How quickly must I balance a pack?

- Must balance at least as quickly as the pack becomes unbalanced, for long-term balancing needs
- But, how quickly is that? Battery pack simulation is an excellent tool to evaluate how quickly pack can reach imbalance
  - Select random cell characteristics typical of variation expected in real cells
  - Simulate battery pack over many repeated realistic drive scenarios
  - Gather statistics on how quickly cells become imbalanced
  - Then, simulate balancing: see how quickly can balance
- In this lesson, we begin to examine code designed to simulate packs having random cell characteristics

Modeling variability (1)

- Modeling temperature ("uniform" or "random")
  - "Uniform" temperature scenario holds all cells at 25 °C
  - "Random" temperature scenario assigns each cell a random temperature, uniformly distributed between 22.5 °C and 27.5 °C, which is then held constant during the simulation
- Modeling capacity ("standard" or "random")
  - "Standard" capacity scenario uses the nominal capacity value from the ESC cell model for the cell being simulated, for the cell's temperature
  - "Random" capacity scenario adds different uniform random numbers between -0.25 Ah and 0.25 Ah to each cell's nominal capacity, and the modified capacities are then held constant during the simulation

Modeling variability (2)

- Modeling resistance ("standard" or "random")
  - "Standard" uses resistance value R_0 from ESC cell model for cell being simulated, for cell's temperature
  - "Random" adds uniform random values between $-0.5\,\mathrm{m}\Omega$ and $1.0\,\mathrm{m}\Omega$ to R_0
  - Resistance held constant during simulation
- Modeling self discharge ("off" or "on")
  - When "on," modeled by placing high-valued resistor in parallel with every cell
    - Value is $R_{\text{sd}} = (-20 + 0.4T_{\text{sd}}) \times \text{SOC} + (35 - 0.5T_{\text{sd}})$ [kΩ], where SOC is the present time-varying cell state-of-charge, between 0 and 1
    - Variation in self-discharge is accomplished by computing $T_{\text{sd}} = T + T_{\text{random}}$, where T_{random} is uniformly distributed between -5 °C and 5 °C for each cell
    - T_{sd} is computed before simulation starts, then held constant during simulation

Modeling variability (3)

- Modeling coulombic efficiency ("ideal" or "random")
  - On discharge, is always assumed to be 1.0
  - "Ideal" case considers coulombic efficiency to be 1.0 on charge as well
  - "Random" uses value of coulombic efficiency on charge for each cell chosen to be uniformly distributed between 99.7 % and 99.9 %, held constant during simulation
- Modeling leakage current ("uniform" or "random")
  - "Uniform" places 10 mA load on every cell, models electronics powered by cell
  - "Random" places uniformly distributed load between 10 and 12 mA on each cell
  - Once leakage current determined, value kept constant during simulation

Beginning of simulator code

- First section gives help information, sets up random options
- `randOptions` fields are set to "0" for a standard simulation
  "1" for random values, as unpacked below the function header

```
% ------------------------------------------------------------------
% simRandPack: Simulate battery pack having Ns cells in series for Nc
% discharge/charge cycles, where all cells in pack can have random
% parameter values (e.g., capacity, resistance, etc.)
%
% Assumes no hysteresis in the cell model (this could be changed
% fairly easily; hysteresis makes results more difficult to interpret,
% so this assumption is okay for a first analysis, at least).
% ------------------------------------------------------------------
function packData = simRandPack(Ns,Nc,cycleFile,model,randOptions)

  tOpt = randOptions(1); qOpt = randOptions(2); rOpt = randOptions(3);
  sdOpt = randOptions(4); cOpt = randOptions(5); lOpt = randOptions(6);
  profile = load(cycleFile); % e.g., 'uddsPower.txt'
```


Reserve storage

- Code then reserves storage, initializes default states
- Standard ESC-format cell model is used by simulation

```
  % Create storage for all cell states after completion of each cycle
  packData.storez = zeros([Ns Nc]);  % create storage for final SOC
  packData.storeirc = zeros([Ns Nc]);

  % Initialize default states for ESC cell model
  maxSOC = 0.95; % cell SOC when pack is "fully charged"
  minSOC = 0.1;  % cell SOC when pack is "fully discharged"
  z   = maxSOC*ones(Ns,1); % start fully charged
  irc = zeros(Ns,1);       % at rest
  ik    = zeros([Ns 1]); % current experienced by each cell
```

Set random quantities

- Next section populates random variables for this battery pack
- Here, set up temperature, self-discharge temperature, leakage

```
% Set cell temperatures based on tOpt
if tOpt, % set to "if 1," to execute, or "if 0," to skip this code
  T = 22.5 + 5*rand([Ns 1]);
else
  T = 25*ones([Ns 1]);
end
% Set self-discharge "cell temperature"
Tsd   = T - 5 + 10*rand([Ns 1]);

% Set cell module leakage current based on lOpt
if lOpt,
  leak = 0.01 + 0.002*rand([Ns 1]);
else
  leak = 0.01*ones([Ns 1]);
end
```


Default parameter initialization

- Default initialization of cell-model parameter values
- These are overwritten later if certain random options chosen

```
% Default initialization for cells within the pack
% Note that since T has Ns elements, there is one parameter value
% per cell (even if all turn out to be identical)
q    = getParamESC('QParam',T,model);
rc   = exp(-1./abs(getParamESC('RCParam',T,model)));
r    = (getParamESC('RParam',T,model)).*(1-rc);
r0   = getParamESC('R0Param',T,model);
rt   = 2*0.000125; % 125 microOhm resistance for each tab
maxVlim = OCVfromSOCtemp(maxSOC,T,model);
minVlim = OCVfromSOCtemp(minSOC,T,model);
eta = ones([Ns 1]);
```


Overwrite default initialization

- Overwrite default init if certain random options chosen
- Specifically, random capacity, resistance, coulombic efficiency

```
% Modified initialization for cell variability
% Set individual random cell-capacity values
if qOpt, % set to "if 1," to execute, or "if 0," to skip this code
  q=q-0.25+0.5*rand([Ns 1]);      % random capacity for ea. cell
end

% Set individual random cell-resistance values
if rOpt, % set to "if 1," to execute, or "if 0," to skip this code
  r0 = r0-0.0005+0.0015*rand(Ns,1);
end
r0 = r0 + rt; % add tab resistance to cell resistance
R = sum(r0,1);

% Set individual random cell-coulombic-efficiency values
if cOpt, % set to "if 1," to execute, or "if 0," to skip this code
  eta = eta - 0.001 - 0.002*rand([Ns 1]);
end
```

Summary

- You have now learned how we might consider thinking about certain “random” quantities among different battery-pack cells
  - Temperature, capacity, resistance, self-discharge, coulombic efficiency, leakage current
- You have also started to learn about Octave code that can simulate battery packs comprising “random” cells
- Next lesson, you will learn about the remainder of the Octave code