Simulation strategy

- Last lesson, learned about beginning of `simRandPack.m`
- Simulation continues by exercising battery pack
  - Repeatedly discharge and charge pack, for predefined number of cycles
  - Discharge: Repeatedly simulate profiles of power versus time until SOC below some minimum value or voltage below some minimum value
  - Charge: Constant-power “level-2” charge until voltage reaches maximum value, then repeatedly cut charge power in half until 1/32 of original power
  - Stores state of every cell (including SOC) after every charge cycle, for later analysis

Beginning of simulation loop

- Return to examining code: simulation loop begins
- Initialize cycles, compute terminal voltages for all cells

```
% Now, simulate pack performance using ESC cell model.
theCycle = 1; theState = 'discharge';
disCnt = 0; % start at beginning of profile
fprintf('  Cycle = 1, discharging... ');
while theCycle <= Nc,
  v  = OCVfromSOCtemp(z,T,model); % get OCV for each cell
  v  = v - r.*irc;        % add in capacitor voltages
  V  = sum(v);            % Total voltage excluding I*R
  vt = v-ik.*r0;          % Cell terminal voltages
```


Discharge state

- Next portion of code simulates discharge sub-cycle
- Recall how to simulate “constant power”

```
  switch( theState )
    case 'discharge';
      % Get instantaneous demanded pack power, repeating profile
      P = profile(rem(disCnt,length(profile))+1);
      % Compute demanded pack current based on unloaded voltage
      I = V/(2*R) - sqrt(V^2/R^2 - 4*P/R)/2;
      % Default cell current = pack current
      ik = I*ones(Ns,1);
      if I < 0, % If we happen to be charging this momement
        ik = ik.*eta;
      end
      if min(z) <= minSOC || min(vt) < minVlim,   % stop discharging
        theState = 'charge'; chargeFactor = 1; ik = 0*ik;
        fprintf('charging... ');
      end
      disCnt = disCnt + 1;
```

Charge state

- Next portion of code simulates charge sub-cycle
- Recall how to simulate “constant power”

```
      case 'charge'; % start charging @ 6.6kW, then taper
        P = -6600/chargeFactor;
        I = eta*max(-min(q),V/(2*R) - sqrt(V^2/R^2 - 4*P/R)/2); % limit to 1C max
        if max(vt)>=maxVlim,
          if chargeFactor > 32, % exit after 6.6kW/32 charge
            packData.storez(:,theCycle) = z; packData.storeirc(:,theCycle) = irc;
            theState = 'discharge';
            disCnt = 0; ik = 0*ik; theCycle = theCycle + 1;
            if theCycle <= Nc,
              fprintf('\n  Cycle = %d, discharging... ',theCycle);
            end
          end
          chargeFactor = chargeFactor*2;
        end
    end % switch
```


Rest of code

- Next, add self-discharge and leakage current
- Update states, continue, return when done

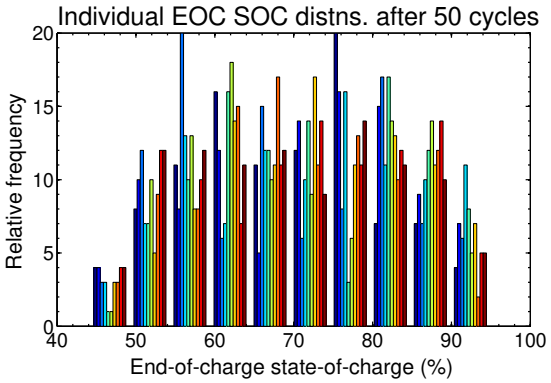
```
    % Simulate self discharge via variable resistor in parallel
    if sdOpt == 1, % compute self-discharge "resistance" and apply
      ik  = ik + vt./((-20+0.4*Tsd).*z + (35-0.5*Tsd))*1e3;
    end

    % Simulate leakage current
    ik = ik + leak;

    z = z - (1/3600)*ik./q;                % Update each cell SOC
    irc = rc.*irc + (1-rc).*ik;            % Update resistor currents
  end % end while
  fprintf('\n');
  packData.q = q; packData.rc = rc; packData.eta = eta;
  packData.r = r; packData.r0 = r0; packData.Tsd = Tsd;
  packData.T = T; packData.leak = leak;
end
```


Calling this function

- This code would generally be called by a “wrapper function” that simulates many random packs (to gather statistics)
- Any and all combination(s) of random options can be chosen, to explore sensitivity of imbalance to each effect
- `packData` structures can be stored for later analysis using balancing algorithms, to see how quickly balancing must occur
- Note that we can modify this code to add balancing logic and simulate different kinds of balancing simultaneous to the pack becoming imbalanced
- Or, can first simulate packs becoming imbalanced, then simulate balancing logic—should be same (or at least very similar)
  - Remember, pack must be balanced at least as quickly as imbalanced

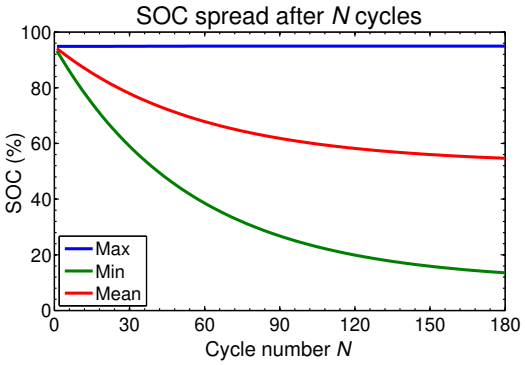
Some results of balancing simulations

- Run code for 100 random packs, each having 100 cells, for 180 dis/charge profiles for each of a number of scenarios
- Histograms of end-of-charge SOC showing data from all 100 random packs gives an idea of variation expected *without balancing* after some number of discharge/charge cycles
- Worst-case results show that imbalance can reach 50 % after only 50 days' driving, for reasonable assumed parameters for good cells



Some results of balancing simulations

- Plots of minimum/maximum/mean SOC versus cycle give an idea of how quickly SOC can diverge
- Divergence slows simply because unbalanced pack cannot be run as long during "discharge" portion of cycle: much less available energy
- 50 % divergence over 50 drive cycles implies we must be able to balance at least 1 % per drive cycle per cell



Summary

- Simulation gives us an idea of how quickly pack becomes imbalanced, and can also help pinpoint dominant causes
- For this example, we must be able to balance at least 1 % per drive cycle per cell
  - For a 7.7 Ah pack, this is at least 77 mAh, per cell, per balancing period (will want to over-design capability)
  - Passive balancing can be fast enough to keep pack in balance
  - Passive balancing is not fast enough to maximize energy/power or to extend life via quick charge transfer
- Also, if vehicle is stored for extended periods, more balancing may be necessary due to self-discharge