Introduction to `OCVfromSOCtemp`

- Principle of operation of `OCVfromSOCtemp` is straightforward
- We desire to compute

$$\text{OCV}(z,T) = \text{OCV0}(z) + T \times \text{OCVrel}(z)$$

- Can compute using two table lookups, one multiply, and one add
- However, also need to handle special cases where z might be out of range for the table (extrapolation)
- Additionally, MATLAB's built-in interpolation function is very slow
- So, we write our own

Interpolation and extrapolation

- Normal linear interpolation—first find x_k and x_{k+1} surrounding x in the table and compute

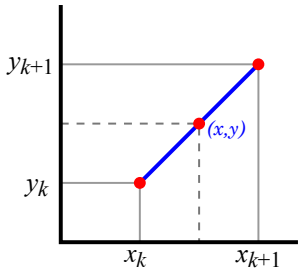
$$y = (x - x_k)\frac{y_{k+1} - y_k}{x_{k+1} - x_k} + y_k$$

- If $x < x_0$ (below all data in table)

$$y = (x - x_0)\frac{y_1 - y_0}{x_1 - x_0} + y_0$$

- If $x > x_N$ (above all data in table)

$$y = (x - x_N)\frac{y_N - y_{N-1}}{x_N - x_{N-1}} + y_N$$



Beginning of function

```
% ------------------------------------------------------------------------
% function OCVfromSOCtemp
%
% This function returns the fully rested open-circuit-voltage of a lithium-ion
% cell given its state-of-charge.
% Function inputs:
%   soc:   state-of-charge (can be scalar, vector, matrix)
%   temp:  temperature (either a scalar, or same size as soc)
%   model: model produced by processOCV and/or processDynamic
function ocv=OCVfromSOCtemp(soc,temp,model)

  % copy data into local variables (easier to use); ensure all are column vectors
  OCV0 = model.OCV0(:);  OCVrel = model.OCVrel(:);
  SOC  = model.SOC(:);   soccol = soc(:);
  if isscalar(temp),
    Tcol = temp*ones(size(soccol)); % copy scalar temperature for all socs
  else
    Tcol = temp(:); % force to be col vector
  end
```

Extrapolation for low SOCs

- To extrapolate off lower end of table, note that `dvdz` is slope between first two OCVs in table

```
% initialize output data structures plus some indexing variables
ocv=zeros(size(soccol));      % initialize the output ocv vector
diffSOC=SOC(2)-SOC(1);        % delta SOC in the model structure
I1=find(soccol <= SOC(1));    % index of input SOC values out of range (low)
I2=find(soccol >= SOC(end)); % index of input SOC values out of range (high)
I3=find(soccol > SOC(1) & soccol < SOC(end)); % index of SOC values in range
I6=isnan(soccol);             % index of input SOC values equal to NaN

% for input SOC values out of range (low), extrapolate off low end of table
if ~isempty(I1),
  dvdz = ((OCV0(2)+Tcol.*OCVrel(2)) - (OCV0(1)+Tcol.*OCVrel(1)))/diffSOC;
  ocv(I1) = (soccol(I1)-SOC(1)).*dvdz(I1) + OCV0(1)+Tcol(I1).*OCVrel(1);
end
```


Computing OCV for remaining SOCs

- Now, `dvdz` is slope between final two OCVs in table

```
% for input SOC values out of range (high), extrapolate off
if ~isempty(I2), % high end of table
  dvdz = ((OCV0(end)+Tcol.*OCVrel(end)) - ...
          (OCV0(end-1)+Tcol.*OCVrel(end-1)))/diffSOC;
  ocv(I2) = (soccol(I2)-SOC(end)).*dvdz(I2) + OCV0(end)+Tcol(I2).*OCVrel(end);
end

% for normal SOC range, manually interpolate (10x faster than "interp1")
I4=(soccol(I3)-SOC(1))/diffSOC; I5=floor(I4); % frac, int index of OCV entry
I45 = I4-I5; omI45 = 1-I45; % distance b/w frac & int; one minus distance
ocv(I3)=OCV0(I5+1).*omI45 + OCV0(I5+2).*I45; % OCV0 part first, then OCVrel
ocv(I3)=ocv(I3) + Tcol(I3).*(OCVrel(I5+1).*omI45 + OCVrel(I5+2).*I45);

% for NaN input SOCs
ocv(I6)=0; % replace NaN SOCs with zero voltage
ocv = reshape(ocv,size(soc)); % output ocv same shape as input soc
return
```


Summary

- You have now learned how `OCVfromSOCtemp.m` works
- For most cases, it simply performs linear interpolation to look up an OCV0 value and an OCVrel value, from which it computes OCV
- But, also has special cases to extrapolate beyond model data to give reasonable estimates at OCV for inputs it has never seen before
- The Octave/MATLAB code is a little tricky for fast operation

Credits

Credits for photos in this lesson

- Interpolation figure on slide 2, by ElectroKid (Own work)
  [Public domain], via Wikimedia Commons (modified from original)
  `https://commons.wikimedia.org/wiki/File:LinearInterpolation.svg`