Introduction to setupSimVehicle.m

- There are two Octave/MATLAB scripts in the EV simulator
  - `setupSimVehicle.m` is an example of how to set up parameter values that describe vehicle and drive cycle
  - `simVehicle.m` is the code that simulates the equations we've just described
- `setupSimVehicle.m`, the topic of this lesson, defines parameter values for battery cell, module, and pack; motor, wheels, and drivetrain
- Parameter values are stored in structures, combined to make a vehicle description, and are later used to simulate the vehicle
- Example values in code roughly describe Gen 1 Chevy Volt operating in pure-electric mode, based on public information (and speculation) prior to vehicle release
  - They're probably close, but not exact or verified for this vehicle

Defining `cell`, `module`, `battery`

- The code begins by defining cell, module, battery

```
% setup simulation of vehicle - pass on to simVehicle.m
function results = setupSimVehicle
  files = {'nycc.txt','udds.txt','us06.txt','hwy.txt'};

  % Setup the Chevy Volt
  % set up cell: capacity [Ah], weight [g], (vmax, vnom, vmin) [V]
  cell = setupCell(15,450,4.2,3.8,3.0);

  % set up module: number of cells in parallel, number of cells in
  % series, overhead of module by fraction of total cells' weight
  module = setupModule(3,8,0.08,cell);

  % set up battery: number of modules in series, overhead of battery by
  % fraction of total modules' weight, (full SOC, empty SOC) [%],
  % efficiency for this module
  battery = setupPack(12,0.1,75,25,0.96,module);
```

Defining `motor`, `wheel`, `drivetrain`, `vehicle`

- It continues by defining motor, wheel, drivetrain, and vehicle

```
  % set up motor: max torque "Lmax" [Nm], (RPMrated, RPMmax)
  % [RPM], efficiency, inertia [kg/m2]
  motor = setupMotor(275,4000,12000,0.95,0.2);

  % set up wheel: radius [m], inertia [kg/m2], rollCoef
  wheel = setupWheel(0.35,8,0.0111);

  % set up drivetrain: inverter efficiency, fractional regen torque
  % limit, gear ratio, gear inertia [kg/m2], gear efficiency for this
  % battery, motor, and wheel
  drivetrain = setupDrivetrain(0.94,0.9,12,0.05,0.97,battery,motor,wheel);

  % set up vehicle: # wheels, roadForce [N], Cd, frontal area [m2],
  % weight [kg], payload [kg], overhead power [W] for this drivetrain
  vehicle = setupVehicle(4,0,0.22,1.84,1425,75,200,drivetrain);
```

Performing the simulation

- This is where the actual simulation is performed
- We investigate `simVehicle.m` in the next lesson

```
  fprintf('\n\nStarting sims...\n');
  for theCycle = 1:length(files),
    cycle = dlmread(files{theCycle},'\t',2,0);
    results = simVehicle(vehicle,cycle,0.3);
    range = (vehicle.drivetrain.battery.socFull - ...
             vehicle.drivetrain.battery.socEmpty) /...
            (vehicle.drivetrain.battery.socFull - ...
             results.batterySOC(end)) * ...
            results.distance(end);
    fprintf('Cycle = %s, range = %6.1f [km]\n',files{theCycle},range);
  end
end
```

Setting up the `cell` data structure

- Now that we've looked at the overall structure and approach of `setupSimVehicle.m`, we explore its nested setup functions
- We begin with the function that sets up the `cell` data structure

```
function cell = setupCell(capacity,weight,vmax,vnom,vmin)
  cell.capacity = capacity; % ampere hours
  cell.weight = weight; % grams
  cell.vmax = vmax; % volts
  cell.vnom = vnom; % volts
  cell.vmin = vmin; % volts
  cell.energy = vnom * capacity; % Watt-hours
  cell.specificEnergy = 1000 * cell.capacity * cell.vnom/ ...
                        cell.weight; % Wh/kg
end
```

Setting up the `module` data structure

- We continue by looking at the nested setup function that sets up the `module` data structure

```
function module = setupModule(numParallel,numSeries,overhead,cell)
  module.numParallel = numParallel;
  module.numSeries = numSeries;
  module.overhead = overhead;
  module.cell = cell;
  module.numCells = numParallel * numSeries;
  module.capacity = numParallel * cell.capacity;
  module.weight = module.numCells * cell.weight * 1/(1 - overhead)/1000; % kg
  module.energy = module.numCells * cell.energy/1000; % kWh
  module.specificEnergy = 1000 * module.energy / module.weight; % Wh/kg
end
```

Setting up the `battery` data structure

- We continue to look at `battery` nested setup function

```
function battery = setupPack(numSeries,overhead,socFull,...
                     socEmpty,efficiency,module)
  battery.numSeries = numSeries;
  battery.overhead = overhead;
  battery.module = module;
  battery.socFull = socFull;
  battery.socEmpty = socEmpty; % unitless
  battery.efficiency = efficiency; % unitless, captures I*I*R losses
  battery.numCells = module.numCells * numSeries;
  battery.weight = module.weight * numSeries * 1/(1 - overhead); % kg
  battery.energy = module.energy * numSeries; % kWh
  battery.specificEnergy = 1000 * battery.energy / battery.weight; % Wh/kg
  battery.vmax = numSeries*module.numSeries*module.cell.vmax;
  battery.vnom = numSeries*module.numSeries*module.cell.vnom;
  battery.vmin = numSeries*module.numSeries*module.cell.vmin;
end
```

Setting up the `motor`, `wheel` data structures

- We continue with `wheel` and `motor` nested setup functions

```
function wheel = setupWheel(radius,inertia,rollCoef)
  wheel.radius = radius; % m
  wheel.inertia = inertia; % km-m2
  wheel.rollCoef = rollCoef;
end

function motor = setupMotor(Lmax,RPMrated,RPMmax,efficiency,inertia)
  motor.Lmax = Lmax; % N-m
  motor.RPMrated = RPMrated;
  motor.RPMmax = RPMmax;
  motor.efficiency = efficiency;
  motor.inertia = inertia; %kg-m2
  motor.maxPower = 2*pi*Lmax*RPMrated/60000; % kW
end
```

Setting up the `drivetrain` data structure

- This is the `drivetrain` nested setup function

```
function drivetrain = setupDrivetrain(inverterEfficiency,...
      regenTorque,gearRatio,gearInertia,gearEfficiency,battery,motor,wheel)
  drivetrain.inverterEfficiency = inverterEfficiency;
  % regen torque is fraction of braking power that is used to charge
  % battery; e.g., value of 0.9 means 90% of braking power contributes
  % to charging battery; 10% lost to heat in friction brakes
  drivetrain.regenTorque = regenTorque;
  drivetrain.battery = battery;
  drivetrain.motor = motor;
  drivetrain.wheel = wheel;
  drivetrain.gearRatio = gearRatio;
  drivetrain.gearInertia = gearInertia; % kg-m2, measured on motor side
  drivetrain.gearEfficiency = gearEfficiency;
  drivetrain.efficiency = battery.efficiency * inverterEfficiency * ...
                          motor.efficiency * gearEfficiency;
end
```

Setting up the `vehicle` data structure

- Finally, the `vehicle` nested setup function

```
function vehicle = setupVehicle(wheels,roadForce,Cd,A,...
                    weight,payload,overheadPwr,drivetrain)
  vehicle.drivetrain = drivetrain;
  vehicle.wheels = wheels; % number of them
  vehicle.roadForce = roadForce; % N
  vehicle.Cd = Cd; % drag coeff
  vehicle.A = A; % frontal area, m2
  vehicle.weight = weight; % kg
  vehicle.overheadPwr = overheadPwr; % W
  vehicle.maxWeight = weight + drivetrain.battery.weight + payload;
  vehicle.rotWeight = ((drivetrain.motor.inertia + drivetrain.gearInertia) * ...
               drivetrain.gearRatio^2 + drivetrain.wheel.inertia*wheels)/...
               drivetrain.wheel.radius^2;
  vehicle.equivMass = vehicle.maxWeight + vehicle.rotWeight;
  vehicle.maxSpeed = 2*pi * drivetrain.wheel.radius * drivetrain.motor.RPMmax ...
                  * 60 / (1000 * drivetrain.gearRatio); % km/h
end
```

Summary

- In this lesson, you learned how to set up data structures in preparation for the EV simulation
- You learned about the main initialization routine, which calls nested setup functions
- You saw the main program loop, which calls `simVehicle.m`
- You learned about the nested setup functions used to define cell, module, battery, motor, wheels, drivetrain, and vehicle
- Next, we look at how to use these structures to simulate the EV