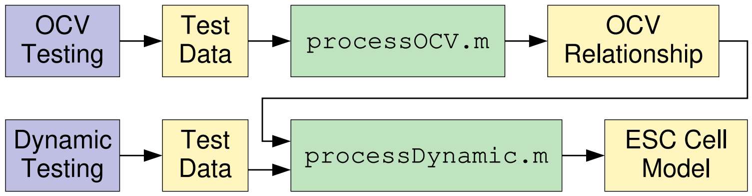
Creating the model

- Figure depicts overall process for creating an ESC cell model
- Blue boxes = laboratory processes; yellow boxes = data files; green boxes = processing by Octave/MATLAB functions



- In this lesson, you will learn the main details of `processOCV.m`
- I recommend that you study this alongside notes for Lessons 2.2.2 through 2.2.4

Preliminary comments (1)

```
% ------------------------------------------------------------------------
% function processOCV
%
% PROCESSOCV assumes that specific cell test scripts have been run to generate
% the input data structure having fields for time, step, current, voltage, chgAh
% and disAh for each script run. The results from four scripts are required at
% every temperature. The steps in each script file are assumed to be:
%   Script 1 (thermal chamber set to test temperature):
%     Step 1: Rest @ 100% SOC to acclimatize to test temperature
%     Step 2: Discharge @ low rate (ca. C/30) to min voltage
%     Step 3: Rest ca. 0%
%   Script 2 (thermal chamber set to 25 degC):
%     Step 1: Rest ca. 0% SOC to acclimatize to 25 degC
%     Step 2: Discharge to min voltage (ca. C/3)
%     Step 3: Rest
%     Step 4: Const voltage at vmin until current small (ca. C/30)
%     Steps 5-7: Dither around vmin
%     Step 8: Rest
%     Step 9: Constant voltage at vmin for 15 min
%     Step 10: Rest
```


Preliminary comments (2)

```
%   Script 3 (thermal chamber set to test temperature):
%     Step 1: Rest at 0% SOC to acclimatize to test temp
%     Step 2: Charge @ low rate (ca. C/30) to max voltage
%     Step 3: Rest
%   Script 4 (thermal chamber set to 25 degC):
%     Step 1: Rest ca. 100% SOC to acclimatize to 25 degC
%     Step 2: Charge to max voltage (ca. C/3)
%     Step 3: Rest
%     Step 4: Const voltage at vmax until current small (ca. C/30)
%     Steps 5-7: Dither around vmax
%     Step 8: Rest
%     Step 9: Constant voltage at vmax for 15 min
%     Step 10: Rest
%
% All other steps (if present) are ignored by PROCESSOCV. The time
% step between data samples is not critical since the Arbin
% integrates ampere-hours to produce the two Ah columns, and this
% is what is necessary to generate the OCV curves.  The rest steps
% must contain at least one data point each.
```

Beginning of function

- Will concentrate on code highlights—not entire code
- Define function, check for existence of 25 °C data

```
function model=processOCV(data,cellID)
  filetemps = [data.temp]; filetemps = filetemps(:);
  numtemps = length(filetemps);

  ind25 = find(filetemps == 25);
  if isempty(ind25),
    error('Must have a test at 25degC');
  end
  not25 = find(filetemps ~= 25);
  data25 = data(ind25);
```

- `data` contains all measured cell-test data; `cellID` is cell's name, saved in final output structure

Compute coulombic efficiency, capacity

- Compute $\eta(25\,°\mathrm{C})$ and $Q(25\,°\mathrm{C})$

```
  % Compute total dis/charge ampere hours
  totDisAh = data25.script1.disAh(end) + ...
             data25.script2.disAh(end) + ...
             data25.script3.disAh(end) + ...
             data25.script4.disAh(end);
  totChgAh = data25.script1.chgAh(end) + ...
             data25.script2.chgAh(end) + ...
             data25.script3.chgAh(end) + ...
             data25.script4.chgAh(end);
  eta25 = totDisAh/totChgAh;

  data25.script1.chgAh = data25.script1.chgAh*eta25;
  data25.script2.chgAh = data25.script2.chgAh*eta25;
  data25.script3.chgAh = data25.script3.chgAh*eta25;
  data25.script4.chgAh = data25.script4.chgAh*eta25;

  Q25 = data25.script1.disAh(end)+data25.script2.disAh(end) - ...
        data25.script1.chgAh(end)-data25.script2.chgAh(end);
```


Compute R_0 estimates

- Compute voltage changes at both ends of dis/charge data
- Limit discharge/charge voltage changes to no more than two times corresponding voltage change in charge/discharge data

```
  indD  = find(data25.script1.step == 2);       % Slow discharge step
  IR1Da = data25.script1.voltage(indD(1)-1) - ...
          data25.script1.voltage(indD(1));      % At beginning of discharge step
  IR2Da = data25.script1.voltage(indD(end)+1) - ...
          data25.script1.voltage(indD(end));    % At end of discharge step
  indC  = find(data25.script3.step == 2);       % Slow charge step
  IR1Ca = data25.script3.voltage(indC(1)) - ...
          data25.script3.voltage(indC(1)-1);    % At beginning of charge step
  IR2Ca = data25.script3.voltage(indC(end)) - ...
          data25.script3.voltage(indC(end)+1); % At end of charge step

  IR1D = min(IR1Da,2*IR2Ca); IR2D = min(IR2Da,2*IR1Ca); % Limit discharge delta V
  IR1C = min(IR1Ca,2*IR2Da); IR2C = min(IR2Ca,2*IR1Da); % Limit charge delta V
```

Adjust voltage curves

- Compensate dis/charge curves for $R_0 i(t)$
  - Compute modified `disV`, then `chgV`

```
blend = (0:length(indD)-1)/(length(indD)-1);
IRblend = IR1D + (IR2D-IR1D)*blend(:);
disV = data(k).script1.voltage(indD) + IRblend;
disZ = 1 - data25.script1.disAh(indD)/Q25;
disZ = disZ + (1 - disZ(1)); % force initial 100% SOC

blend = (0:length(indC)-1)/(length(indC)-1);
IRblend = IR1C + (IR2C-IR1C)*blend(:);
chgV = data25.script3.voltage(indC) - IRblend;
chgZ = data25.script3.chgAh(indC)/Q25;
chgZ = chgZ - chgZ(1); % force initial 0% SOC
```


Compensate for steady-state resistance

- Compensate for steady-state resistance
  - `rawocv` midway between dis/charge voltages at 50 % SOC

```
deltaV50 = interp1(chgZ,chgV,0.5) - interp1(disZ,disV,0.5);

ind = find(chgZ < 0.5);
vChg = chgV(ind) - chgZ(ind)*deltaV50;
zChg = chgZ(ind);

ind = find(disZ > 0.5);
vDis = flipud(disV(ind) + (1 - disZ(ind))*deltaV50);
zDis = flipud(disZ(ind));

rawocv = interp1([zChg; zDis],[vChg; vDis],SOC,'linear','extrap');
filedata(ind25).rawocv = rawocv;
filedata(ind25).temp = data25.temp;
```

- Then, repeat same basic procedure for data collected at all other temperatures

Make final relationship

- Finally, use all approximate OCV relationships together to find `OCV0` and `OCVrel` relationships

```
% Compile voltages and temperatures into arrays rather than a structure
Vraw = []; temps = [];
for k = 1:numtemps,
  if filedata(k).temp > 0,
    Vraw = [Vraw; filedata(k).rawocv]; %#ok<AGROW>
    temps = [temps; filedata(k).temp]; %#ok<AGROW>
  end
end
% Perform least-squares fit of model to data
X = [ones(size(temps)), temps] \ Vraw;
model.OCV0 = X(1,:);
model.OCVrel = X(2,:);
model.SOC = SOC;
```

Summary

- `processOCV.m` computes OCV relationship from lab-test data
  - Code first computes $\eta(25\,^\circ\text{C})$ and $Q(25\,^\circ\text{C})$
  - Then adjusts dis/charge curves to compensate for estimated R_0
  - Computes approximate OCV versus SOC, compensating for steady-state resistance
  - Repeats above for all other test temperatures
  - Finally, computes `OCV0` and `OCVrel`, combining data from all temperatures
  - Results saved to a model file
- You will later learn how to use this model file to simulate a battery cell