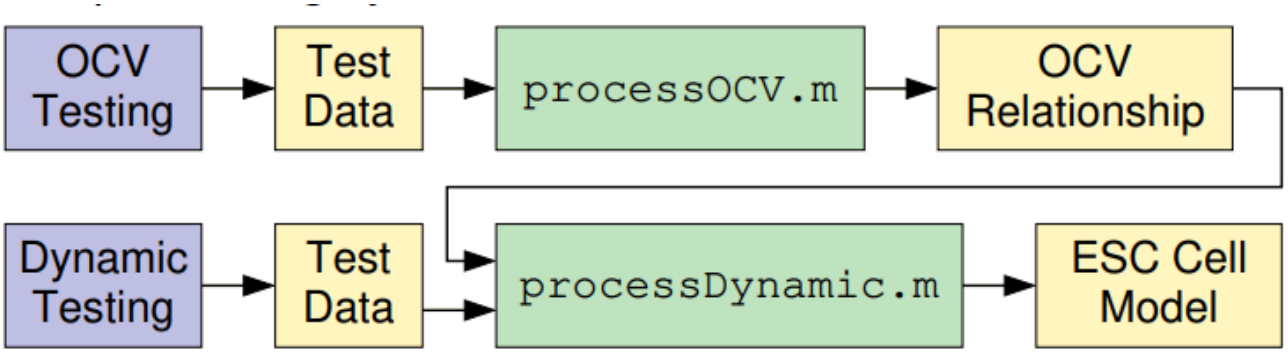
2.2.5 Introdução ao código Octave para determinar a parte estática de um ECM

Você aprendeu como realizar experimentos de laboratório para coletar os dados necessários para construir uma relação de Tensão de Circuito Aberto (OCV) versus Estado de Carga (SOC). Nesta lição, vamos explorar como escrever código Octave (compatível com MATLAB) para processar esses dados e calcular a relação OCV usando as equações que você já aprendeu.

Este processo é a primeira metade de um fluxo de trabalho maior para criar um modelo de célula completo. Como ilustrado no fluxograma, os testes de OCV no laboratório (caixa azul) produzem dados de teste (caixa amarela), que são processados pela função Octave `processOCV.m` (caixa verde) para gerar a Relação OCV final (caixa amarela). Depois, aprenderemos sobre os testes dinâmicos, que alimentarão uma função `processDynamic.m` para criar o modelo ESC final.



Um Passo a Passo pelo Código Octave `processOCV.m`

Vamos destacar as seções mais importantes do código `processOCV.m` para demonstrar como ele implementa diretamente os conceitos teóricos que discutimos.

Passo 1: Configuração Inicial e Verificação de Dados

O código começa com comentários que descrevem o procedimento de teste de laboratório assumido,

```
% ------------------------------------------------------------------------
% 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
 %
 % 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 .
```

seguido pela definição da função: `function model = processOCV(data, cellID)`.

O primeiro passo computacional é organizar os dados de temperatura e realizar uma verificação crítica: o script garante que exista um conjunto de dados de teste realizado a **25°C**. Isso é obrigatório porque os dados de 25°C servem como linha de base para os cálculos de eficiência coulômbica. Se nenhum dado de 25°C for encontrado, o script para com um erro.

```
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 25 degC ');
 end
 not25 = find ( filetemps ~= 25);
 data25 = data ( ind25 );
```

Passo 2: Cálculo da Eficiência e Capacidade de Base (η(25∘C) e Q(25∘C))

Em seguida, o código calcula a eficiência coulômbica e a capacidade total usando os dados de 25°C, conforme o método da aula anterior.

1. Ele soma os Ampere-horas (Ah) totais descarregados e carregados em todos os quatro scripts de teste.

2. Calcula a eficiência coulómbica a 25°C, η(25∘C), como a razão entre os Ah totais descarregados e os Ah totais carregados.
3. Calcula a capacidade total a 25°C, Q(25∘C), somando a carga líquida efetiva removida durante a porção de descarga do teste (scripts 1 e 2).

```
% 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 );
```

Passo 3: Compensação da Resistência em Série (R_0)

O código então implementa o método para superar o "desafio dos dados ausentes", começando pela compensação do efeito da resistência em série (R_0).

1. Ele calcula as mudanças de tensão instantâneas no início e no fim dos principais trechos de carga e descarga lenta dos scripts.
2. Ele interpola linearmente esses valores para estimar a queda de tensão $I \times R_0$ em cada ponto ao longo das curvas.
3. Ele cria novas curvas de tensão compensadas (`disV` e `chgV`) adicionando (para descarga) ou subtraindo (para carga) essa queda de tensão estimada dos dados de tensão medidos.

```
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
```

```
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
```

Passo 4: Cálculo da Curva de OCV para uma Única Temperatura

Com as curvas de tensão compensadas por R_0, o código mescla as duas para formar uma única curva de OCV aproximada para a temperatura de teste.

1. Ele calcula a diferença de tensão restante entre as curvas de carga e descarga compensadas a 50% de SOC.
2. Ele cria a curva final de OCV "bruta" (`rawocv`) que segue a curva de carga compensada em baixo SOC e a curva de descarga compensada em alto SOC, com uma transição suave na região intermediária.
3. Este processo é então repetido em um loop para todos os outros dados de temperatura de teste.

```
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;
```

Passo 5: Combinação dos Resultados para Criar o Modelo Final Dependente da Temperatura

Finalmente, o código combina todas as curvas de OCV "brutas" de cada temperatura para criar o modelo final e eficiente de duas tabelas.

1. Ele compila todas as curvas de OCV (geralmente de testes acima de 0°C para maior precisão) em uma grande matriz de dados `Vraw` (a matriz Y da aula anterior).
2. Ele constrói uma matriz `A` com uma coluna de uns e uma coluna contendo as temperaturas de teste.
3. Ele resolve o sistema de equações lineares Y=AX para a matriz desconhecida X usando o método de mínimos quadrados, que em Octave/MATLAB é feito com o comando conciso e poderoso: `X = A \ Vraw`.

A primeira linha da matriz resultante `X` é o vetor `OCV0` (a OCV a 0°C), e a segunda linha é o vetor `OCVrel` (a variação de OCV por grau Celsius).

```
% 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;
```