```
%
% January 2015
% Robert Richardson
% Energy & Power Group, University of Oxford

%-------------------------------------------------------------------------
%
% Battery Temperature Estimation using Impedance-Temperature Detection
 (ITD)
%-------------------------------------------------------------------------
%

%{
This code implements a Dual Extended Kalman Filter for estimation of
battery internal temperature distribution with unknown convection
coefficient using single frequency Electrochemical Impedance
Spectroscopy (EIS) measurements as measurement input. Execution of the
Mainscript.m file runs the simulation. The simulation results are
 compared
to experimental data.

I would ask that you cite this paper as Richardson, Robert R., and
David A. Howey. "Sensorless battery internal temperature estimation
 using a
kalman filter with impedance measurement." Sustainable Energy, IEEE
Transactions on 6.4 (2015): 1190-1199. if you want to use this code
 for
your own research. For further details on the work of the Energy Power
Group at Oxford, please see epg.eng.ox.ac.uk.

Copyright (c) 2016 by Robert Richardson, David Howey
and The Chancellor, Masters and Scholars of the University of Oxford.
See the licence file LICENCE.txt for more information.

%}

%
%{
% Overview of structs:
% P.         = parameters
% RAW.       = raw experimental data
% INTERP.    = interpolated experimental data
% RESULTS.   = results
%}

%-------------------------------------------------------------------------
%
% Main test file.  Initializes problem and calls DEKF.
%-------------------------------------------------------------------------
%

% Clear environment
```

```
clear; close all; clc;
addpath(genpath('./Functions'));
addpath(genpath('./Data'));

% Set plotting preferences
set(0,'defaultlinelinewidth',1.5)
set(0,'DefaultFigureColor','White');
set(0,'defaulttextfontsize', 11);
set(0,'DefaultAxesBox', 'on');

%-------------------------------------------------------------------------
%
% User parameters
%-------------------------------------------------------------------------
%

%-------------------------------------%
% select_dataset
%-------------------------------------%
% Choose which dataset to use
% HEV_drive_cycle_1 -> select_dataset = 1;
% HEV_drive_cycle_2 -> select_dataset = 2;
select_dataset = 1;

%-------------------------------------%
% select_EIS
%-------------------------------------%
% Choose whether to use Re(Z) or Im(Z) as measurement input
% real -> select_EIS = 1;
% imag -> select_EIS = 2;
select_EIS = 2;

%-------------------------------------------------------------------------
%
% Model Parameters
%-------------------------------------------------------------------------
%

% Deliberate errors
P.err_convection    = 39.3;             % h_init = h_true +
 err_convection
P.err_temperature   = 20;               % x_init = T_inf +
 err_temperature

% Measured parameters
P.r_o   = 0.0129;                       % Outer radius
P.Vb    = 3.4219e-5;                    % Cell volume
P.T_inf = 8.0;                          % Coolant fluid temperature
P.rho   = 2107;                         % Cell density

% Identified parameters
% (using HEV_drive_cycle_1)
P.cp        = 1171.6;                   % Specific heat capacity
P.kt        = 0.404;                    % Thermal conductivity
```

```
P.al        = P.kt/(P.rho*P.cp);        % Thermal diffusivity
P.h_true    = 39.3;                     % Convection coefficient


% Impedance-temperature coefficients
switch select_EIS;
    case 1;                             % Re(Z): Z' = a1*T^2 + a2*T +
 a3
    P.a1 = 0.4077969775136;
    P.a2 = -1.3492994134168;
    P.a3 = 195.3589523142926;
    case 2;                             % Im(Z): Z" = a1*T^2 + a2*T +
 a3
    P.a1 = 0.3156312310984;
    P.a2 = 4.6448029810131;
    P.a3 = 231.05989357985;
end

%-----------------------------------------------------------------------
%
% Experimental data
%-----------------------------------------------------------------------
%

% Load Temperature/Impedance/Voltage/Current data
load('Temperature_data')                % [t[s], T_surf, T_core,
 T_inf]
load('Impedance_data')                  % [t[s], phi, Re_Z, Im_Z]
load('Voltage_current_data')            % [t[s], I(mA), V(V)]

% Choose dataset
switch select_dataset
    case 1
        T_data = T_data_1;
        EIS_data = EIS_data_1;
        VC_data = VC_data_1;
    case 2
        T_data = T_data_2;
        EIS_data = EIS_data_2;
        VC_data = VC_data_2;
end

% Raw temperature data
RAW.TEMP.t      = T_data(:,1);
RAW.TEMP.T_surf = T_data(:,2);
RAW.TEMP.T_core = T_data(:,3);
RAW.TEMP.T_inf  = T_data(:,4);

% Raw impedance data
RAW.EIS.t       = EIS_data(:,1);
RAW.EIS.phi     = EIS_data(:,3);
RAW.EIS.Re_Z    = EIS_data(:,4);
RAW.EIS.Re_Z    = RAW.EIS.Re_Z - 0.008;    % subtract current
 collector resistance
```

```
RAW.EIS.Im_Z     = EIS_data(:,5);
RAW.EIS.Im_Z     = RAW.EIS.Im_Z + 0.001;    % ensure Im(Z)>0 for all Z
 for robustness

% Raw voltage & current data
RAW.VC.t         =  VC_data(:,1);
RAW.VC.I         =  VC_data(:,2);
RAW.VC.V         =  VC_data(:,3);

%-------------------------------------------------------------------------
%
% Interpolate experimental data
%-------------------------------------------------------------------------
%

% Adjust all to common time step (1 second)
P.delta_t    = 1;
INTERP.t     = [0:round(RAW.VC.t(end))]';

% Voltage/current data
INTERP.I     = zeros(length(INTERP.t),1);
INTERP.V     = zeros(length(INTERP.t),1);
for i = 1:P.delta_t:round(RAW.VC.t(end))
    INTERP.I(i) = interp1(RAW.VC.t,RAW.VC.I,INTERP.t(i));
    INTERP.V(i) = interp1(RAW.VC.t,RAW.VC.V,INTERP.t(i));
end

% Temperature data
INTERP.T_core = interp1(RAW.TEMP.t,RAW.TEMP.T_core,INTERP.t);
INTERP.T_surf = interp1(RAW.TEMP.t,RAW.TEMP.T_surf,INTERP.t);
INTERP.T_core(isnan(INTERP.T_core)) = P.T_inf;
INTERP.T_surf(isnan(INTERP.T_core)) = P.T_inf;

% Impedance data
% (note: for time steps with no measurements, set z = -1)
INTERP.Im_Z = -ones(1,length(INTERP.t));
INTERP.Re_Z = -ones(1,length(INTERP.t));
for i = 1:length(RAW.EIS.t)
    interpd_els = find(abs(RAW.EIS.t(i) - INTERP.t) < 0.5);
    INTERP.Im_Z(interpd_els) = RAW.EIS.Im_Z(i);
    INTERP.Re_Z(interpd_els) = RAW.EIS.Re_Z(i);
end

%-------------------------------------------------------------------------
%
% Heat generation
%-------------------------------------------------------------------------
%
P.dUdT           = -0.5e-3;                                 %
 d(U_OCV)/dT at 50% SOC (Forgez, 2010)
Q_ohm            = abs(INTERP.I.*(INTERP.V-3.3));         % ohmic
Q_rev            = INTERP.I.*INTERP.T_core*P.dUdT;        % reversible
 (small)
RESULTS.Q_tot    = Q_ohm + Q_rev;                           % total
```

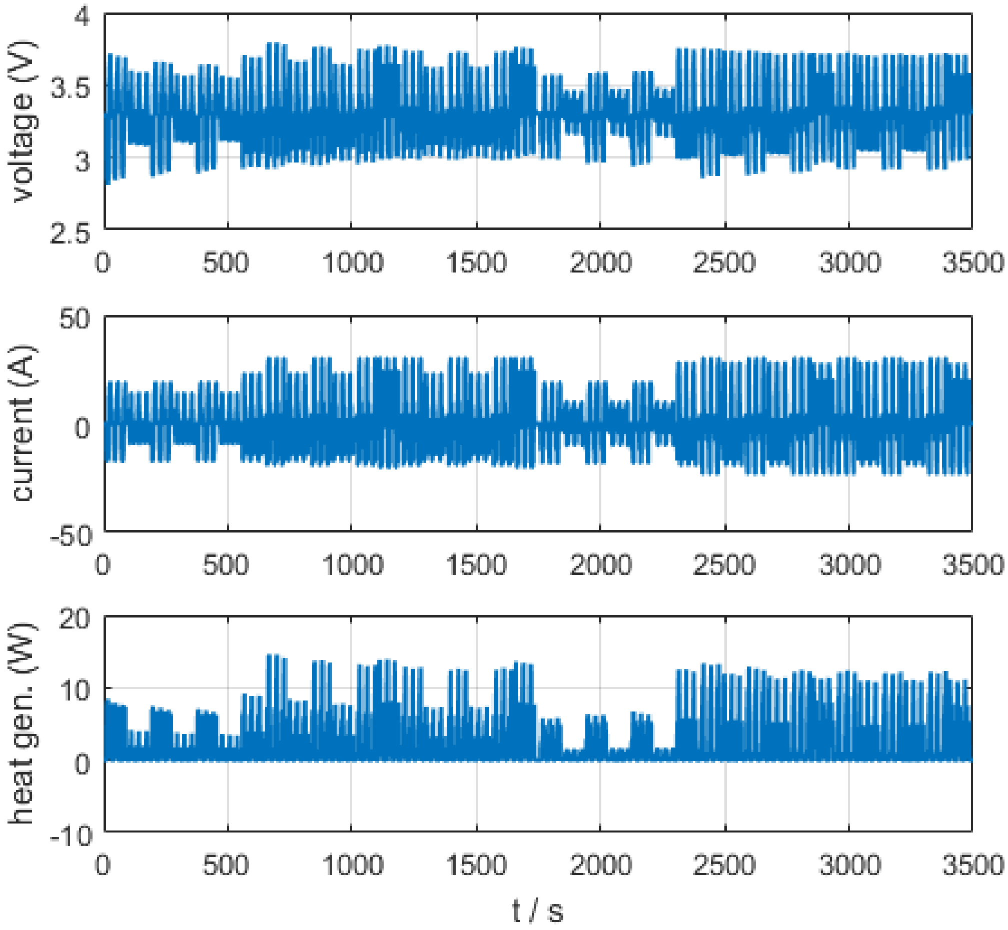
```
%-----------------------------------------------------------------------
%
% State space (ss) model of Polynomial Approximation (PA)
%-----------------------------------------------------------------------
%

% Initial values
P.h_init = P.h_true + P.err_convection;
P.x_init = [P.T_inf + P.err_temperature, 0];              % x =
 [T_avg, gamma_avg]


% Define state matrices
P.A = [-48*P.al*P.h_init / (P.r_o*(24*P.kt+P.r_o*P.h_init)),
 -15*P.al*P.h_init / (24*P.kt + P.r_o*P.h_init);
  ...
      -320*P.al*P.h_init / ((P.r_o^2)*(24*P.kt+P.r_o*P.h_init)),
    -120*P.al*(4*P.kt + P.r_o*P.h_init) / ((P.r_o^2)*(24*P.kt
+P.r_o*P.h_init))];

P.B = [P.al/(P.kt*P.Vb),
 48*P.al*P.h_init / (P.r_o*(24*P.kt + P.r_o*P.h_init));
      ...
      0,
 320*P.al*P.h_init / ((P.r_o^2)*(24*P.kt+P.r_o*P.h_init))];

P.C = [(24*P.kt - 3*P.r_o*P.h_init)/(24*P.kt + P.r_o*P.h_init),     -
(120*P.r_o*P.kt+15*(P.r_o^2)*P.h_init)/(8*(24*P.kt + P.r_o*P.h_init));
       ...
      24*P.kt/(24*P.kt + P.r_o*P.h_init),
 15*P.r_o*P.kt/(48*P.kt + 2*P.r_o*P.h_init)];

P.D = [0,
 4*P.r_o*P.h_init / (24*P.kt + P.r_o*P.h_init);
     ...
      0,
 P.r_o*P.h_init / (24*P.kt + P.r_o*P.h_init)];

%-----------------------------------------------------------------------
%
% Open loop (OL) model (no measurement feedback)
%-----------------------------------------------------------------------
%

% Discrete time state matrices
P.A_d = expm(P.A*P.delta_t);
P.B_d = inv(P.A)*(P.A_d-eye(2))*P.B;
%{
Alternatively use approx. values:
P.A_d = eye(2) + P.A*deltat;
P.B_d = P.B*deltat;
%}

% Simulate
```

```
x = zeros(2,length(INTERP.t));
u = zeros(2,length(INTERP.t));
y = zeros(2,length(INTERP.t));
x(:,1) = P.x_init';
for i = 1:length(INTERP.t)
    u(:,i) = [RESULTS.Q_tot(i); P.T_inf];
    x(:,i+1) = P.A_d*x(:,i) + P.B_d*u(:,i);
    y(:,i) = P.C*x(:,i) + P.D*u(:,i);
end

% Assign outputs to variables
RESULTS.OL.t = INTERP.t;
RESULTS.OL.T_core = y(1,:)';
RESULTS.OL.T_surf = y(2,:)';

%-----------------------------------------------------------------------
%
% Prepare Kalman Filters
%-----------------------------------------------------------------------
%

%-------------------------------------%
% Extended Kalman Filter (EKF)
%-------------------------------------%

% Linear terms
EKF.A   = P.A_d;
EKF.B   = P.B_d;
EKF.C   = P.C;
EKF.D   = P.D;
EKF.x   = P.x_init';
EKF.P_x = 1*eye(1);
EKF.u   = [0; P.T_inf];
EKF.H_x = [0.1; 0.1];
EKF.R_n = (0.0001^2)*eye(1);
EKF.R_v = (0.1^2)*eye(2);

% Measurements coefficients
EKF.a1 = P.a1;
EKF.a2 = P.a2;
EKF.a3 = P.a3;

% Fixed terms
EKF.r_o     = P.r_o;
EKF.T_inf   = P.T_inf;
EKF.Vb      = P.Vb;

%-------------------------------------%
% Dual Extended Kalman Filter (DEFK)
%-------------------------------------%

% Copy from EKF
DEKF = EKF;
```

```
% Additional variable terms
DEKF.h      = P.h_init;
DEKF.H_h    = 0.1;
DEKF.P_h    = 1*eye(1);
DEKF.R_e    = (2.5^2)*eye(1);
DEKF.R_r    = EKF.R_n;

% Additional fixed terms
DEKF.al     = P.al;
DEKF.kt     = P.kt;
DEKF.delta_t= P.delta_t;

%-----------------------------------------------------------------------------
%
% Run Kalman Filters
%-----------------------------------------------------------------------------
%
h_mat = zeros(size(INTERP.t));
for i = 1:length(INTERP.t)
    % EKF
    EKF(:,i).u = [RESULTS.Q_tot(i); P.T_inf];
    switch select_EIS;
        case 1;
        EKF(:,i).z = INTERP.Re_Z(i);
        case 2;
        EKF(:,i).z = INTERP.Im_Z(i);
    end
    EKF(:,i+1) = func_EKF(EKF(:,i));

    % DEKF
    RESULTS.DEKF.h_mat(i) = DEKF(i).h;
    DEKF(:,i).u = [RESULTS.Q_tot(i); P.T_inf];
    switch select_EIS;
        case 1;
        DEKF(:,i).z = INTERP.Re_Z(i);
        case 2;
        DEKF(:,i).z = INTERP.Im_Z(i);
    end
    DEKF(:,i+1) = func_DEKF(DEKF(:,i));
end

%-----------------------------------------------------------------------------
%
% Calculate T_core and T_surf from states
%-----------------------------------------------------------------------------
%

% EKF
EKF_x = [EKF(1:end-1).x];
EKF_z = [EKF(1:end-1).z];
EKF_u = [EKF(1:end-1).u];

RESULTS.EKF.T = zeros(size(EKF_x));
for i = 1:length(EKF_x(1,:))
```

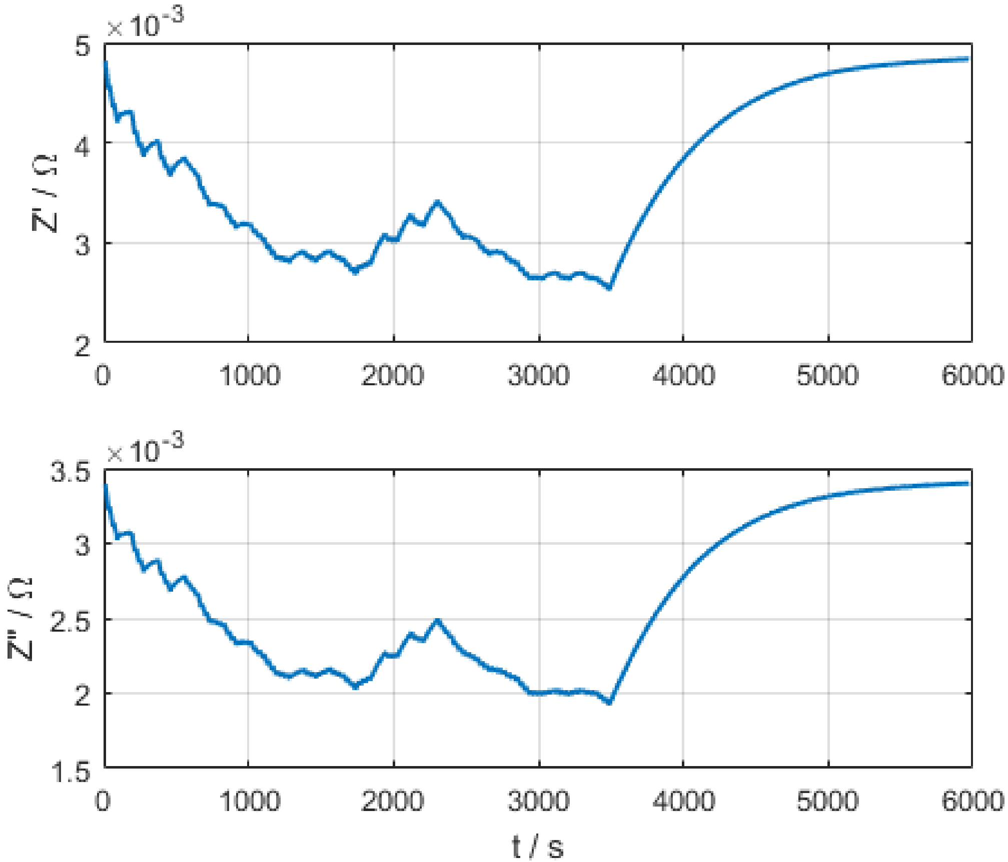
```
    RESULTS.EKF.T(:,i) = P.C*EKF_x(:,i) + P.D*EKF_u(:,i);
end
RESULTS.EKF.T_core = RESULTS.EKF.T(1,:);
RESULTS.EKF.T_surf = RESULTS.EKF.T(2,:);

% DEKF
DEKF_x = [DEKF(1:end-1).x];
DEKF_z = [DEKF(1:end-1).z];
DEKF_u = [DEKF(1:end-1).u];

RESULTS.DEKF.T = zeros(size(DEKF_x));
for i = 1:length(DEKF_x(1,:))
    RESULTS.DEKF.T(:,i) = DEKF(i).C*DEKF_x(:,i) +
 DEKF(i).D*DEKF_u(:,i);
end
RESULTS.DEKF.T_core = RESULTS.DEKF.T(1,:);
RESULTS.DEKF.T_surf = RESULTS.DEKF.T(2,:);

%-------------------------------------------------------------------------
%
% Calculate errors
%-------------------------------------------------------------------------
%

% Open Loop
RESULTS.OL.err_T_core = RESULTS.OL.T_core - INTERP.T_core;
RESULTS.OL.err_T_surf = RESULTS.OL.T_surf - INTERP.T_surf;

% EKF
RESULTS.EKF.err_T_core = RESULTS.EKF.T_core - INTERP.T_core';
RESULTS.EKF.err_T_surf = RESULTS.EKF.T_surf - INTERP.T_surf';

% DEKF
RESULTS.DEKF.err_T_core = RESULTS.DEKF.T_core - INTERP.T_core';
RESULTS.DEKF.err_T_surf = RESULTS.DEKF.T_surf - INTERP.T_surf';

% RMS errors (DEKF)
RESULTS.DEKF.RMS_T_core =
 sqrt(mean(RESULTS.DEKF.err_T_core(2:3500).^2));
RESULTS.DEKF.RMS_T_surf =
 sqrt(mean(RESULTS.DEKF.err_T_surf(2:3500).^2));
RESULTS.DEKF.RMS_T_core_1200_3500s =
 sqrt(mean(RESULTS.DEKF.err_T_core(1200:3500).^2));
RESULTS.DEKF.RMS_T_surf_1200_3500s =
 sqrt(mean(RESULTS.DEKF.err_T_surf(1200:3500).^2));

%-------------------------------------------------------------------------
%
% Plot voltage/current/heat generation
%-------------------------------------------------------------------------
%
hFig = figure;
set(hFig, 'Position', [100 100 500 450])
```

```
% Voltage
subplot(3,1,1)
plot(INTERP.t,INTERP.V)
xlim([0 3500])
ylabel('voltage (V)')

% Current
subplot(3,1,2)
plot(INTERP.t,INTERP.I)
xlim([0 3500])
ylabel('current (A)')

% Heat generation
subplot(3,1,3)
plot(INTERP.t,RESULTS.Q_tot)
xlim([0 3500])
xlabel 't / s';
ylabel('heat gen. (W)')
```




```
%------------------------------------------------------------------------
%
```

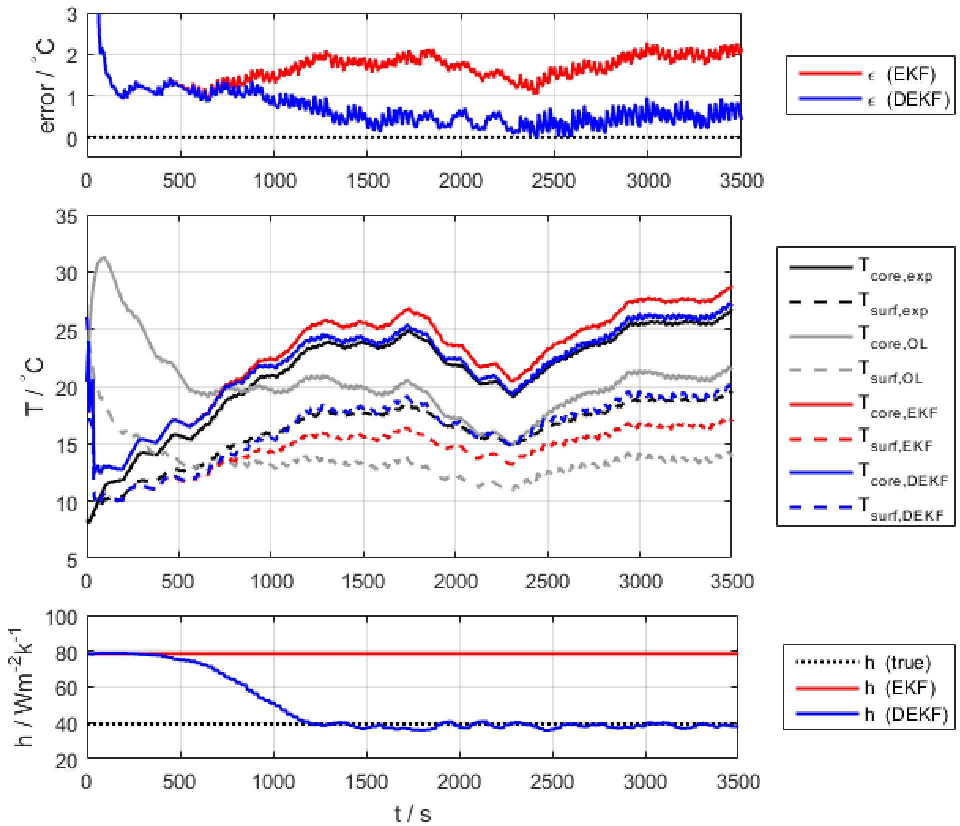
```
% Plot real/imaginary impedance
%-----------------------------------------------------------------------
%
hFig = figure;
set(hFig, 'Position', [100 100 500 400])

% Real Impedance
subplot(2,1,1)
plot(RAW.EIS.t, RAW.EIS.Re_Z);
ylabel 'Z'' / \Omega'

% Imaginary Impedance
subplot(2,1,2)
plot(RAW.EIS.t, RAW.EIS.Im_Z);
xlabel 't / s';
ylabel 'Z" / \Omega'
```




```
%-----------------------------------------------------------------------
%
% Plot results
%-----------------------------------------------------------------------
%
hFig = figure;
set(hFig, 'Position', [100 100 (560+150) 580])
```

```
%-------------------------------------%
% Subplot 1: Core temperature error
%-------------------------------------%
subplot(4,1,1);
hold on;

hplot_err_zero = plot(RESULTS.OL.t,
 zeros(size(RESULTS.DEKF.err_T_core)), 'k:');
hplot_EKF_err_T_core = plot(RESULTS.OL.t,
 RESULTS.EKF.err_T_core, 'r-');
hplot_DEKF_err_T_core = plot(RESULTS.OL.t,
 RESULTS.DEKF.err_T_core, 'b-');

xlim([0 3500])
ylim([-0.5 3])
ylabel('error / ^\circC')

legend([hplot_EKF_err_T_core,  hplot_DEKF_err_T_core],...
    '\epsilon  (EKF)',...
    '\epsilon  (DEKF)',...
    'location','eastoutside'...
    )


%-------------------------------------%
% Subplot 2: Core/surface temperature
%-------------------------------------%
subplot(4,1,[2 3])
hold on;

% Exp. measurements
hplot_Tc_exp        = plot(RAW.TEMP.t, RAW.TEMP.T_core ,'k-');
hplot_Ts_exp        = plot(RAW.TEMP.t, RAW.TEMP.T_surf ,'k--');

% Open Loop (OL)
h_plot_Tc_open_loop = plot(RESULTS.OL.t,
 RESULTS.OL.T_core ,'-','color',[0.6 0.6 0.6]);
h_plot_Ts_open_loop = plot(RESULTS.OL.t,
 RESULTS.OL.T_surf ,'--','color',[0.6 0.6 0.6]);

% EKF
hplot_Tc_EKF        = plot(INTERP.t,RESULTS.EKF.T_core,'r-');
hplot_Ts_EKF        = plot(INTERP.t,RESULTS.EKF.T_surf,'r--');

% DEKF
hplot_Tc_DEKF       = plot(INTERP.t,RESULTS.DEKF.T_core,'b-');
hplot_Ts_DEKF       = plot(INTERP.t,RESULTS.DEKF.T_surf,'b--');

% Labels
xlim([0 3500])
ylim([5 35])
ylabel('T / ^\circC')

% Legend
```

```
legend([hplot_Tc_exp,  hplot_Ts_exp,...
    h_plot_Tc_open_loop, h_plot_Ts_open_loop,...
    hplot_Tc_EKF, hplot_Ts_EKF,...
    hplot_Tc_DEKF, hplot_Ts_DEKF,...
    ],...
    'T_c_o_r_e_,_e_x_p',...
    'T_s_u_r_f_,_e_x_p',...
    'T_c_o_r_e_,_O_L',...
    'T_s_u_r_f_,_O_L',...
    'T_c_o_r_e_,_E_K_F',...
    'T_s_u_r_f_,_E_K_F',...
    'T_c_o_r_e_,_D_E_K_F',...
    'T_s_u_r_f_,_D_E_K_F',...
    'location','eastoutside'...
    )

%-------------------------------------%
% Subplot 3: Convection coefficient
%-------------------------------------%
subplot(4,1,4);
hold on;

hplot_h_exp     = plot(INTERP.t, P.h_true*ones(size(INTERP.t)), 'k:');
hplot_h_EKF     = plot(INTERP.t, P.h_init*ones(size(INTERP.t)),'r-');
hplot_h_DEKF    = plot(INTERP.t, [RESULTS.DEKF.h_mat],'b-');

xlim([0 3500])
ylim([20 100])
xlabel('t / s');
ylabel('h / Wm^-^2k^-^1')

legend([hplot_h_exp hplot_h_EKF hplot_h_DEKF],...
    'h  (true)','h  (EKF)','h  (DEKF)',...
    'location','eastoutside')
```




```
%-----------------------------------------------------------------------
%
% Save data for other uses
%-----------------------------------------------------------------------
%
save('./Data/MainScriptResults/
MainScriptResults','P','RAW','INTERP','RESULTS','EKF','DEKF')
```

Published with MATLAB® R2015b