('M0Param',Tk,model);
  RC = exp(-deltat./abs(getParamESC('RCParam',Tk,model)))';
  R  = getParamESC('RParam',Tk,model)';
  R0 = getParamESC('R0Param',Tk,model);
  eta = getParamESC('etaParam',Tk,model);
  if ik<0, ik=ik*eta; end;
```


SPKF iteration code, load covariances/states

- SPKF iteration code continues
  - Load constants, covariances, states from prior iteration

```
  % Get data stored in spkfData structure
  I = spkfData.priorI;
  SigmaX = spkfData.SigmaX;
  xhat = spkfData.xhat;
  Nx = spkfData.Nx;
  Nw = spkfData.Nw;
  Nv = spkfData.Nv;
  Na = spkfData.Na;
  Snoise = spkfData.Snoise;
  Wc = spkfData.Wc;
  irInd = spkfData.irInd;
  hkInd = spkfData.hkInd;
  zkInd = spkfData.zkInd;
  if abs(ik)>Q/100, spkfData.signIk = sign(ik); end;
  signIk = spkfData.signIk;
```


SPKF iteration code, step 1a (1)

- SPKF iteration code continues
  - Compute augmented $\hat{x}^{a,+}_{k-1}$, $\sqrt{\Sigma^{a,+}_{\tilde{x},k-1}}$

```
  % Step 1a: State estimate time update
  %          - Create xhatminus augmented SigmaX points
  %          - Extract xhatminus state SigmaX points
  %          - Compute weighted average xhatminus(k)

  % Step 1a-1: Create augmented SigmaX and xhat
  [sigmaXa,p] = chol(SigmaX,'lower');
  if p>0,
    fprintf('Cholesky error.  Recovering...\n');
    theAbsDiag = abs(diag(SigmaX));
    sigmaXa = diag(max(SQRT(theAbsDiag),SQRT(spkfData.SigmaW)));
  end
  sigmaXa=[real(sigmaXa) zeros([Nx Nw+Nv]); zeros([Nw+Nv Nx]) Snoise];
  xhata = [xhat; zeros([Nw+Nv 1])];
  % NOTE: sigmaXa is lower-triangular
```

SPKF iteration code, steps 1a–1b

- SPKF iteration code continues
  - Compute $\hat{x}_k^-$ and $\Sigma_{\tilde{x},k}^-$

```
% Step 1a-2: Calculate SigmaX points (strange indexing of xhata to
% avoid "repmat" call, which is very inefficient in MATLAB)
Xa = xhata(:,ones([1 2*Na+1])) + spkfData.h*[zeros([Na 1]), sigmaXa, -sigmaXa];

% Step 1a-3: Time update from last iteration until now
%     stateEqn(xold,current,xnoise)
Xx = stateEqn(Xa(1:Nx,:),I,Xa(Nx+1:Nx+Nw,:));
xhat = Xx*spkfData.Wm;

% Step 1b: Error covariance time update
%          - Compute weighted covariance sigmaminus(k)
%            (strange indexing of xhat to avoid "repmat" call)
Xs = Xx - xhat(:,ones([1 2*Na+1]));
SigmaX = Xs*diag(Wc)*Xs';
```


SPKF iteration code, steps 1c–2a

- SPKF iteration code continues
  - Compute $\hat{y}_k$ and L_k

```
% Step 1c: Output estimate
%          - Compute weighted output estimate yhat(k)
I = ik; yk = vk;
Y = outputEqn(Xx,I,Xa(Nx+Nw+1:end,:),Tk,model);
yhat = Y*spkfData.Wm;

% Step 2a: Estimator gain matrix
Ys = Y - yhat(:,ones([1 2*Na+1]));
SigmaXY = Xs*diag(Wc)*Ys';
SigmaY = Ys*diag(Wc)*Ys';
L = SigmaXY/SigmaY;
```


SPKF iteration code, steps 2b–2c

- SPKF iteration code continues
  - Compute $\hat{x}_k^+$ and $\Sigma_{\tilde{x},k}^+$

```
% Step 2b: State estimate measurement update
r = yk - yhat; % residual.  Use to check for sensor errors...
if r^2 > 100*SigmaY, L(:,1)=0.0; end
xhat = xhat + L*r;
xhat(zkInd)=min(1.05,max(-0.05,xhat(zkInd)));

% Step 2c: Error covariance measurement update
SigmaX = SigmaX - L*SigmaY*L';
[~,S,V] = svd(SigmaX);
HH = V*S*V';
SigmaX = (SigmaX + SigmaX' + HH + HH')/4; % Help maintain robustness
```

SPKF iteration code, step 2c (cont)

- SPKF iteration code continues
  - Adjust $\Sigma^{+}_{\tilde{x},k}$ if needed, store data for next iteration

```
% Q-bump code
if r^2>4*SigmaY, % bad voltage estimate by 2-SigmaX, bump Q
  fprintf('Bumping sigmax\n');
  SigmaX(zkInd,zkInd) = SigmaX(zkInd,zkInd)*spkfData.Qbump;
end

% Save data in spkfData structure for next time...
spkfData.priorI = ik;
spkfData.SigmaX = SigmaX;
spkfData.xhat = xhat;
zk = xhat(zkInd);
zkbnd = 3*sqrt(SigmaX(zkInd,zkInd));
```


Nested state-equation function

- SPKF "helper" function to implement state equation
  - Vector operations across all sigma points simultaneously

```
% Calculate new states for all of the old state vectors in xold.
function xnew = stateEqn(xold,current,xnoise)
  current = current + xnoise; % noise adds to current
  xnew = 0*xold;
  xnew(irInd,:) = RC*xold(irInd,:) + (1-RC)*current;
  Ah = exp(-abs(current*G*deltat/(3600*Q)));  % hysteresis factor
  xnew(hkInd,:) = Ah.*xold(hkInd,:) - (1-Ah).*sign(current);
  xnew(zkInd,:) = xold(zkInd,:) - current/3600/Q;
  xnew(hkInd,:) = min(1,max(-1,xnew(hkInd,:)));
  xnew(zkInd,:) = min(1.05,max(-0.05,xnew(zkInd,:)));
end
```


Nested output-equation function

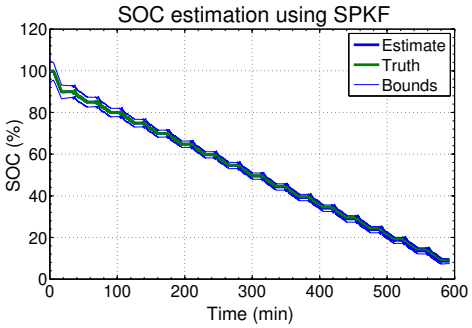
- SPKF "helper" function to implement output equation
  - Also, a "safe" square-root function

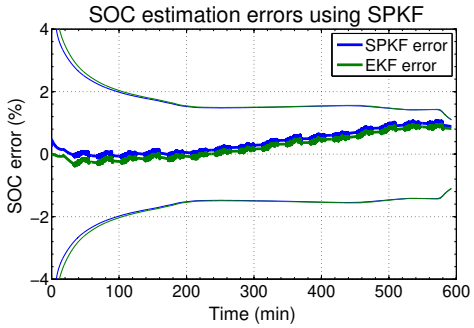
```
  % Calculate cell output voltage for all of state vectors in xhat
  function yhat = outputEqn(xhat,current,ynoise,T,model)
    yhat = OCVfromSOCtemp(xhat(zkInd,:),T,model);
    yhat = yhat + M*xhat(hkInd,:) + M0*signIk;
    yhat = yhat - R*xhat(irInd,:) - R0*current + ynoise(1,:);
  end

  % "Safe" square root
  function X = SQRT(x)
    X = sqrt(max(0,x));
  end
end
```

Example SPKF on ESC results

- For the following example, the SPKF was executed for the same test profiles as before
  - RMS SOC estimation error = 0.53 %
  - Percent of time error outside bounds = 0 %





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 entire codeset plus some example results
- SPKF works quite well as SOC estimator using ESC model