## Introduction to `simVehicle.m`

- Last lesson, you learned there are two Octave/MATLAB scripts
  - `setupSimVehicle.m` is an example of how to set up the parameter values that describe the vehicle and the drive cycle
  - `simVehicle.m` is the code that executes the equations we've just described to accomplish the simulation
- `setupSimVehicle.m` sets up structures containing definitions of battery cell, module, and pack; motor, wheels, and drivetrain
- `simVehicle.m`, the topic of this lesson, simulates a drive profile and returns a structure called `results`, which has all kinds of information in it





## simVehicle.m

- Function header plus initialization

```
% results = simVehicle(vehicle,cycle,grade)
%   - simulate vehicle defined by "vehicle", e.g., created via setupSimVehicle.m
%   - cycle is Nx2: column 1 = time (s); column 2 = desired speed (mph)
%   - grade is road grade in percent - either a constant grade for all
%     time, or a different grade value for every point in time
function results = simVehicle(vehicle,cycle,grade)
  rho = 1.225; % air density, kg/m3

  results.vehicle = vehicle;
  results.cycle = cycle; % time in s, desired speed in miles/hour
  results.time = cycle(:,1); % s
  results.grade = atan(grade/100); % convert percent to radians
  if isscalar(grade),
    results.grade = repmat(results.grade,size(results.time));
  end
  results.desSpeedKPH = cycle(:,2) * 1.609344; % convert to km/h
  results.desSpeed = min(vehicle.maxSpeed,results.desSpeedKPH*1000/3600); % m/s
```





## simVehicle.m

- Preallocate storage for results to be computed

```
  results.desAccel = zeros(size(results.desSpeed)); % m/s2
  results.desAccelForce = zeros(size(results.desSpeed)); % N
  results.aeroForce = zeros(size(results.desSpeed)); % N
  results.rollGradeForce = zeros(size(results.desSpeed)); % N
  results.demandTorque = zeros(size(results.desSpeed)); % N-m
  results.maxTorque = zeros(size(results.desSpeed)); % N-m
  results.limitRegen = zeros(size(results.desSpeed)); % N-m
  results.limitTorque = zeros(size(results.desSpeed)); % N-m
  % and so forth... also preallocate (zeroed out) storage for
  % results.motorTorque (N-m),     results.demandPower (kW)
  % results.limitPower (kW),       results.batteryDemand (kW)
  % results.current (A),           results.batterySOC (0..100)
  % results.actualAccelForce (N), results.actualAccel (m/s2)
  % results.motorSpeed (RPM),      results.actualSpeed (m/s)
  % results.actualSpeedKPH (km/h), results.distance (km)
```

## Start simulation, compute desired forces

- The simulation loop now begins, forces computed

```
prevSpeed = 0; prevMotorSpeed = 0; prevDistance = 0;
prevTime = results.time(1) - 1; prevSOC = vehicle.drivetrain.battery.socFull;
for k = 1:length(results.desSpeed),
  results.desAccel(k) = (results.desSpeed(k) - prevSpeed)/ ...
                        (results.time(k) - prevTime);
  results.desAccelForce(k) = vehicle.equivMass * results.desAccel(k);
  results.aeroForce(k) = 0.5 * rho * vehicle.Cd * vehicle.A * prevSpeed^2;
  results.rollGradeForce(k) = vehicle.maxWeight * 9.81 * ...
                              sin(results.grade(k));
  if abs(prevSpeed) > 0,
    results.rollGradeForce(k) = results.rollGradeForce(k) + ...
      vehicle.drivetrain.wheel.rollCoef * vehicle.maxWeight * 9.81;
  end
```





## Compute desired torques, limit them

- Torques are computed, and limited

```
  results.demandTorque(k) = (results.desAccelForce(k) + ...
    results.aeroForce(k) + results.rollGradeForce(k) + vehicle.roadForce) * ...
    vehicle.drivetrain.wheel.radius / vehicle.drivetrain.gearRatio;
  if prevMotorSpeed < vehicle.drivetrain.motor.RPMrated,
    results.maxTorque(k) = vehicle.drivetrain.motor.Lmax;
  else
    results.maxTorque(k) = vehicle.drivetrain.motor.Lmax * ...
      vehicle.drivetrain.motor.RPMrated / prevMotorSpeed;
  end
  results.limitRegen(k) = min(results.maxTorque(k),...
    vehicle.drivetrain.regenTorque * vehicle.drivetrain.motor.Lmax);
  results.limitTorque(k) = min(results.demandTorque(k),results.maxTorque(k));
  if results.limitTorque(k) > 0,
    results.motorTorque(k) = results.limitTorque(k);
  else
    results.motorTorque(k) = max(-results.limitRegen(k),...
                                 results.limitTorque(k));
  end
```





## Compute actual speed, distance

- Now, we compute the actual speed and distance traveled

```
  results.actualAccelForce(k) = results.limitTorque(k) * ...
    vehicle.drivetrain.gearRatio / vehicle.drivetrain.wheel.radius - ...
    results.aeroForce(k) - results.rollGradeForce(k) - vehicle.roadForce;
  results.actualAccel(k) = results.actualAccelForce(k) / vehicle.equivMass;
  results.motorSpeed(k) = min(vehicle.drivetrain.motor.RPMmax,...
    vehicle.drivetrain.gearRatio * (prevSpeed + results.actualAccel(k) * ...
    (results.time(k) - prevTime))*60 / (2*pi*vehicle.drivetrain.wheel.radius));
  results.actualSpeed(k) = results.motorSpeed(k) * ...
    2*pi*vehicle.drivetrain.wheel.radius / (60 * vehicle.drivetrain.gearRatio);
  results.actualSpeedKPH(k) = results.actualSpeed(k) * 3600/1000;
  deltadistance = (results.actualSpeed(k) + prevSpeed)/2 *...
                  (results.time(k) - prevTime)/1000;
  results.distance(k) = prevDistance + deltadistance;
```

## Computing motor and battery power demands

- Compute motor power demand, battery power demand

```
if results.limitTorque(k) > 0,
  results.demandPower(k) = results.limitTorque(k);
else
  results.demandPower(k) = max(results.limitTorque(k),-results.limitRegen(k));
end
results.demandPower(k) = results.demandPower(k) * 2*pi * ...
        (prevMotorSpeed + results.motorSpeed(k)) / 2 / 60000;
results.limitPower(k) = max(-vehicle.drivetrain.motor.maxPower,...
  min(vehicle.drivetrain.motor.maxPower,results.demandPower(k)));
results.batteryDemand(k) = vehicle.overheadPwr/1000;
if results.limitPower(k) > 0,
  results.batteryDemand(k) = results.batteryDemand(k) + ...
        results.limitPower(k)/vehicle.drivetrain.efficiency;
else
  results.batteryDemand(k) = results.batteryDemand(k) + ...
        results.limitPower(k)*vehicle.drivetrain.efficiency;
end
```





## Update battery SOC, storing results

- Update battery current, SOC, store some results

```
  results.current(k) = results.batteryDemand(k)*1000/...
                       vehicle.drivetrain.battery.vnom;
  results.batterySOC(k) = prevSOC - results.current(k) * (results.time(k) - ...
    prevTime) / (36*vehicle.drivetrain.battery.module.capacity);

  prevTime = results.time(k);
  prevSpeed = results.actualSpeed(k);
  prevMotorSpeed = results.motorSpeed(k);
  prevSOC = results.batterySOC(k);
  prevDistance = results.distance(k);
end
```





## Summary

- In this lesson, you have learned how `simVehicle.m` works
- As you have seen, it closely follows model equations developed earlier in the week
  - Desired acceleration, forces, torques are computed
  - Limited torques, forces, and acceleration computed
  - Speed, distance, battery current and SOC updated
- This brings you to the end of the content in this week!