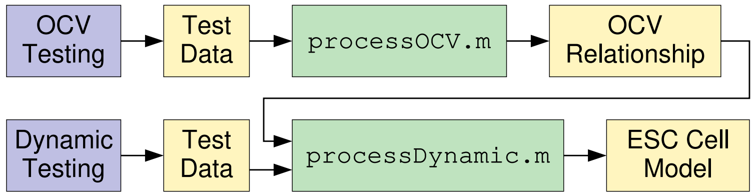
## Creating the model

- 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 `processDynamic.m`
- I recommend that you study this alongside notes for Lessons 2.3.1 through 2.3.2





## Preliminary comments (1)

```
% ------------------------------------------------------------------
% function processDynamic
%
% PROCESSDYNAMIC assumes that specific cell test scripts have been run to gen-
% erate the input data structure having fields for time, step, current, voltage,
% chgAh, and disAh for each script run. The results from three 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 @ 1C to reach ca. 90% SOC
%     Step 3: Repeatedly execute dynamic profiles (and possibly
%             intermediate rests) until SOC is around 10%
%   Script 2 (thermal chamber set to 25 degC):
%     Step 1: Rest ca. 10% SOC to acclimatize to 25 degC
%     Step 2: Discharge to min voltage (ca. C/3)
%     Step 3: Rest
%     Step 4: Constant voltage at vmin until current small (ca. C/30)
%     Steps 5-7: Dither around vmin
%     Step 8: Rest
```





## Preliminary comments (2)

```
%   Script 3 (thermal chamber set to 25 degC):
%     Step 2: Charge @ 1C to max voltage
%     Step 3: Rest
%     Step 4: Constant voltage at vmax until current small (ca. C/30)
%     Steps 5-7: Dither around vmax
%     Step 8: Rest
% All other steps (if present) are ignored by PROCESSDYNAMIC. The time step
% between data samples must be uniform--we assume a 1s sample period in this code
%
% The inputs:
% - data: An array, with one entry per temperature to be processed.
%         One of the array entries must be at 25 degC. The fields of
%         "data" are: temp (the test temperature), script1,
%         script 2, and script 3, where the latter comprise data
%         collected from each script.  The sub-fields of these script
%         structures that are used by PROCESSDYNAMIC are the vectors:
%         current, voltage, chgAh, and disAh
% - model: The output from processOCV, comprising the OCV model
% - numpoles: The number of R-C pairs in the model
% - doHyst: 0 if no hysteresis model desired; 1 if hysteresis desired
```

## Preliminary comments (3); function header

```
% The output:
% - model: A modified model, which now contains the dynamic
%          fields filled in.

function model = processDynamic(data,model,numpoles,doHyst)
  global bestcost

  % used by fminbnd later on
  options=optimset('TolX',1e-8,'TolFun',1e-8,'MaxFunEval',100000, ...
    'MaxIter',1e6,'Jacobian','Off'); % for later optimization

  % ------------------------------------------------------------------
  % Step 1: Determine coulombic efficiency and capacity
  % ------------------------------------------------------------------
  % Code omitted here. See Lesson 2.2.5 for similar code + description
```





## Compute OCV, subtract from $v[k]$

```
  % ------------------------------------------------------------------
  % Step 2: Compute OCV for "discharge portion" of test
  % ------------------------------------------------------------------
  for k = 1:length(data),
    etaParam = model.etaParam(k);     % retrieve eta for this test
    etaik = data(k).script1.current; % modify current using eta
    etaik(etaik<0)= etaParam*etaik(etaik<0);
    data(k).Z = 1 - cumsum([0,etaik(1:end-1)])*1/(data(k).Q*3600); % SOC
    data(k).OCV = OCVfromSOCtemp(data(k).Z(:),alltemps(k),model);  % OCV
  end % OCV is actually subtracted from voltage later on...

  % ------------------------------------------------------------------
  % Step 3: Set up optimization, optimize!
  % ------------------------------------------------------------------
  model.GParam  = NaN(1,numTemps); % "gamma" hysteresis parameter
  model.M0Param = NaN(1,numTemps); % "M0" hysteresis parameter
  model.MParam  = NaN(1,numTemps); % "M" hysteresis parameter
  model.R0Param = NaN(1,numTemps); % "R0" ohmic resistance parameter
  model.RCParam = NaN(numTemps,numpoles); % time const.
  model.RParam  = NaN(numTemps,numpoles); % Rk
```





## The code that invokes the optimizations

- The following code invokes the optimizations (discussed on following slides)

```
  for theTemp = 1:numTemps,
    fprintf('Processing temperature %d\n',model.temps(theTemp));
    bestcost = Inf;
    if doHyst,
      model.GParam(theTemp) = abs(fminbnd(@(x) optfn(x,data,...
                                model,model.temps(theTemp),...
                                doHyst),1,250,options));
    else
      model.GParam(theTemp) = 0; theGParam = 0;
      % call optfn to display plots, if desired
      optfn(theGParam,data,model,model.temps(theTemp),doHyst);
    end
    % set final model fields
    [~,model] = minfn(data,model,model.temps(theTemp),doHyst);
  end
return % from processDynamic
```

## Function to be minimized by optimization

- `optfn` is nested function called by `fminbnd.m` to optimize $\gamma$
- It can also update plots of best parameter values found to date

```
% This function has the correct syntax to be invoked by fminbnd.m to optimize
% the model parameter values (esp. gamma), returning the rms model error "cost"
function cost=optfn(theGParam,data,model,theTemp,doHyst)
  global bestcost

  model.GParam(model.temps == theTemp) = abs(theGParam);
  [cost,model] = minfn(data,model,theTemp,doHyst);
  if cost<bestcost,
    bestcost = cost;
    disp('Best ESC model values yet!');
    % You can plot some things here if you want to
  end
return
```





## Given $\gamma$, find best model parameter values (1)

- `minfn` is the function that does most of the work
- Given OCV and $\gamma$, find the remaining parameter values

```
% ------------------------------------------------------------------
% Using an assumed value for gamma (already stored in the model), find
% optimum values for remaining cell parameters, and compute the RMS
% error between true and predicted cell voltage
% ------------------------------------------------------------------
function [cost,model]=minfn(data,model,theTemp,doHyst)
  alltemps = [data(:).temp];
  ind = find(alltemps == theTemp); numfiles = length(ind);
  rmserr = zeros(1,numfiles);

  G = abs(getParamESC('GParam',theTemp,model));
  Q = abs(getParamESC('QParam',theTemp,model));
  eta = abs(getParamESC('etaParam',theTemp,model));
  RC = getParamESC('RCParam',theTemp,model);
  numpoles = length(RC);
```





## Given $\gamma$, find best model parameter values (2)

- `minfn` is the function that does most of the work
- Given OCV and $\gamma$, find the remaining parameter values

```
 for thefile = 1:numfiles;
    ik = data(ind(thefile)).script1.current(:);
    vk = data(ind(thefile)).script1.voltage(:);
    tk = (1:length(vk))-1;
    etaik = ik; etaik(ik<0) = etaik(ik<0)*eta;

    h=0*ik; sik = 0*ik;
    fac=exp(-abs(G*etaik/(3600*Q)));
    for k=2:length(ik),
      h(k)=fac(k-1)*h(k-1)-(1-fac(k-1))*sign(ik(k-1));
      sik(k) = sign(ik(k));
      if abs(ik(k))<Q/100, sik(k) = sik(k-1); end
    end
```

## Given $\gamma$, find best model parameter values (3)

- minfn is the function that does most of the work
- Given OCV and $\gamma$, find the remaining parameter values

```
    % First modeling step: Compute error with model = OCV only
    vest1 = data(ind(thefile)).OCV; verr = vk - vest1;

    % Second modeling step: Compute time constants in "A" matrix
    A = SISOsubid(-diff(verr),diff(etaik),numpoles);
    % modify results to ensure real, preferably distinct, between 0 and 1
    eigA = eig(A); eigAr = eigA + 0.001*randn(size(eigA));
    eigA(eigA ~= conj(eigA)) = abs(eigAr(eigA ~= conj(eigA)));
    eigA(eigA<0) = abs(eigA(eigA<0)); eigA(eigA>1) = 1./eigA(eigA>1);
    RCfact = sort(eigA); RCfact = RCfact(end-numpoles+1:end);
    RC = -1./log(RCfact);

    % Simulate the R-C filters to find R-C currents
    vrcRaw = dlsim(diag(RCfact),1-RCfact,eye(numpoles),zeros(numpoles,1),etaik);
```





## Given $\gamma$, find best model parameter values (4)

- minfn is the function that does most of the work
- Given OCV and $\gamma$, find the remaining parameter values

```
    % Third modeling step: Hysteresis parameters
    if doHyst,
      H = [h,sik,-etaik,-vrcRaw];
      W = lsqnonneg(H,verr); % or, W = H\verr;
      M = W(1); M0 = W(2); R0 = W(3); Rfact = W(4:end)';
    else
      H = [-etaik,-vrcRaw];
      W = H\verr;
      M=0; M0=0; R0 = W(1); Rfact = W(2:end)';
    end
    ind = find(model.temps == data(ind(thefile)).temp,1);
    model.M0Param(ind) = M0;        model.MParam(ind) = M;
    model.RCParam(ind,:) = RC';     model.RParam(ind,:) = Rfact';
    model.R0Param(ind) = R0;
```





## Given $\gamma$, find best model parameter values (5)

- minfn is the function that does most of the work
- Given OCV and $\gamma$, find the remaining parameter values

```
    vest2 = vest1 + M*h + M0*sik - R0*etaik - vrcRaw*Rfact';
    verr = vk - vest2;

    % Compute RMS error only on data roughly in 5% to 95% SOC
    v1 = OCVfromSOCtemp(0.95,data(ind(thefile)).temp,model);
    v2 = OCVfromSOCtemp(0.05,data(ind(thefile)).temp,model);
    N1 = find(vk<v1,1,'first'); N2 = find(vk<v2,1,'first');
    if isempty(N1), N1=1; end; if isempty(N2), N2=length(verr); end
    rmserr(thefile)=sqrt(mean(verr(N1:N2).^2));
  end % for thefile = 1:numfiles

  cost=sum(rmserr);
  fprintf('RMS error = %0.2f (mV)\n',cost*1000);
  if isnan(cost), stop, end
return % end of minfn
```

## Table of `model` fields (1)

- The following table lists fields in the `model` data structure

| | |
|---|---|
| *Identifier field* | |
| `name` | An identifying string storing a name for the cell type |
| *Fields pertaining to the OCV versus SOC relationship* | |
| `OCV0` | Vector of OCV versus SOC at 0 °C [ V] |
| `OCVrel` | Vector of change in OCV versus SOC per °C [ V/°C] |
| `SOC` | SOC vector at which `OCV0` and `OCVrel` are stored |
| `SOC0` | Vector of SOC versus OCV at 0 °C (unitless) |
| `SOCrel` | Vector of change in SOC versus OCV per °C [ 1/°C] |
| `OCV` | OCV vector at which `SOC0` and `SOCrel` are stored |

(continued. . . )





## Table of `model` fields (2)

- Fields in the `model` data structure (continued)

| | |
|---|---|
| *Fields pertaining to the dynamic relationship* | |
| `temps` | Temperatures at which dynamic parameters stored [ °C] |
| `QParam` | Capacity $Q$ at each temperature [ Ah] |
| `etaParam` | Coulombic efficiency $\eta$ at each temperature (unitless) |
| `GParam` | Hysteresis “gamma” parameter $\gamma$ (unitless) |
| `MParam` | Hysteresis $M$ parameter [ V] |
| `M0Param` | Hysteresis $M_0$ parameter [ V] |
| `R0Param` | Series resistance parameter $R_0$ [Ω] |
| `RCParam` | The R–C time constant parameter $R_jC_j$ [ s] |
| `RParam` | Resistance $R_j$ of the R–C parameter [Ω] |





## Summary

- In this lesson, you have seen Octave/MATLAB code to optimize model parameter values
  - Coulombic efficiency and capacity calculated directly from data, allowing computation of OCV
  - Subspace system identification method finds R–C time constants
  - Line-search algorithm optimizes hysteresis rate constant $\gamma$
  - All other values found via least-squares fitting
- Code should match theory in earlier lessons quite closely