Thermal Modelling of a Cylindrical 2.3Ah LiFePO4 Battery

We would like to calculate the temperature in the Li-ion battery as a function of time. The parabolic equation describing heat transfer is

$$\rho C\frac{\partial u}{\partial t} - \nabla\cdot(k\nabla u) = q,$$

where $\rho, C,$ and k are the density, specific heat, and thermal conductivity of the material, respectively, u is the temperature, and q is the heat generated in the battery.

Since the problem is axisymmetric, it is convenient to write this equation in a cylindrical coordinate system.

$$\rho C\frac{\partial u}{\partial t} - \frac{1}{r}\frac{\partial}{\partial r}\left(kr\frac{\partial u}{\partial r}\right) - \frac{1}{r^2}\frac{\partial}{\partial \theta}\left(k\frac{\partial u}{\partial \theta}\right) - \frac{\partial}{\partial z}\left(k\frac{\partial u}{\partial z}\right) = q,$$

where r, θ, and z are the three coordinate variables of the cylindrical system. Because the problem is axisymmetric, $\partial u/\partial\theta = 0$ and after multiplying by r the equation can be rewritten as

$$r\rho C\frac{\partial u}{\partial t} - \frac{\partial}{\partial r}\left(kr\frac{\partial u}{\partial r}\right) - \frac{\partial}{\partial z}\left(kr\frac{\partial u}{\partial z}\right) = rq.$$

The equation can be converted to the form supported by PDE Toolbox if r is defined as y and z is defined as x. Rewriting the above equation gives

$$\rho Cy\frac{\partial u}{\partial t} - \nabla\cdot(ky\nabla u) = qy.$$

The boundary conditions contain Dirichlet type, where the temperature on the boundary is specified, or Neumann type where the heat flux is specified. The generalized Neumann boundary condition equation is

$$\vec{n}\cdot(k\nabla u) + qu = g$$

The equation can be rewritten as temperature expression

$$\vec{n}\cdot(k\nabla T) + qT = g$$

Convection and radiation heat transfer have been applied on surface of battery and zero heat flux(zero neumann boundary) has been applied on axial-symmetry in this study using generalized neumann bounday condition equation.

Optimized parameters has been used to obtain results.

Simulation results have been compared with experimental results in this study.

Contents

experimental power data (W) is used in this study calls @expfcoef function mfile

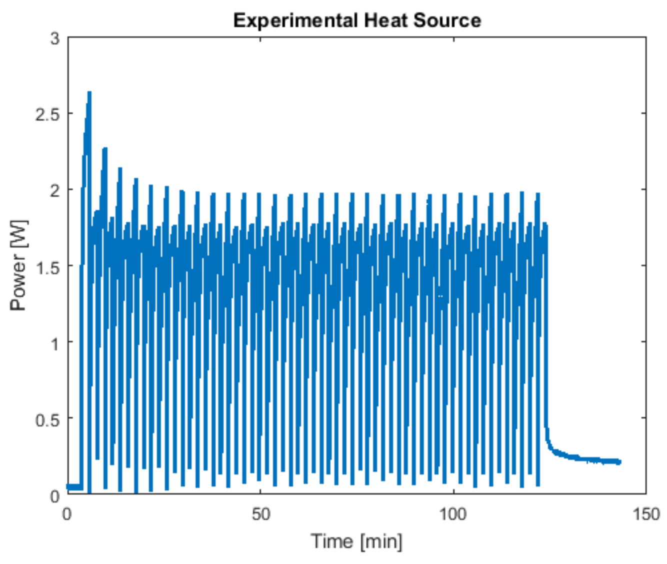
```
addpath( 'functions' );

clearvars -global -except rad* leng* irad* lengActive* thickAlu*  thickGas*  vol*
clear;
clc;
close all;
load('HeatPower.mat')
load TempDataMaccor
load TempDataPCB
```

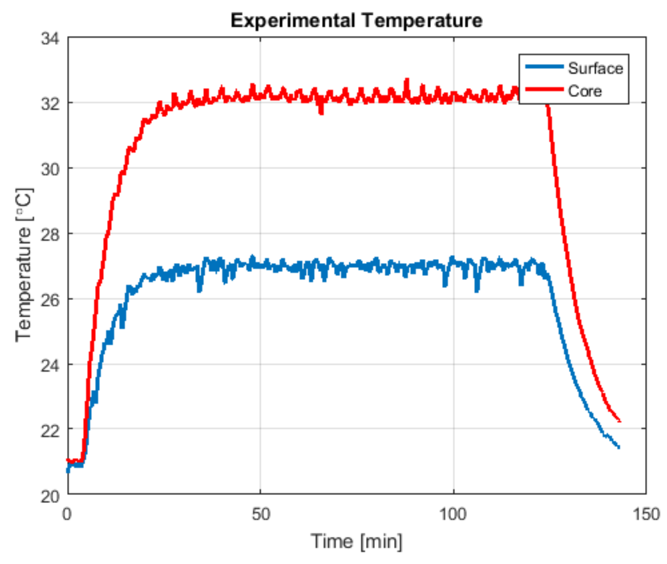
Geometry Properties

```
global rad leng irad radActive lengActive thickAlu thickGas vol HeatPowerExp mymodel Ta mesh

rad = 1.3e-2;                               % battery radius meter(m)
thickAlu = 0.4e-3;                          % thickness of aluminum case (m)
irad = 1.65e-3;                             % inner tube's radius (m)
radActive = rad - 1*thickAlu;               % active material's radius (m)
leng = 6.5e-2;                              % length of battery (m)
thickGas = 0.005;                           % thickness of gas insulator (m)
lengActive = leng - 2*(thickAlu+thickGas);  % active material's length (m)
vol = pi*lengActive*(radActive^2 - irad^2); % active volume (m^3)
```

Experimental Plots

```
HeatPower1(:,1) = HeatPower(43000:51000,1) - HeatPower(43000,1); % Measurment time
HeatPower1(:,2) = HeatPower(43000:51000,2); % Power = I * eta

HeatPowerExp = HeatPower1;

figure;
plot( HeatPowerExp(:,1)./60 , HeatPowerExp(:,2) , 'LineWidth' , 2 );
xlabel('Time [min]');
ylabel('Power [W]');
title('Experimental Heat Source')

TempSurf = TempDataMaccor(43000:51000,3);
TempCore = TempDataMaccor(43000:51000,2);

figure;
plot( HeatPowerExp(:,1)./60 , TempSurf , ...
      HeatPowerExp(:,1)./60 , TempCore , 'r' , 'LineWidth' , 2 );
xlabel('Time [min]');
ylabel('Temperature [\circC]');
title('Experimental Temperature');
legend( 'Surface' , 'Core' );
grid on
```





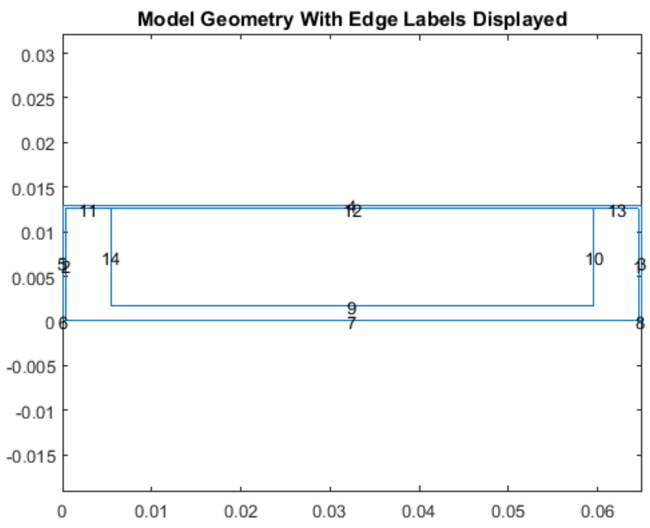
Create The PDE Model

```
mymodel = createpde; % for 1 equation

P1 = [2,8,...
    thickAlu, (leng-thickAlu), (leng-thickAlu),  leng-(thickAlu+thickGas), leng-(thickAlu+thickGas), (thickAlu+thickGas), (thickAlu+thickGas), thickAlu,...
    0,           0,        (rad-thickAlu),       (rad-thickAlu),                 irad,            irad,                (rad-thickAlu),     (rad-thickAlu)];

R2 = [3, 4, 0, leng, leng, 0, 0, 0, rad, rad];  % aluminum case
R2 = [R2,zeros(1,length(P1) - length(R2))];

R3 = [3, 4, (thickAlu+thickGas), leng-(thickAlu+thickGas), leng-(thickAlu+thickGas), (thickAlu+thickGas), irad, irad, (rad-thickAlu), (rad-thickAlu)];
R3 = [R3,zeros(1,length(P1) - length(R3))];     % active domain

gdm = [P1;R2;R3]';
```

```
sf = '(P1+R2+R3)';
ns = char('P1','R2','R3');
ns = ns';
[dl,bt] = decsg(gdm,sf,ns);
geometryFromEdges(mymodel,dl);

% [dl2,bt2] = csgdel(dl,bt);   % remove subdomain
```

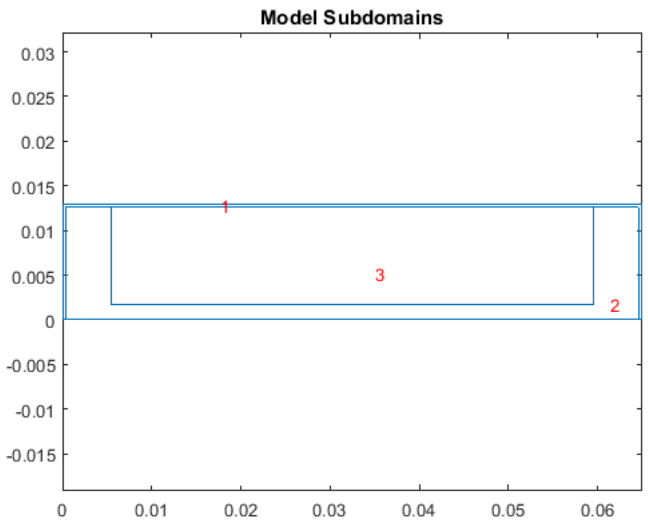
Geometry of Battery

```
figure;
pdegplot(mymodel,'edgeLabels','on')
title('Model Geometry With Edge Labels Displayed');
axis equal;
```



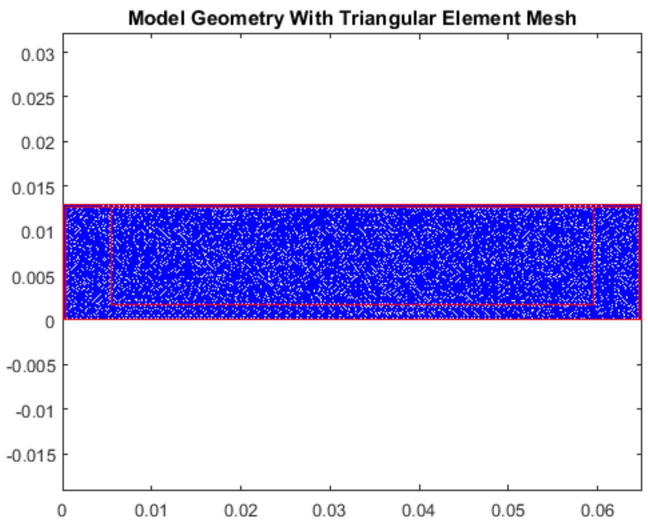
Subdomains Plot

```
figure;
pdegplot(mymodel,'SubdomainLabels','on');
title('Model Subdomains');
axis equal;
```



Generate Mesh

```
hmax = 0.0004;  % control mesh density
mesh = generateMesh(mymodel,'Hmax',hmax,'GeometricOrder','linear','Jiggle','on');
figure;
pdeplot(mymodel);
title('Model Geometry With Triangular Element Mesh');
axis equal;
```



Material Properties

```
% Active(positive electrode + separator + negative electrode) material
kActive = 0.5548;  % W/m*K       % thermal conductivity of active metal
CpActive = 958;    % J/kg*K      % heat capacity of active material
rhoActive = 2000;  % kg/m^3      % density of active material

% Aluminum
kAlu = 238;        % W/m*K       % thermal conductivity of aluminum  %1.7602;
CpAlu = 903;       % J/kg*K      % heat capacity of aluminum
rhoAlu = 1500;     % kg/m^3      % density of aluminum

% Assumed gas(air) in tubes
kGas = 0.025;     % W/m*K        % thermal conductivity of gas
CpGas = 1.005;    % J/kg*K       % heat capacity of gas
rhoGas = 1.205;   % kg/m^3       % density of gas
```

System Constants

```
kelvin = 273.15;
Ta = 20 + kelvin;           % ambient temperature (K)
hCoef = 20;                 % heat transfer coefficient (W/m^2/K)
emissivity = 0.65;          % emissivity coeffient for radiation heat transfer
stefanBoltzmann = 5.67e-8;  % stefan-boltzmann constant
```

Insert Optimized Values

```
load x5.mat;

p.ini = x5;
```

Specify Coefficients

```
m = 0; % common coefficient for all domains
a = 0; % common coefficient for all domains

cActive = @(region,~) p.ini(1) * region.y;                % 'c' coeffiecient for active domain
dActive = @(region,~) p.ini(3) * p.ini(2) * region.y;     % 'd' coeffiecient for active domain

cAlu = @(region,~) p.ini(4) * region.y;                   % 'c' coeffiecient for aluminum case
dAlu = @(region,~) p.ini(6) * p.ini(5) * region.y;        % 'd' coeffiecient for aluminum case

cGas = @(region,~) p.ini(7) * region.y;                   % 'c' coeffiecient for air
dGas = @(region,~) p.ini(9) * p.ini(8) * region.y;        % 'd' coeffiecient for air

% Call heat source function
f = @expfcoef;

% Specify Coefficient for each subdomain
specifyCoefficients(mymodel,'m',m,'d',dAlu,'c',cAlu,'a',a,'f',0,'face',1);
specifyCoefficients(mymodel,'m',m,'d',dGas,'c',cGas,'a',a,'f',0,'face',2);
specifyCoefficients(mymodel,'m',m,'d',dActive,'c',cActive,'a',a,'f',f,'face',3);
```

Boundary Conditions

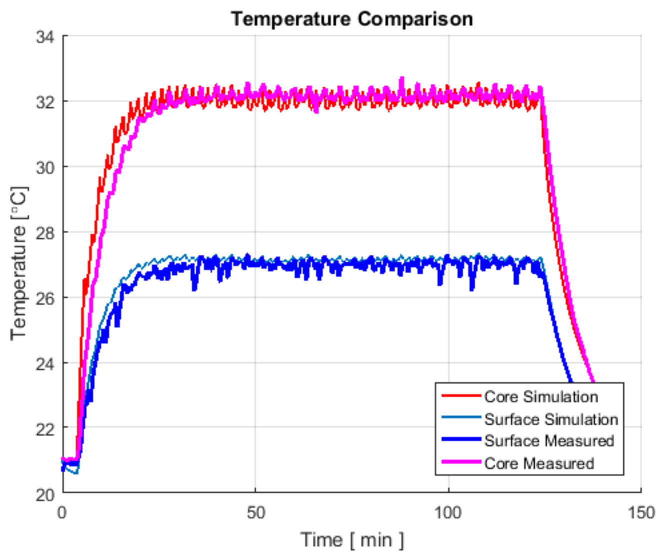
```
gb = @(region,state) region.y .* ( p.ini(10) * Ta + stefanBoltzmann * p.ini(11) * Ta^4 );
qb = @(region,state) region.y .* ( p.ini(10) + stefanBoltzmann * p.ini(11) * state.u.^3 );

% Apply boundary condition on edges for cooling
applyBoundaryCondition(mymodel,'Edge',6,'g',0,'q',0);
applyBoundaryCondition(mymodel,'Edge',7,'g',0,'q',0);
applyBoundaryCondition(mymodel,'Edge',8,'g',0,'q',0);
applyBoundaryCondition(mymodel,'Edge',3,'g',gb,'q',qb);
applyBoundaryCondition(mymodel,'Edge',4,'g',gb,'q',qb);
applyBoundaryCondition(mymodel,'Edge',5,'g',gb,'q',qb);
```

Unsteady State Condition

```
tlist = HeatPowerExp(:,1);
T0 = 21 + kelvin;       % initial temperature

% Set initital conditions
setInitialConditions(mymodel, T0);

% Set solver options
mymodel.SolverOptions.AbsoluteTolerance = 1e-8;
mymodel.SolverOptions.RelativeTolerance = 1e-4;
mymodel.SolverOptions.ResidualTolerance = 1e-5;
mymodel.SolverOptions.MaxIterations = 100;
mymodel.SolverOptions.MinStep = 1e-6;

% Simulate model
result = solvepde(mymodel, tlist);

% Get dependent variable result (temperature)
u = result.NodalSolution;
```

Result and Plot

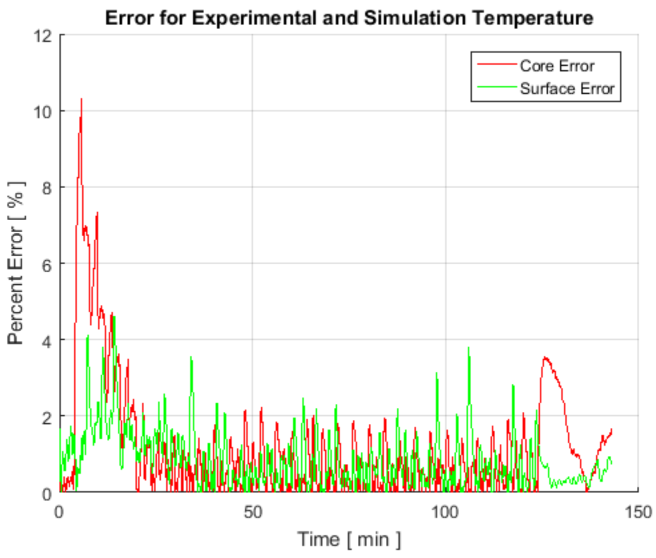
```
% Get a node near the spesific point using helper function
getClosestNode = @(p,x,y) min((p(1,:) - x).^2 + (p(2,:) - y).^2);

% Call this function to get a node
[~,nid] = getClosestNode( mesh.Nodes, leng/2, 0 );

% Core Temperature Node
[~,nid1] = getClosestNode( mesh.Nodes, leng/2, 0 );

% Surface Temperature Node
[~,nid4] = getClosestNode( mesh.Nodes, leng/2, rad );


% Plot of temperature comparisons at different locations

% Core & Surface Temperature of Simulation
TempCoreSim = u(nid1,:) - kelvin;
TempSurfSim = u(nid4,:) - kelvin;

figure;
hold on;

plot(tlist./60, TempCoreSim, 'r', 'LineWidth',1.5);
plot(tlist./60, TempSurfSim, 'LineWidth', 1.5);
plot( HeatPowerExp(:,1)./60 , TempSurf , 'b', ...
      HeatPowerExp(:,1)./60 , TempCore , 'm' , 'LineWidth' , 2 );

xlabel('Time [ min ]');
ylabel('Temperature [\circC]');
title('Temperature Comparison');
legend( 'Core Simulation' , 'Surface Simulation' , 'Surface Measured' , 'Core Measured' , 'Location' , 'SouthEast' );
grid on;
```



Error Analysis

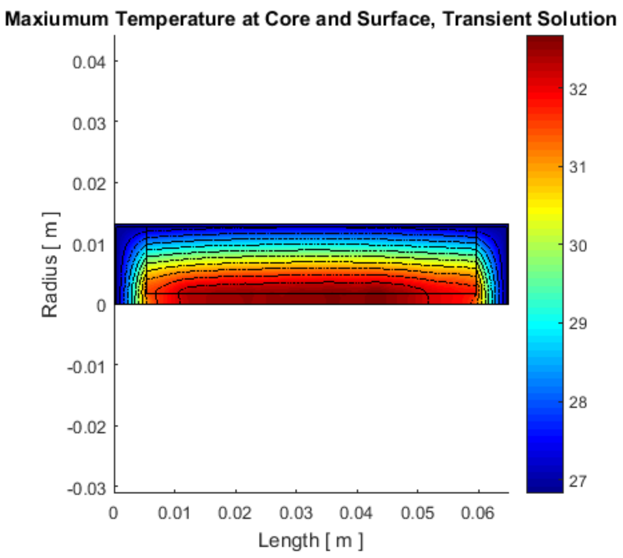
```
% Absolute Difference of Measured and Simulation Temperature
diffCore = abs(TempCore' - TempCoreSim);
diffSurf = abs(TempSurf' - TempSurfSim);

% Percent Error
```

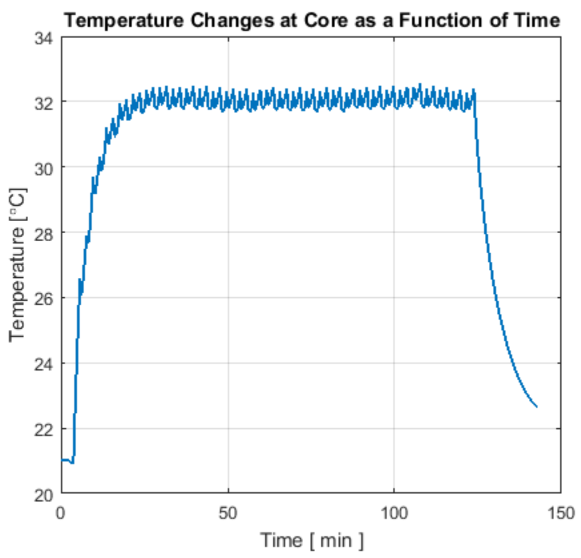
```
errorCore = (diffCore./TempCoreSim)*100;
errorSurf = (diffSurf./TempSurfSim)*100;

figure;
hold on;
plot(tlist./60, errorCore,'r', 'LineWidth', 1);
plot(tlist./60, errorSurf,'g', 'LineWidth', 1);
xlabel('Time [ min ]');
ylabel('Percent Error [ % ]');
legend('Core Error', 'Surface Error');
title('Error for Experimental and Simulation Temperature');
grid on;
```



Two plots are useful in understanding the results from this transient analysis. The first is a plot of the maximum heating and temperature during the simulation. The second is a plot of the temperature at core in the battery

```
% Get Maximum Temperature
maxT = max(u') - kelvin;

s = figure;
s.Position = [1 1 2 1].*s.Position;

subplot(1,2,1);
pdeplot(mymodel,'xydata',maxT,'contour','on','colormap','jet');
title('Maximum Temperature at Core and Surface, Transient Solution');
xlabel('Length [ m ]');
ylabel('Radius [ m ]');
axis equal;

subplot(1,2,2);
plot(tlist./60, (u(nid,:)-kelvin),'LineWidth', 1.5)
title('Temperature Changes at Core as a Function of Time');
xlabel('Time [ min ]');
ylabel('Temperature [\circC]');
grid on;

% The two plots are shown side-by-side in the figure below
```





```
% model report on command window
mymodel.SolverOptions.ReportStatistics = 'on'
```

```
mymodel =

  PDEModel with properties:

           PDESystemSize: 1
         IsTimeDependent: 1
                Geometry: [1x1 AnalyticGeometry]
    EquationCoefficients: [1x1 CoefficientAssignmentRecords]
      BoundaryConditions: [1x6 BoundaryCondition]
       InitialConditions: [1x1 InitialConditionsRecords]
                    Mesh: [1x1 FEMesh]
           SolverOptions: [1x1 PDESolverOptions]
```

Published with MATLAB® R2016a