Doyle-Fuller-Newman (DFN) Full Order Model (FOM) Tutorial

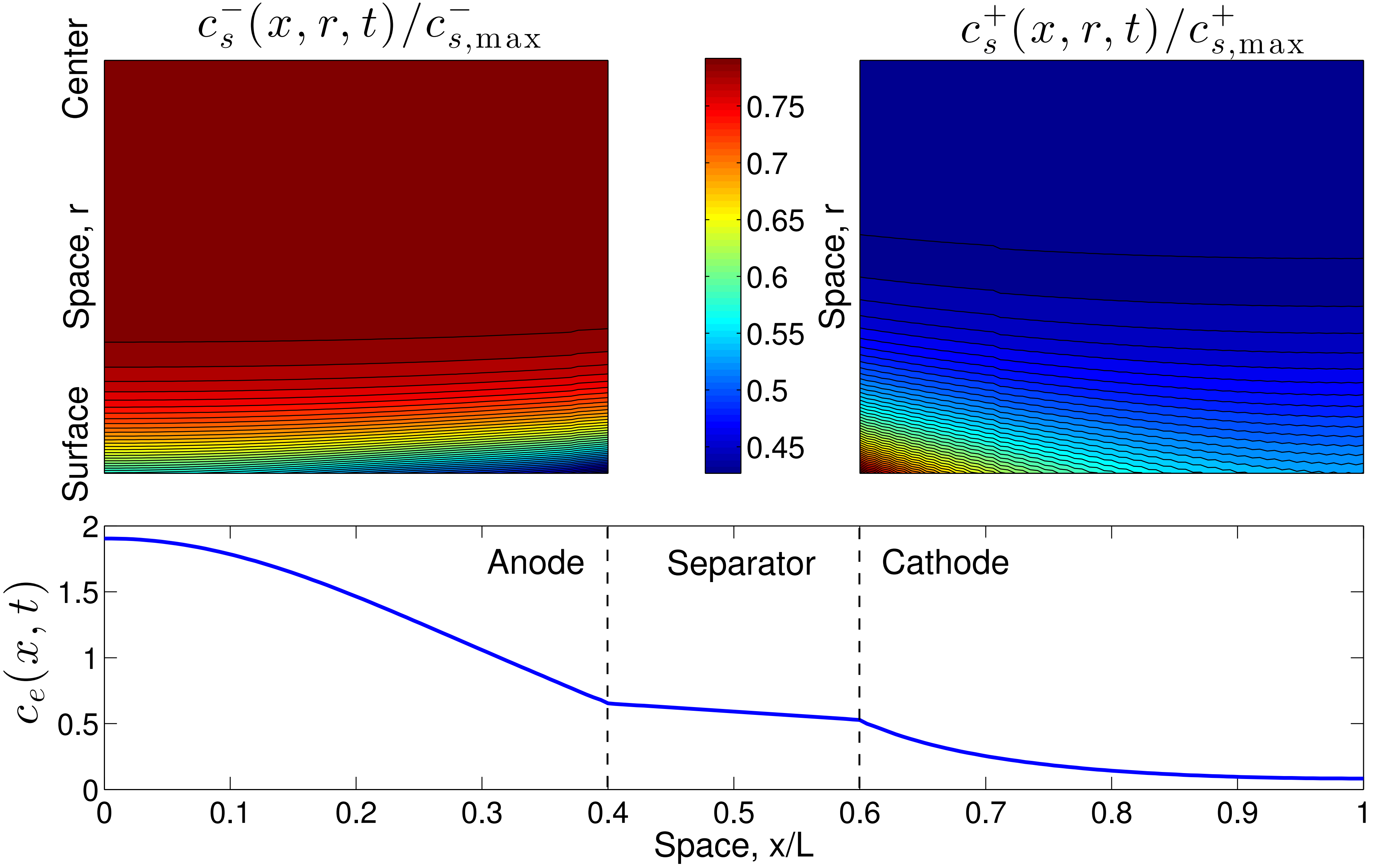
UC Berkeley

eCAL

Professor Scott Moura

May 12, 2016

Sample Output



Background

This code is Matlab implementation of electrochemical model equations found in

- Doyle, Marc, Thomas F. Fuller, and John Newman. "Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell." Journal of the Electrochemical Society 140.6 (1993): 1526-1533.
- Fuller, Thomas F., Marc Doyle, and John Newman. "Simulation and optimization of the dual lithium ion insertion cell." Journal of the Electrochemical Society 141.1 (1994): 1-10.
- Thomas, Karen E., John Newman, and Robert M. Darling. "Mathematical modeling of lithium batteries." Advances in lithium-ion batteries. Springer US, 2002. 345-392.
- and more…

What it does

- Simulates
  - Terminal voltage [V]
  - Solid & electrolyte concentrations [mol/m^3]
  - Solid & electrolyte potentials [V]
  - Internal temperature [K]
- Generates
  - Various static plots and animations
- Takes as input
  - An applied electric current time-series (time, current)
- Requires
  - A parameter file, which defines electrochemical model parameters
  - OCP functions, electrolyte conductivity & diffusivity fcns
- *Assumes user is proficient with Matlab*

What it does ***NOT*** do

- Provide polished user-interface. This code is for developers, and is under development.
- Simulate
  - Blended cathode materials
  - Different particle sizes
  - non Li-ion chemistries (e.g. NiMH)
  - Battery pack geometries
  - Mechanical stress
  - Aging
- Include the model identification code (PSO)
- Some of these features are under-development
- Keep your expectations reasonable please ☺

Tutorial Outline

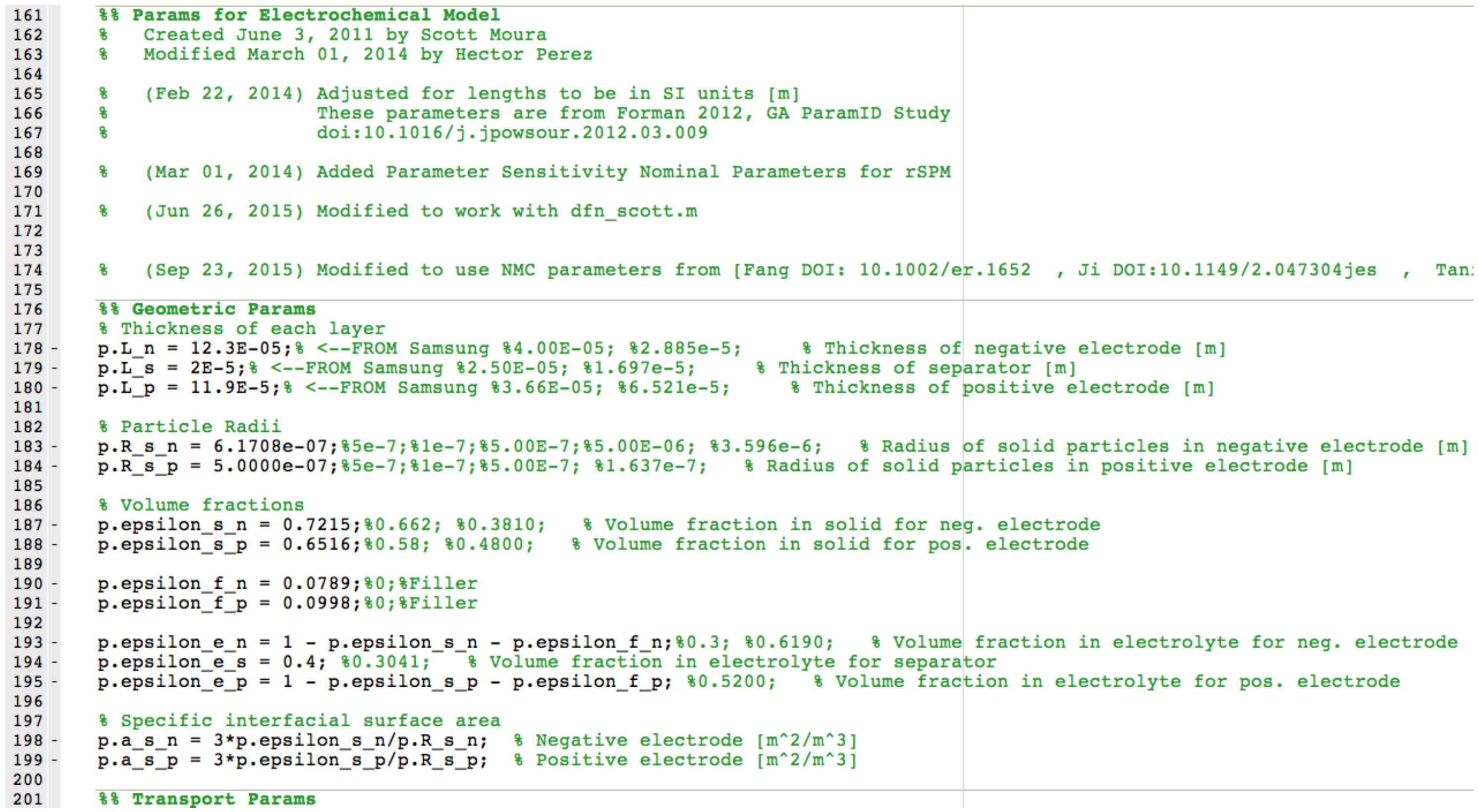
1. Parameter file
2. Material function files
3. Input current data
4. Main script to run DFN model
5. Generate plots and animations

Tutorial Outline

1. Parameter file
2. Material function files
3. Input current data
4. Main script to run DFN model
5. Generate plots and animations

Parameter File

Open `params_NMC_Samsung_new_iteration.m`



```
%% Params for Electrochemical Model
%   Created June 3, 2011 by Scott Moura
%   Modified March 01, 2014 by Hector Perez

%   (Feb 22, 2014) Adjusted for lengths to be in SI units [m]
%                  These parameters are from Forman 2012, GA ParamID Study
%                  doi:10.1016/j.jpowsour.2012.03.009

%   (Mar 01, 2014) Added Parameter Sensitivity Nominal Parameters for rSPM

%   (Jun 26, 2015) Modified to work with dfn_scott.m


%   (Sep 23, 2015) Modified to use NMC parameters from [Fang DOI: 10.1002/er.1652  , Ji DOI:10.1149/2.047304jes   ,  Tan

%% Geometric Params
% Thickness of each layer
p.L_n = 12.3E-05;% <--FROM Samsung %4.00E-05; %2.885e-5;     % Thickness of negative electrode [m]
p.L_s = 2E-5;% <--FROM Samsung %2.50E-05; %1.697e-5;     % Thickness of separator [m]
p.L_p = 11.9E-5;% <--FROM Samsung %3.66E-05; %6.521e-5;     % Thickness of positive electrode [m]

% Particle Radii
p.R_s_n = 6.1708e-07;%5e-7;%1e-7;%5.00E-7;%5.00E-06; %3.596e-6;    % Radius of solid particles in negative electrode [m]
p.R_s_p = 5.0000e-07;%5e-7;%1e-7;%5.00E-7; %1.637e-7;    % Radius of solid particles in positive electrode [m]

% Volume fractions
p.epsilon_s_n = 0.7215;%0.662; %0.3810;    % Volume fraction in solid for neg. electrode
p.epsilon_s_p = 0.6516;%0.58; %0.4800;    % Volume fraction in solid for pos. electrode

p.epsilon_f_n = 0.0789;%0;%Filler
p.epsilon_f_p = 0.0998;%0;%Filler

p.epsilon_e_n = 1 - p.epsilon_s_n - p.epsilon_f_n;%0.3; %0.6190;    % Volume fraction in electrolyte for neg. electrode
p.epsilon_e_s = 0.4; %0.3041;    % Volume fraction in electrolyte for separator
p.epsilon_e_p = 1 - p.epsilon_s_p - p.epsilon_f_p; %0.5200;    % Volume fraction in electrolyte for pos. electrode

% Specific interfacial surface area
p.a_s_n = 3*p.epsilon_s_n/p.R_s_n;  % Negative electrode [m^2/m^3]
p.a_s_p = 3*p.epsilon_s_p/p.R_s_p;  % Positive electrode [m^2/m^3]

%% Transport Params
```

- Embeds scalar parameter values
- Generates a struct object `p`, which is passed throughout the code

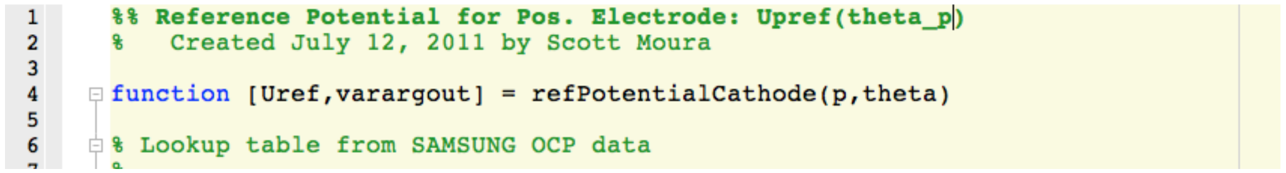
Tutorial Outline

1. Parameter file
2. Material function files
3. Input current data
4. Main script to run DFN model
5. Generate plots and animations

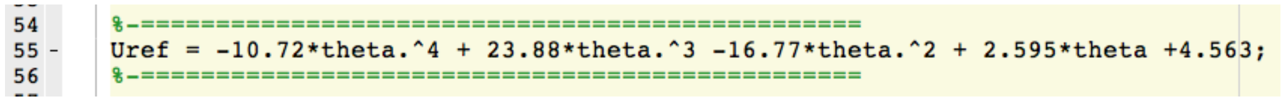
Open Circuit Potential Files

Open

- `refPotentialAnode.m`
- `refPotentialCathode.m`



```
%% Reference Potential for Pos. Electrode: Upref(theta_p)
%   Created July 12, 2011 by Scott Moura

function [Uref,varargout] = refPotentialCathode(p,theta)

% Lookup table from SAMSUNG OCP data
```



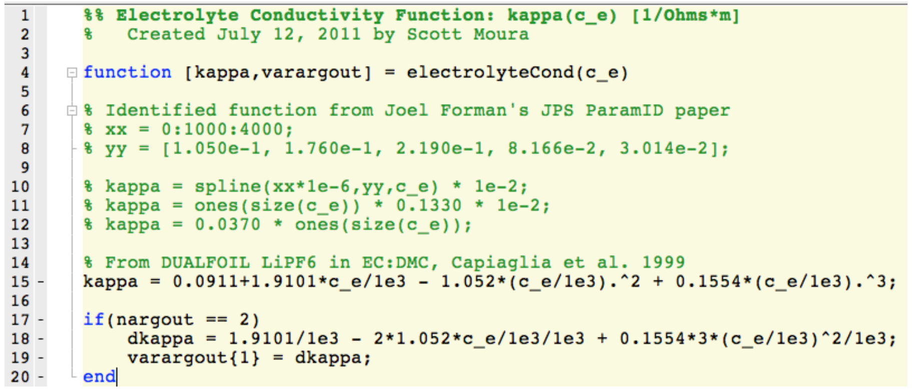
```
%-================================================
Uref = -10.72*theta.^4 + 23.88*theta.^3 -16.77*theta.^2 + 2.595*theta +4.563;
%-================================================
```

- Implements open circuit potentials for each electrode material
- Computes OCP, and it's derivative.

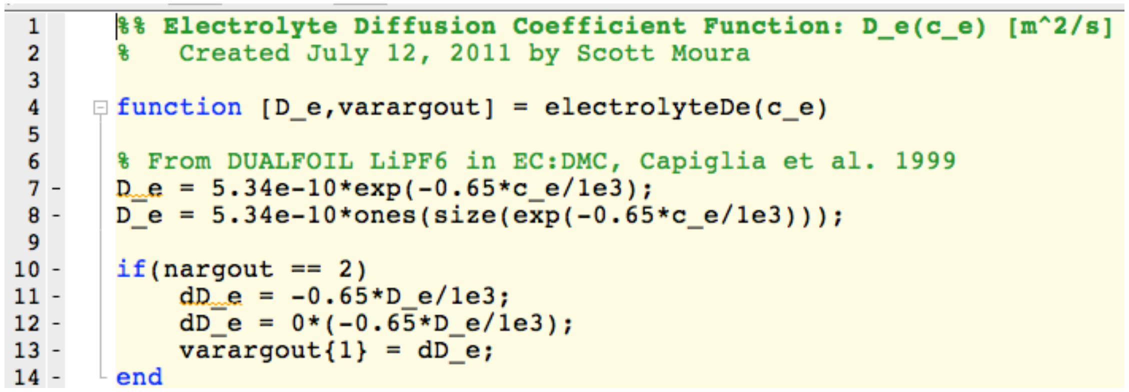
Electrolyte Conductivity & Diffusivity

Open

- `electrolyteCond.m`
- `electrolyteDe.m`



```
%% Electrolyte Conductivity Function: kappa(c_e) [1/Ohms*m]
%   Created July 12, 2011 by Scott Moura

function [kappa,varargout] = electrolyteCond(c_e)

% Identified function from Joel Forman's JPS ParamID paper
% xx = 0:1000:4000;
% yy = [1.050e-1, 1.760e-1, 2.190e-1, 8.166e-2, 3.014e-2];

% kappa = spline(xx*1e-6,yy,c_e) * 1e-2;
% kappa = ones(size(c_e)) * 0.1330 * 1e-2;
% kappa = 0.0370 * ones(size(c_e));

% From DUALFOIL LiPF6 in EC:DMC, Capiaglia et al. 1999
kappa = 0.0911+1.9101*c_e/1e3 - 1.052*(c_e/1e3).^2 + 0.1554*(c_e/1e3).^3;

if(nargout == 2)
    dkappa = 1.9101/1e3 - 2*1.052*c_e/1e3/1e3 + 0.1554*3*(c_e/1e3)^2/1e3;
    varargout{1} = dkappa;
end
```



```
%% Electrolyte Diffusion Coefficient Function: D_e(c_e) [m^2/s]
%   Created July 12, 2011 by Scott Moura

function [D_e,varargout] = electrolyteDe(c_e)

% From DUALFOIL LiPF6 in EC:DMC, Capiglia et al. 1999
D_e = 5.34e-10*exp(-0.65*c_e/1e3);
D_e = 5.34e-10*ones(size(exp(-0.65*c_e/1e3)));

if(nargout == 2)
    dD_e = -0.65*D_e/1e3;
    dD_e = 0*(-0.65*D_e/1e3);
    varargout{1} = dD_e;
end
```

- Electrolyte conductivity & diffusivity are functions of local concentration
- Computes conductivity/diffusivity and it's derivative

Tutorial Outline

1. Parameter file
2. Material function files
3. Input current data
4. Main script to run DFN model
5. Generate plots and animations

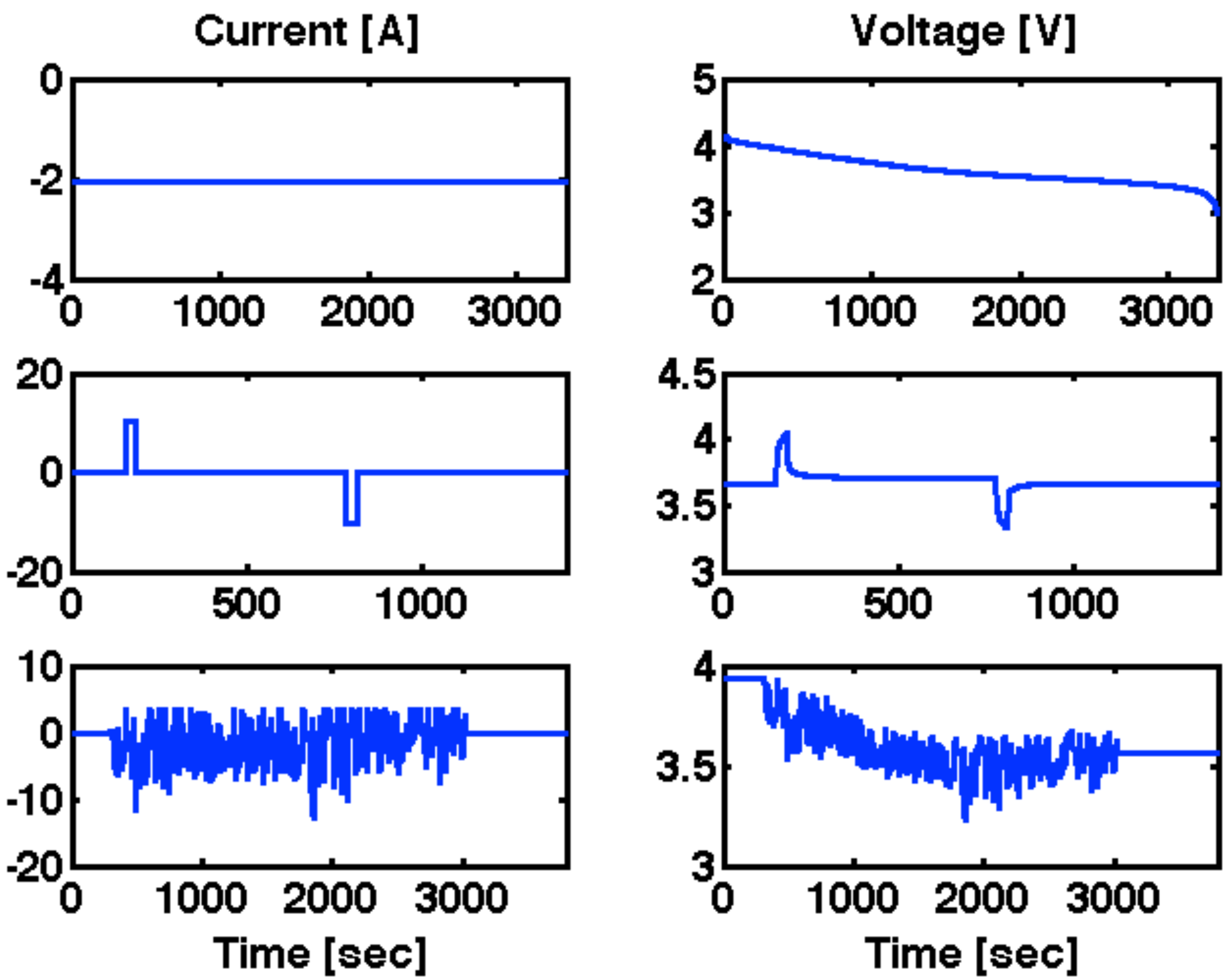
Input current files

- All experimentally collected data is located in: `data/Int_Obs/` directory
- Samples:

1C Discharge
`data/Int_Obs/1C_data_Oct_26_2015_05_sample.mat`

5C Pulses
`data/Int_Obs/dfn_5c.mat`

UDDS drive cycle
`data/Int_Obs/UDDS_data_Oct_26_2015_Sample_05sec.mat`



Tutorial Outline

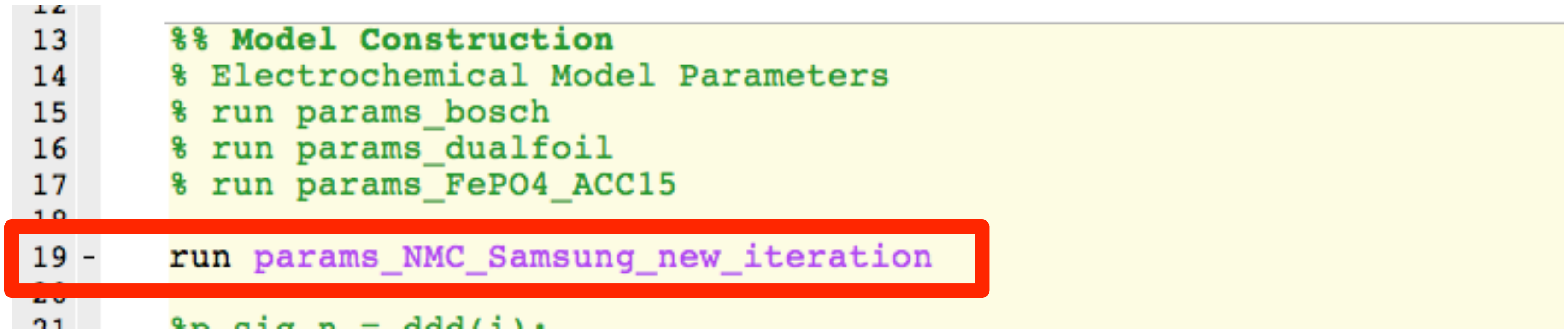
1. Parameter file
2. Material function files
3. Input current data
4. Main script to run DFN model
5. Generate plots and animations

Main Script

- Open dfn_scott_testing_Satadru.m

This is the main script that runs the DFN model

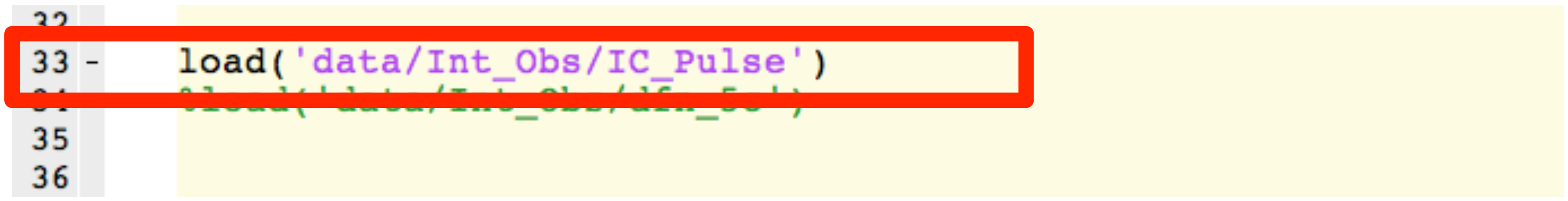
1. To input parameter file, go to line 19. Enter desired parameter file.



```
13      %% Model Construction
14      % Electrochemical Model Parameters
15      % run params_bosch
16      % run params_dualfoil
17      % run params_FePO4_ACC15
18
19 -    run params_NMC_Samsung_new_iteration
20
21      %p_sig_n = ddd(i);
```

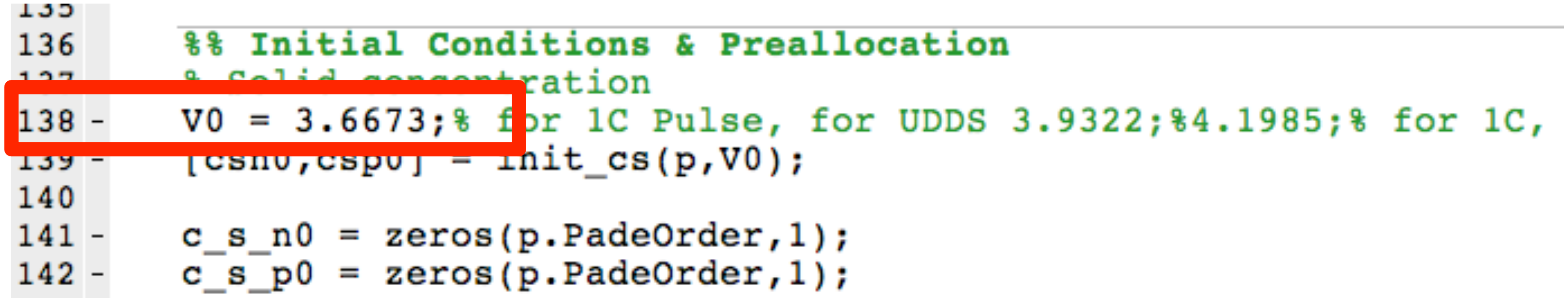
Main Script

2. To load input current, go to line 33. Enter desired input current file.



```
32
33 -    load('data/Int_Obs/IC_Pulse')
34      %load('data/Int_Obs/dfn_50')
35
36
```

3. To set initial voltage, go to line 138.

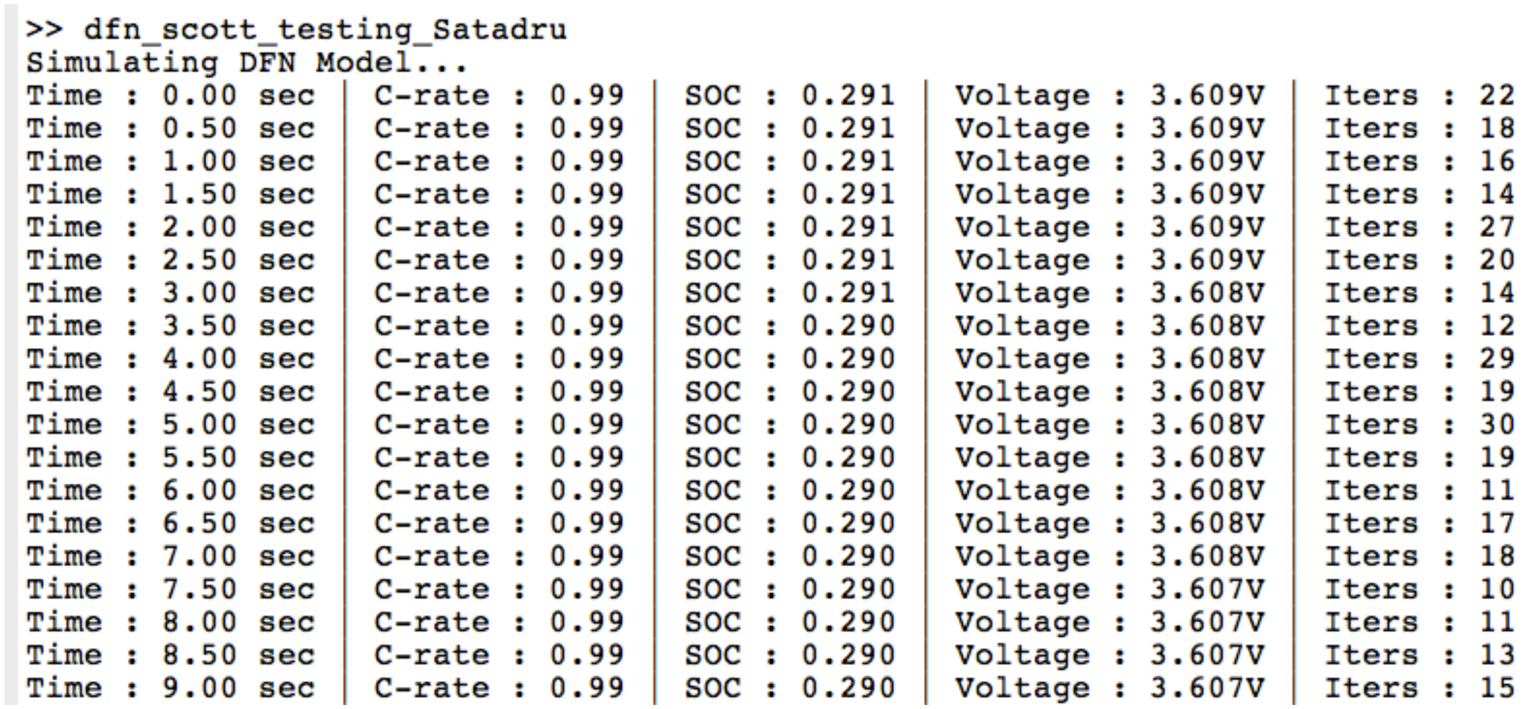


```
135
136     %% Initial Conditions & Preallocation
137     % Solid concentration
138 -   V0 = 3.6673;% for 1C Pulse, for UDDS 3.9322;%4.1985;% for 1C,
139 -   [csn0,csp0] = init_cs(p,V0);
140
141 -   c_s_n0 = zeros(p.PadeOrder,1);
142 -   c_s_p0 = zeros(p.PadeOrder,1);
```

Main Script

4. You are ready! Run the m-file.

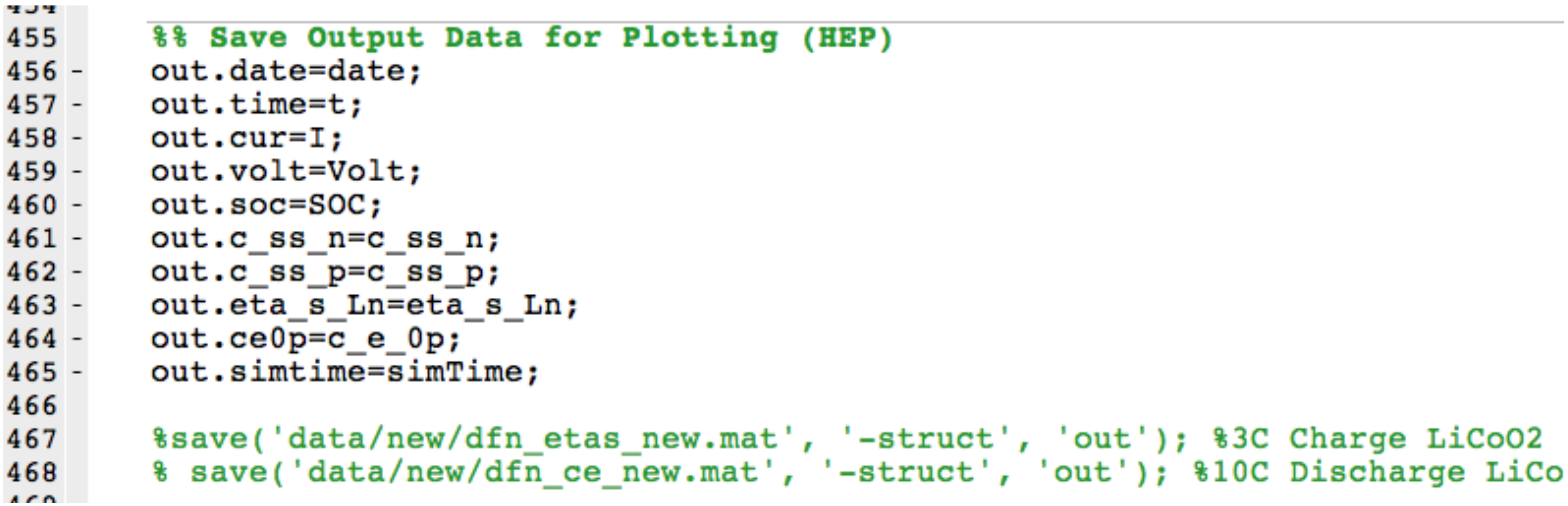
The command window output will look like…



```
>> dfn_scott_testing_Satadru
Simulating DFN Model...
Time : 0.00 sec | C-rate : 0.99 | SOC : 0.291 | Voltage : 3.609V | Iters : 22
Time : 0.50 sec | C-rate : 0.99 | SOC : 0.291 | Voltage : 3.609V | Iters : 18
Time : 1.00 sec | C-rate : 0.99 | SOC : 0.291 | Voltage : 3.609V | Iters : 16
Time : 1.50 sec | C-rate : 0.99 | SOC : 0.291 | Voltage : 3.609V | Iters : 14
Time : 2.00 sec | C-rate : 0.99 | SOC : 0.291 | Voltage : 3.609V | Iters : 27
Time : 2.50 sec | C-rate : 0.99 | SOC : 0.291 | Voltage : 3.609V | Iters : 20
Time : 3.00 sec | C-rate : 0.99 | SOC : 0.291 | Voltage : 3.608V | Iters : 14
Time : 3.50 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.608V | Iters : 12
Time : 4.00 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.608V | Iters : 29
Time : 4.50 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.608V | Iters : 19
Time : 5.00 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.608V | Iters : 30
Time : 5.50 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.608V | Iters : 19
Time : 6.00 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.608V | Iters : 11
Time : 6.50 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.608V | Iters : 17
Time : 7.00 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.608V | Iters : 18
Time : 7.50 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.607V | Iters : 10
Time : 8.00 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.607V | Iters : 11
Time : 8.50 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.607V | Iters : 13
Time : 9.00 sec | C-rate : 0.99 | SOC : 0.290 | Voltage : 3.607V | Iters : 15
```

Main Script

5. After running, the simulation outputs of interest are saved to struct object `out`. Go to line 467, uncomment, and save to a user-specified filename.



```
455     %% Save Output Data for Plotting (HEP)
456 -   out.date=date;
457 -   out.time=t;
458 -   out.cur=I;
459 -   out.volt=Volt;
460 -   out.soc=SOC;
461 -   out.c_ss_n=c_ss_n;
462 -   out.c_ss_p=c_ss_p;
463 -   out.eta_s_Ln=eta_s_Ln;
464 -   out.ce0p=c_e_0p;
465 -   out.simtime=simTime;
466
467     %save('data/new/dfn_etas_new.mat', '-struct', 'out'); %3C Charge LiCoO2
468     % save('data/new/dfn_ce_new.mat', '-struct', 'out'); %10C Discharge LiCo
```

Main Script – Ex: HEV Validation

To reproduce the model vs. experiment validation on the Samsung HEV cycle…

1. Use `params_NMC_Samsung_new_iteration.m`
2. Load `data/Int_Obs/Samsung_HEV_data.mat` cycle
3. Initialive model voltage to equal initial voltage from experimental data on line 138.
4. Run `dfn_scott_testing_Satadru.m`
5. The model will require 30min or so. After it's finished, then the last code (line 507) will compute the RMS error between experiments and model

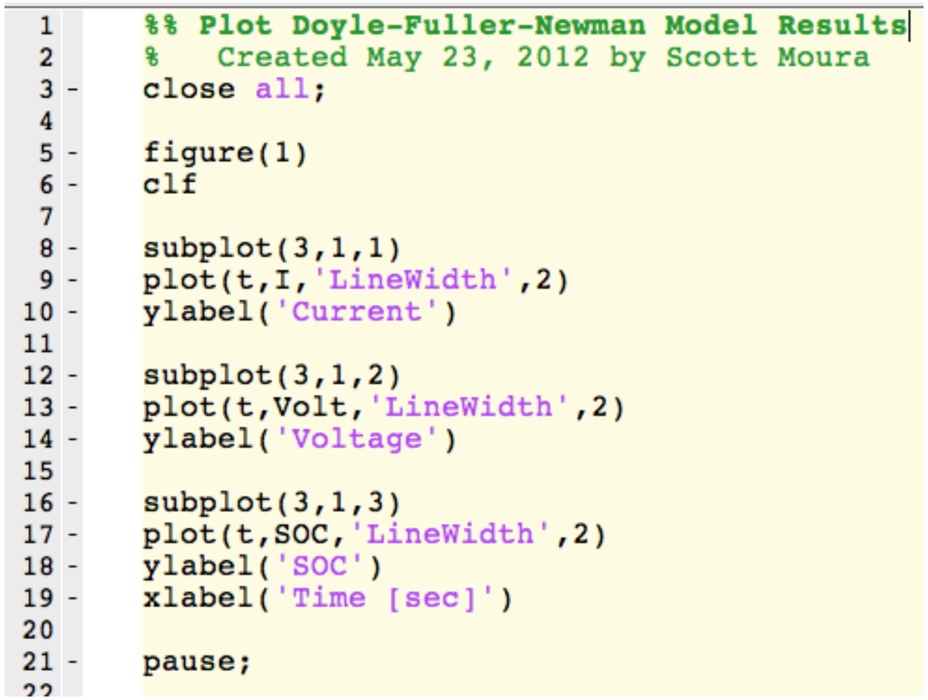
Tutorial Outline

1. Parameter file
2. Material function files
3. Input current data
4. Main script to run DFN model
5. Generate plots and animations

Generate Plots and Animations

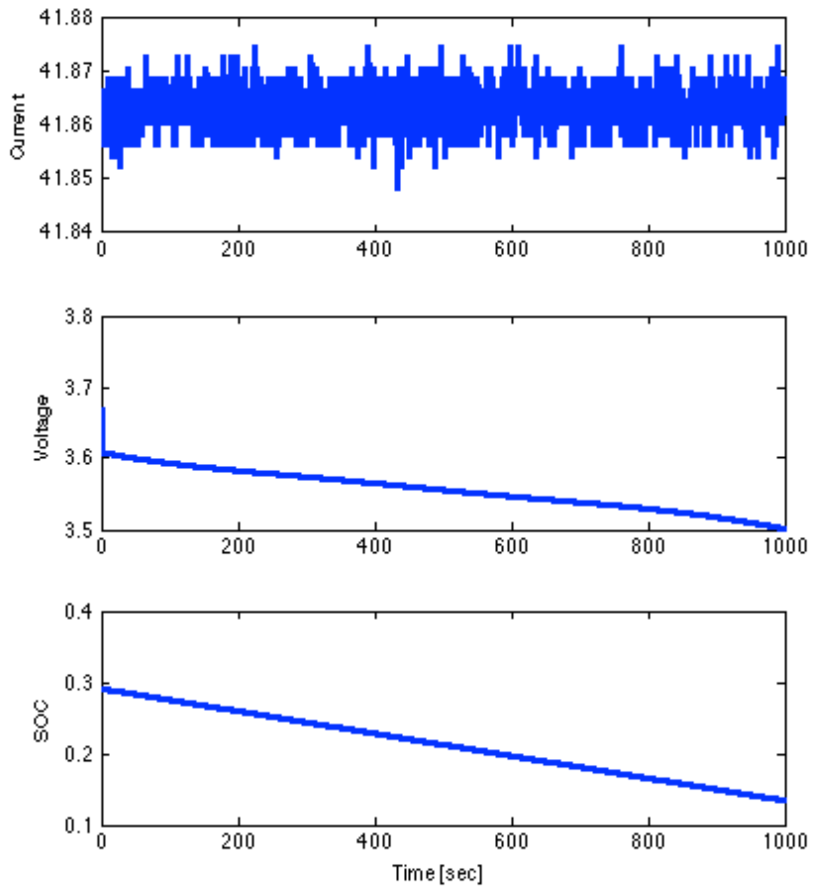
Open `plot_dfn.m`

1. Plot basic outputs: current, SOC, and voltage



```
%% Plot Doyle-Fuller-Newman Model Results
%   Created May 23, 2012 by Scott Moura
close all;

figure(1)
clf

subplot(3,1,1)
plot(t,I,'LineWidth',2)
ylabel('Current')

subplot(3,1,2)
plot(t,Volt,'LineWidth',2)
ylabel('Voltage')

subplot(3,1,3)
plot(t,SOC,'LineWidth',2)
ylabel('SOC')
xlabel('Time [sec]')

pause;
```

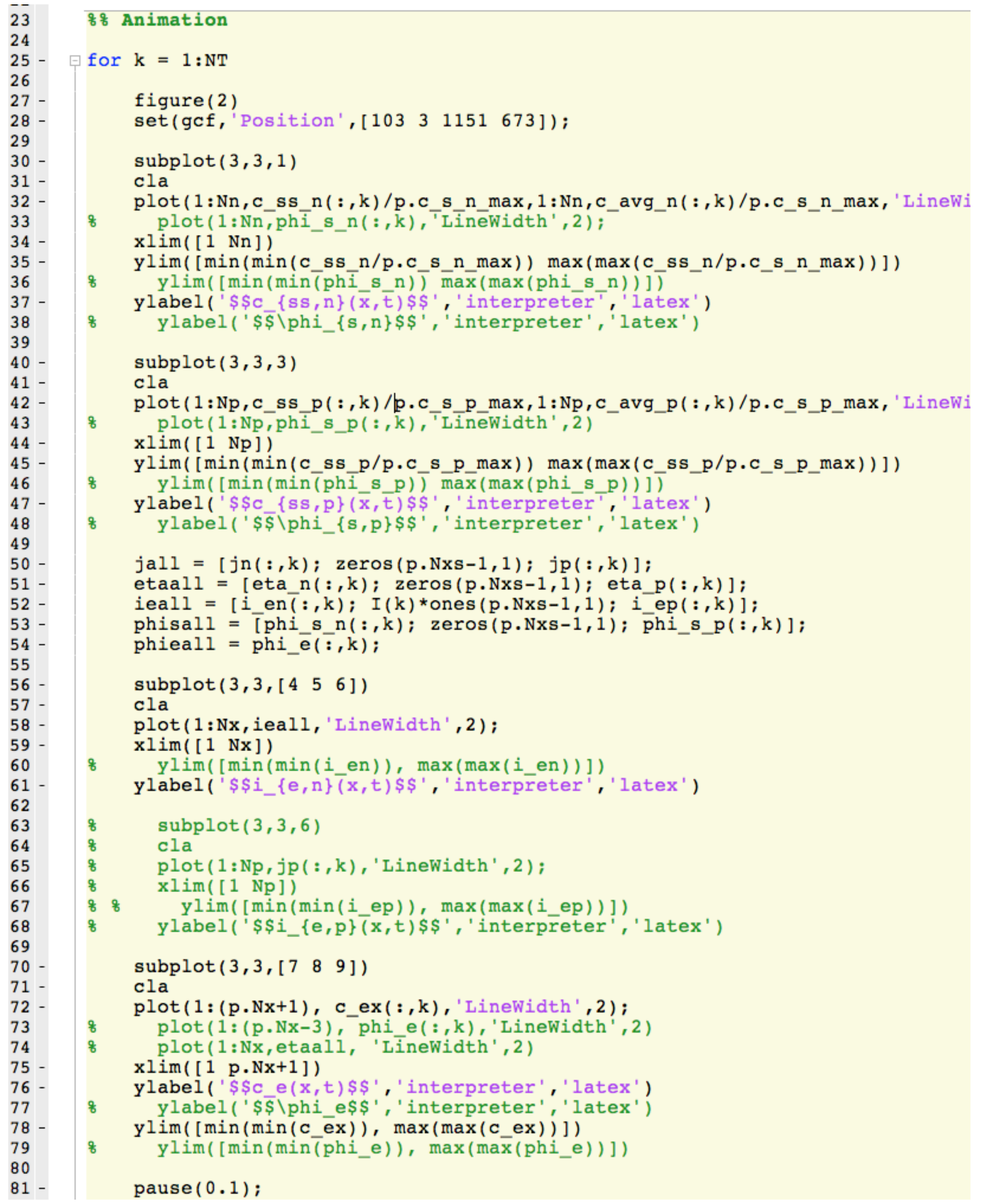
Example: 1C discharge for 1000 sec



Generate Plots and Animations

Open `plot_dfn.m`

2. Animate electrochemical states, e.g. solid & electrolyte concentrations and ionic current. Please feel free to uncomment/comment parts of code to view different echem states.



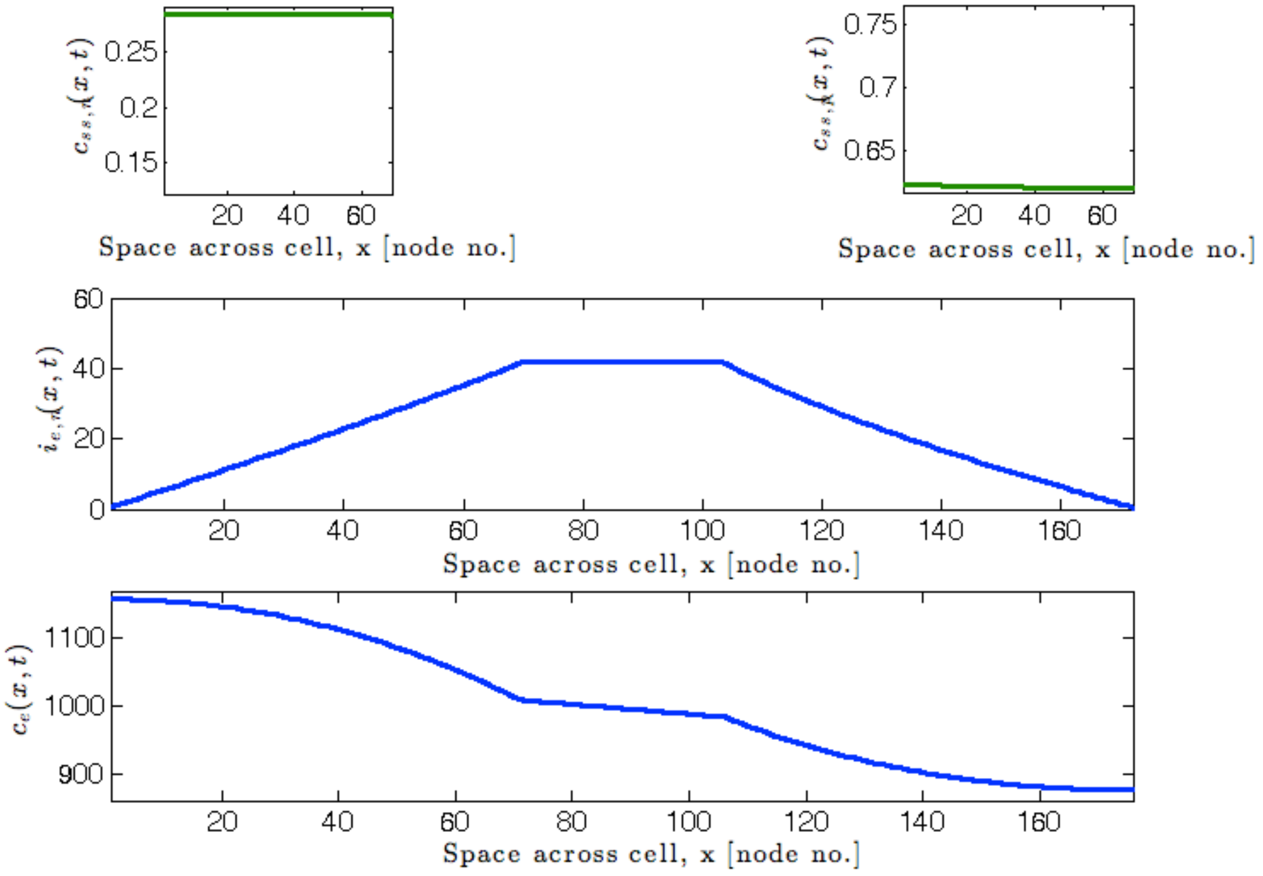
```
%% Animation

for k = 1:NT

    figure(2)
    set(gcf,'Position',[103 3 1151 673]);

    subplot(3,3,1)
    cla
    plot(1:Nn,c_ss_n(:,k)/p.c_s_n_max,1:Nn,c_avg_n(:,k)/p.c_s_n_max,'LineWi
%     plot(1:Nn,phi_s_n(:,k),'LineWidth',2);
    xlim([1 Nn])
    ylim([min(min(c_ss_n/p.c_s_n_max)) max(max(c_ss_n/p.c_s_n_max))])
%     ylim([min(min(phi_s_n)) max(max(phi_s_n))])
    ylabel('$$c_{ss,n}(x,t)$$','interpreter','latex')
%     ylabel('$$\phi_{s,n}$$','interpreter','latex')

    subplot(3,3,3)
    cla
    plot(1:Np,c_ss_p(:,k)/p.c_s_p_max,1:Np,c_avg_p(:,k)/p.c_s_p_max,'LineWi
%     plot(1:Np,phi_s_p(:,k),'LineWidth',2)
    xlim([1 Np])
    ylim([min(min(c_ss_p/p.c_s_p_max)) max(max(c_ss_p/p.c_s_p_max))])
%     ylim([min(min(phi_s_p)) max(max(phi_s_p))])
    ylabel('$$c_{ss,p}(x,t)$$','interpreter','latex')
%     ylabel('$$\phi_{s,p}$$','interpreter','latex')

    jall = [jn(:,k); zeros(p.Nxs-1,1); jp(:,k)];
    etaall = [eta_n(:,k); zeros(p.Nxs-1,1); eta_p(:,k)];
    ieall = [i_en(:,k); I(k)*ones(p.Nxs-1,1); i_ep(:,k)];
    phisall = [phi_s_n(:,k); zeros(p.Nxs-1,1); phi_s_p(:,k)];
    phieall = phi_e(:,k);

    subplot(3,3,[4 5 6])
    cla
    plot(1:Nx,ieall,'LineWidth',2);
    xlim([1 Nx])
%     ylim([min(min(i_en)), max(max(i_en))])
    ylabel('$$i_{e,n}(x,t)$$','interpreter','latex')

%     subplot(3,3,6)
%     cla
%     plot(1:Np,jp(:,k),'LineWidth',2);
%     xlim([1 Np])
% %     ylim([min(min(i_ep)), max(max(i_ep))])
%     ylabel('$$i_{e,p}(x,t)$$','interpreter','latex')

    subplot(3,3,[7 8 9])
    cla
    plot(1:(p.Nx+1), c_ex(:,k),'LineWidth',2);
%     plot(1:(p.Nx-3), phi_e(:,k),'LineWidth',2)
%     plot(1:Nx,etaall, 'LineWidth',2)
    xlim([1 p.Nx+1])
    ylabel('$$c_e(x,t)$$','interpreter','latex')
%     ylabel('$$\phi_e$$','interpreter','latex')
    ylim([min(min(c_ex)), max(max(c_ex))])
%     ylim([min(min(phi_e)), max(max(phi_e))])

    pause(0.1);
```

Generate Plots and Animations

Open `plot_dfn.m`

2. Animate electrochemical states, e.g. solid & electrolyte concentrations and ionic current.

Example: Snapshot of 1C discharge animation



Tutorial Outline

1. Parameter file
2. Material function files
3. Input current data
4. Main script to run DFN model
5. Generate plots and animations

Closing remarks: This code is under development. Future versions will have enhanced speed, accuracy, usability, etc. Enjoy!