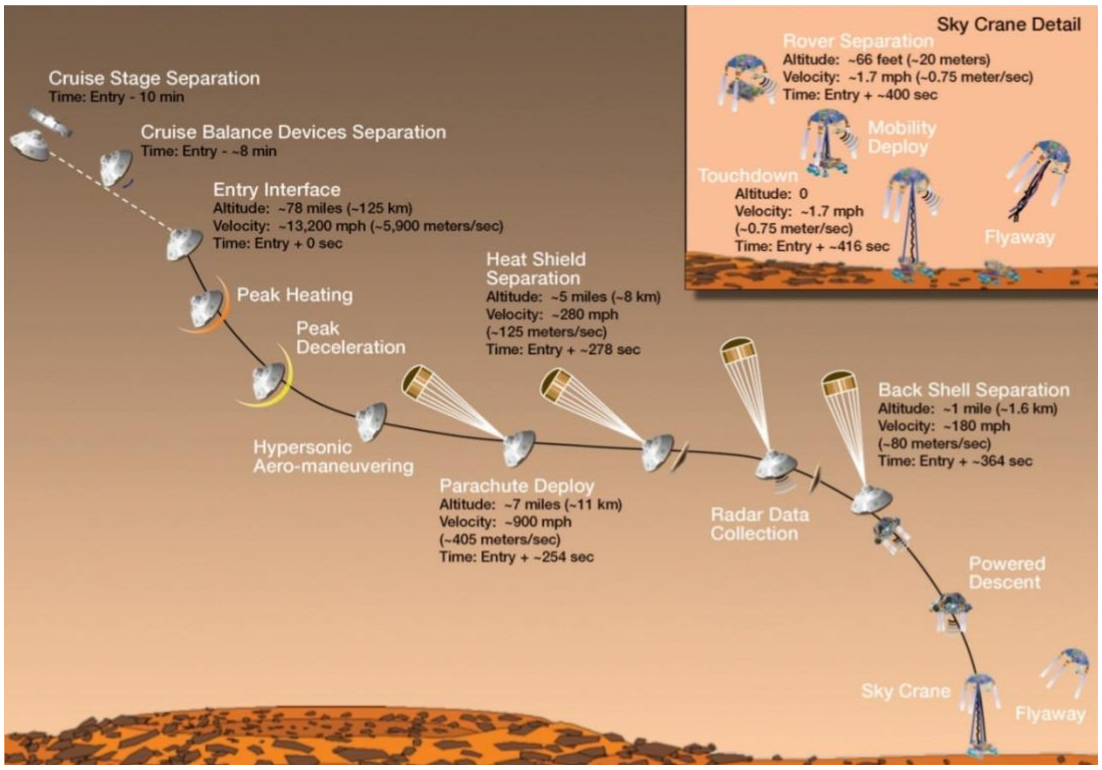
MEEN 357 DESIGN PROJECT
PHASE 2: ROVER DYNAMICAL ANALYSIS

Assigned:	Sept 24	**Submission:**	Electronic
Due Date:	October 27	**Collaboration Type:**	Groups of 2 or 3
Due Time:	11:59pm	**Grading:**	75 points

Design Project Scenario

You are a member of a small team of engineers at your firm. You and your teammates are among your firm's most promising young engineers (naturally, since you all studied at Texas A&M!). Consequently, they have tapped your team to complete several critical tasks on the firm's most important project: to design a new Mars exploration mission.

In particular the Mars mission is to land a large mobile robot safely on the surface of Mars and to navigate this robot across unknown Martian terrain to points of scientific interest. The landing phase involves four steps: (1) high-speed entry into the Martian atmosphere, (2) parachute deployment and deceleration, (3) powered descent, and (4) Sky Crane operation, in which the rover is lowered gently to the Martian surface from a hovering platform. This sequence is similar to what was performed by the Mars Science Lab (a.k.a., Mars Curiosity), illustrated below.



Once the rover is on the ground, the mission focus turns to gathering scientific data by driving the rover to interesting locations on the Martian surface. The terrain is not known precisely prior to the mission, so the rover must be capable of navigating many types of terrain safely. Your firm already has determined that the general configuration of the rover will be similar to that of the Mars Science Lab rover (see figure below). But your team still will have significant design freedom.

The project is divided into four phases that you must complete:

1. System modeling and preliminary analysis of the rover
2. Dynamical modeling and analysis of the rover
3. Simulation of landing phase
4. System optimization / Decision Making

Being fans of classic cartoons, your team has decided to name your rover Marvin, after the Loony Toons character Marvin the Martian (right).

Contents

1 PHASE 1 RECAP AND OVERVIEW OF PHASE 2

In Phase 1 of this project, your team developed several functions useful for modeling Marvin the rover and performed a preliminary analysis of rover performance. In this phase, you will conduct a more thorough analysis of Marvin. This includes performing a dynamical simulation of the rover traversing changing terrain and relaxing some of the assumptions you made in Phase 1 (such as that the motors are 100% efficient).

Phase 2 Assignment Instructions: Read this document carefully. It contains important background information and defines several tasks you and your team must complete. *Follow instructions carefully!*

2 BACKGROUND: NEW MODELS REQUIRED FOR PHASE 2

2.1 *Rover Dynamics*

Unlike the steady state situation you explored in Phase 1, the net force in a dynamic analysis is nonzero and acceleration varies over time. You require a model that can account for this behavior.

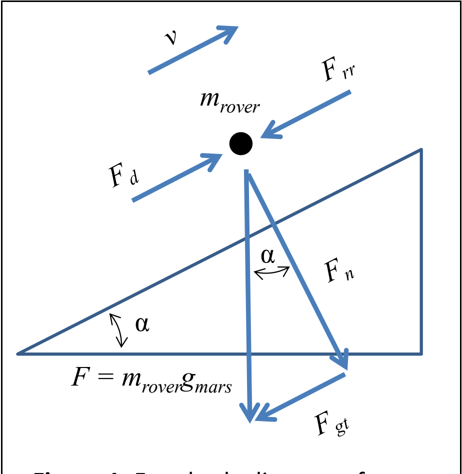


Figure 1: Free body diagram of rover on inclined terrain

To understand the dynamics of Marvin, it is helpful to construct a differential equation of motion (DEOM). The best place to start is with Newton's second law. Making the appropriate substitutions:

$$F_{net} = m_{rover}a$$

$$a = \ddot{x} = \left(\frac{1}{m_{rover}}\right)F_{net} \tag{1}$$

where $\ddot{x}$ denotes the second derivative of position with respect to time. The gravitational and rolling resistance forces are a function of the terrain angle, α, but are otherwise independent of rover dynamics. In contrast, drive force is a function of rover dynamical behavior. Recall that $F_d = 6F_w = 6\left(\frac{N_g}{r}\right)\tau_{motor}$, where F_w is the force applied by a single drive wheel, N_g is the speed reducer gear ratio, and r is the wheel radius.

2.2 *Power and Efficiency Modeling*

Without adequate power, Marvin will be unable to complete its mission. Thus, you must perform an analysis of power consumption by Marvin's drive system. Ultimately, your concern is whether the batteries contain enough energy to power the drive system through a particular mission.

2.2.1 Mechanical Energy

As a first step in analyzing the power consumption of Marvin, you can conduct a mechanical power analysis. Recall from physics that one can model the power of a rotating shaft at any point in time as $P(t) = \tau(t)\omega(t)$. It is convenient to consider this at the motor shaft. Thus,

$$P_{motor}(t) = \tau_{motor}(t)\omega_{motor}(t). \tag{2}$$

To determine the total energy output by the motor from $t = 0$ to $t = t_f$, one can take the integral of the preceding equation:

$$E_{motor} = \int_{t=0}^{t=t_f} P_{motor}(t)dt \tag{3}$$

Since you do not have an analytical expression for $P_{motor}(t)$, you will need to use numerical methods to determine E_{motor}.

Remember that the preceding analysis is for **one wheel**. There are a total of **six wheels** on Marvin.

Note: your supervisor has instructed you to assume the speed reducer is 100% efficient, which means 100% of the power from the motor goes to drive the rover. This is a reasonable assumption compared to other losses in the system. Properly constructed gearing can have mechanical power efficiencies in the 98-99% range.

2.2.2 Motor Efficiency and Required Battery Energy

The rotational mechanical energy output by the motor is useful information for analyzing the Marvin rover system. However, it is an incomplete picture of the energy needs of the drive system because it does not account for losses that occur in the motor itself.

Unlike the torque-speed and torque-power relationships for DC motors, there is no simple analytical form to the efficiency curve. Engineers generally fit a curve to samples of torque-efficiency data they obtain experimentally.

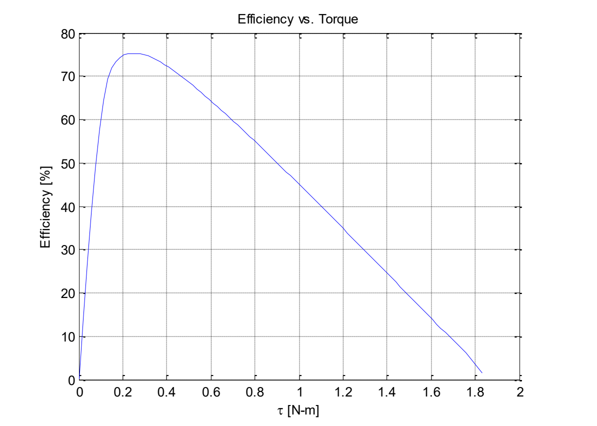


Figure 2: Efficiency curve for a typical DC motor.

Figure 2 is an example of a torque-efficiency curve for a typical DC motor. At its most efficient speed, only about 75% of the energy supplied by the batteries is converted into rotational mechanical energy. The remaining energy is lost, most typically as waste heat. Consequently, one will tend to underestimate required battery capacity when considering mechanical energy alone.

One can state the relationship between battery power consumption and motor power output as:

$$P_{motor}(t) = \eta\big(\tau(t)\big)P_{batt}(t) \tag{4}$$

where $P_{batt}(t)$ is the power consumed by the batteries at time t and $\eta\big(\tau(t)\big)$ is the efficiency at the motor operating point, $\tau(t)$.

3 EXPERIMENT SPECIFICATIONS

It is impractical, if not outright impossible, to test a system in all situations it may encounter in its use phase. What designers do instead is define a set of representative scenarios—often referred to as use cases—in which they evaluate a particular design alternative. We will take this approach with Marvin.

Since our focus is on analyzing the Marvin drive system, we will focus on its capacity to traverse varying terrain. We will define a specific use case in terms of topological information in the form of terrain angle-position pairs. For example, the data might look something like the following:

Angle (deg)	0	0	10	-5	0
Position (m)	0	20	60	80	100

One would interpret this as follows:

- Over the first 20 meters, the terrain angle holds steady at zero degrees
- Over the next 40 meters, the terrain angle changes steadily from zero to 10 degrees
- Over the next 20 meters, the terrain angle changes steadily from 10 degrees to -5 degrees
- Over the final 20 meters, the terrain angle changes steadily from -5 degrees to level.

Note that this interpretation leaves unresolved exactly what "steadily" means. It implies that one should model the regions between data points as smooth functions of position, but it does not specify exactly what this smooth change looks like. For example, one person may model this as linear segments between the points (so a continuous function with discontinuities in its derivatives) whereas another might choose to use a spline. In this project, you will be told exactly how to generalize this type of data using cubic spline interpolation in Python. For Task 2, please use the data provided in `define_experiment.py` file on Canvas – not the example data above.

In addition to defining the terrain Marvin must cross, an experiment specification also includes parameters that control the simulation. These include the coefficient of rolling resistance, initial and final times of the simulation, and the initial conditions for the simulation (i.e., initial state of Marvin). Refer to the data structures defined in the following section.

4 EXPANDED DATA STRUCTURES

In this phase you must include additional information in your rover data dictionary, as well as create a new dictionary relating the experiment scenarios you will simulate. The new and modified dictionaries are detailed below (those not listed remain unchanged from Phase 1).

rover['wheel_assembly']['motor']		
Field Name	*Type*	*Description*
`torque_stall`	scalar	Motor stall torque [N-m]
`torque_noload`	scalar	Motor no-load torque [N-m]
`speed_noload`	scalar	Motor no-load speed [rad/s]
`mass`	scalar	Motor mass [kg]
`effcy_tau`	1D numpy array	N-element array of torque data points at which efficiency data is gathered [N-m]
`effcy`	1D numpy array	N-element array of efficiency measurements corresponding to torque data [-]

Input the following data for the effcy_tau and effcy fields of the motor dictionary.

effcy_tau	0	10	20	40	70	165
effcy	0	0.55	0.75	0.71	0.50	0.05

The `experiment` dictionary is provided in **`define_experiment.py`** and contains the following fields. You can also create your own experiment dictionary for testing purposes.

experiment		
Field	*Type*	*Description*
`time_range`	1D numpy array	Two-element vector containing the start and stop time of the simulation. To be passed to an ODE solver. [s]
`initial_conditions`	1D numpy array	Two-element vector containing initial conditions for your experiment. The elements are defined as follows: `initial_conditions[0]` : rover velocity [m/s] `initial_conditions[1]` : rover position [m]
`alpha_dist`	1D numpy array	N-element vector of locations corresponding to terrain slope data [m]
`alpha_deg`	1D numpy array	N-element vector of terrain slope data for locations given in preceding vector [deg]
`Crr`	scalar	Coefficient of rolling resistance. Set this to 0.1 for this phase.

The `end_event` dictionary can be populated as shown below using the default values while you are testing your codes. For Task 8, you will use a different set of values for the fields (which are given in the description of that task).

end_event			
This structure organizes information about the conditions necessary and sufficient to terminate the solution to an ordinary differential equation using ODE solvers.			
Field Name	*Type*	*Description*	*Default/Initial Values*
`max_distance`	scalar	Maximum distance to be traverse by the rover [m]	50
`max_time`	scalar	Maximum time for the simulation by the ODE solver [s]	5000
`min_velocity`	scalar	Minimum velocity necessary to halt the solution of the differential equation. This condition is necessary in case the rover gets 'stuck'. [m/s]	0.01

In Task 8 you will simulate the rover trajectory over some experimental terrain. The data you record during this simulation will be stored in the rover['telemetry'] dictionary described below.

rover['telemetry']		
This structure contains information about the time history of the rover as it follows a trajectory on Mars surface		
Field Name	*Type*	*Description*
`Time`	1D numpy array	N-element array containing the time history of the rover [s]
`completion_time`	scalar	Time to complete a mission [s]
`velocity`	1D numpy array	N-element array containing the velocity of the rover as it follows a trajectory [m/s]
`position`	1D numpy array	N-element array containing the position of the rover as it follows a trajectory [m]
`distance_traveled`	scalar	Total distance traveled by the rover [m]
`max_velocity`	scalar	Maximum velocity of rover along a given trajectory [m/s]
`average_velocity`	scalar	Average velocity of rover along a given trajectory [m/s]
`power`	1D numpy array	N-element array containing the instantaneous power outputted by the motor along a trajectory [W]
`battery_energy`	scalar	Total energy to be extracted from the battery to complete trajectory [J]
`energy_per_distance`	scalar	Total energy spent (from battery) per meter traveled [J/m]

5 TEAM TASKS AND DELIVERABLES

The Phase 2 deliverables include Python code and results analysis geared towards simulating the trajectory of the rover and estimating the energy storage needs of the rover. The following tasks will take you through the process.

5.1 Task 1: Helper function motorW

To simplify coding and minimize the possibility of typos and other bugs, programmers often create several "helper" functions that compute commonly-needed results. You are instructed to create a helper function in Python called **`motorW`**, which computes the rotational speed of the motor given the translational velocity of the rover and a definition for the rover itself. This function should be added to your `subfunctions.py` file. See Appendix for more details.

5.2 Task 2: Visualizing the Terrain

To test the rover system, you have been provided with a terrain representation containing distance and terrain-angle pairs. This data is in the `define_experiment.py` file on Canvas. You will create a new Python script called **`experiment_visualization.py`** to graph the experimental terrain. To calculate terrain angles between data points, you will first define an interpolation function using the following line of code (note that the `interp1d` function is part of the `scipy.interpolate` library):

```
alpha_fun = interp1d(alpha_dist, alpha_deg, kind = 'cubic', fill_value='extrapolate') #
fit the cubic spline
```

This function can then be used to calculate the terrain angle at various distances along your rover's path. Your script should evaluate the terrain angle using 100 points, evenly spaced between the minimum and maximum distance (experiment['alpha_dist'] contains distance information) in the experiment file. The script should create a single figure of terrain angle vs. position with the data contained in `define_experiment.py` plotted as star symbols, and the 100 evaluated terrain angles plotted as a line. Axes should be labeled appropriately. Please include a copy of your figure and an explanation of what you are observing in your write-up. *For the explanation portion, discuss if the curve is smooth or not, why it is so, and whether the curve goes through all the known data points.*

5.3 Task 3: rover_dynamics

For this task you will implement a function called **`rover_dynamics`** that is compatible for use with an ODE solver. This function should be added to your `subfunctions.py` file. See Appendix for more details.

5.4 Task 4: mechpower

For this task you will create a Python function called **`mechpower`** that computes the instantaneous mechanical power output by a single DC motor at each point in a simulation run, in Watts. This function takes data about rover velocity during a simulation run, in m/s, and a rover dictionary as input. This function should be added to your `subfunctions.py` file. See Appendix for more details.

5.5 Task 5: Visualizing Motor Efficiency

In this task you will create a script called **`efficiency_visualization.py`** to plot motor torque vs. efficiency. The rover dictionary should contain data points for efficiency as a function of torque as noted in Section 4. To calculate efficiency values at torque values not listed in the rover dictionary, define an interpolation function using the following line of code:

```
effcy_fun = interp1d(effcy_tau, effcy, kind = 'cubic') # fit the cubic spline
```

where `effcy_tau` is the torque data and `effcy` is the efficiency data in the rover dictionary. Your script should evaluate the efficiency using 100 points, evenly spaced between the minimum and maximum motor torque given in the rover dictionary. The script should create a single figure of efficiency vs. torque with the data contained in the rover struct plotted as star symbols, and the 100 evaluated efficiencies plotted as a line. Axes should be labeled appropriately. Please include a copy of your figure and an explanation of what you are observing in your write-up. *For the explanation portion, discuss how this figure compares to Fig 2 of this handout, if the curve you obtained is smooth or not, why it is so, and whether the curve goes through all the known data points.*

5.6 Task 6: battenergy

For this task you will create a Python function called **`battenergy`** that computes the total energy consumed, in Joules, from the rover batteries over the course of a simulation run. The function takes time-velocity data pairs from a simulation run and a rover dictionary as input. This function should be added to your `subfunctions.py` file. See Appendix for more details.

5.7 Task 7: simulate_rover

For this task you will integrate a trajectory of the rover specified by the `experiment` and `end_event` dictionaries. The function should populate all subfields of the rover['telemetry'] dictionary. This function should be added to your `subfunctions.py` file. See Appendix for more details.

5.8 Task 8: Rover Simulation

For this task you will create a Python script, **`rover_experiment1.py`** to simulate the trajectory of the rover using the **`simulate_rover`** function.

- You need to load the `experiment` and `end_event` dictionaries (download `define_experiment.py` from Canvas first)
- You should also set `end_event` fields to the following values:
    - end_event['max_distance'] = 1000
    - end_event['max_time'] = 10000
    - end_event['min_velocity'] = 0.01

The script should create a single figure with three subfigures (in a 3x1 arrangement):

1. Position vs. time
2. Velocity vs. time
3. Power vs. time

For this task you should also include the figure in your write-up and explain what you observe. *The explanation must be based on what you know about the particular terrain that the rover must traverse. Please discuss: (a) if the position vs time graph is linear or non-linear and why that is the case (b) The relationship between velocity and power that you observe and the basis of that relationship (c) how the velocity profile is related to the experimental terrain from the experiment_visualization.py plot, and (d) If the velocity graph is smooth or not and why that is the case.*

Finally, please include a table of the rover['telemetry'] data for the fields: completion_time, distance_traveled, max_velocity, average_velocity, battery_energy, and batt_energy_per_distance.

5.9 Task 9: Analysis of Energy Needs

Your boss has informed you that the rover will be equipped with a 0.9072e6 [J] Lithium Iron Phosphate battery back. Can the rover complete the case defined by experiment1 (in `define_experiment.py`) with this battery pack? Please include your answer to this question and how you arrived at it in your write-up. Note that this task is a continuation of Task 8.

6 SUBMISSION PROCEDURES

To submit:	Only one final submission from each team is necessary on Gradescope. Make sure you add your team members on the final submission from the roster on Gradescope. The submission is completely electronic.
File convention:	Files must be submitted to Gradescope directly - not as part of a folder or zip/compressed file. The names of Python files are specified elsewhere in this document.

What to submit:	1) Function Files (added to subfunctions.py)
	a) motorW
	b) rover_dynamics
	c) mechpower
	d) battenergy
	e) simulate_rover
	
	2) Script Files
	a) experiment_visualization.py
	b) efficiency_visualization.py
	c) rover_experiment1.py
	
	3) Write up for Task 2, 5, 8, and 9 in a file called **Project2.pdf**.
	4) Collaboration statement, in a file called **collaboration.pdf**. Inform us if you have consulted with any other teams or generative AI tools on this project. *Include your individual contributions in this document*.
	5) All of your function files must contain comments regarding the function description (one sentence is fine) as well as function inputs and outputs that will display when "help(<fname>)" is typed in the command line.
	6) All of your Python files (functions and scripts) should be thoroughly commented, making it clear what every calculation is meant to achieve. For example, in **get_gear_ratio** from Phase 1, a useful comment for a calculation would be something like:
	`# Compute gear ratio using the pinion and gear diameters`
	`Ng = (d2/d1)**2`

7 APPENDIX: DEFINITIONS OF REQUIRED PYTHON FUNCTIONS

motorW

motorW		
General Description		
Compute the rotational speed of the motor shaft [rad/s] given the translational velocity of the rover and the rover dictionary. This function should be "vectorized" such that if given a vector of rover velocities it returns a vector the same size containing the corresponding motor speeds.		
Calling Syntax		
`w = motorW(v, rover)`		
Input Arguments		
`v`	1D numpy array OR scalar float/int	Rover translational velocity [m/s]
`rover`	dictionary	Data structure containing rover parameters
Return Arguments		
`w`	1D numpy array OR scalar float/int	Motor speed [rad/s]. Return argument should be the same size as the input argument `v`.
Additional Specifications and Notes		
▪ This function should validate (a) that the first input is a scalar or vector. If it is a vector, it must be expressed as a numpy array. Further, matrices are not allowed. and (b) that the second input is a dictionary. If any condition fails, call `raise Exception('<message here>')` with a meaningful message to the user. ▪ This function should call **get_gear_ratio** from Phase 1. ▪ Be wary of units.		

rover_dynamics

rover_dynamics		
General Description		
This function computes the derivative of the state vector (state vector is: [velocity, position]) for the rover given its current state. It requires rover and experiment dictionary input parameters. It is intended to be passed to an ODE solver.		
Calling Syntax		
`dydt = rover_dynamics(t, y, rover, planet, experiment)`		
Input Arguments		
`t`	scalar	Time sample [s]
`y`	1D numpy array	Two-element array of dependent variables (i.e., state vector). First element is rover velocity [m/s] and second element is rover position [m]
`rover`	dict	Data structure containing rover definition
`planet`	dict	Data structure containing planet definition
`experiment`	dict	Data structure containing experiment definition
Return Arguments		
`dydt`	1D numpy array	Two-element array of first derivatives of state vector. First element is rover acceleration [m/s^2] and second element is rover velocity [m/s]
Additional Specifications and Notes		

- This function should validate your inputs, and, if necessary, call `raise Exception('<message here>')` with a meaningful message to the user.
- This function should call **motorW**, **Fnet, and get_mass**.
- To calculate terrain inclination angle, you will create an interpolation function (see Section 5.2) and evaluate this function at the rover position. I.e.:

  `terrain_angle = alpha_fun(y[1])`
- This assumes you have defined `y[1]` to be the rover position. With this code in place, you can pass the scalar return value `terrain_angle` to your existing functions. This is the mechanism for specifying the terrain for a mission scenario we want to investigate.
- You may create your own experiment dictionary for testing purposes if you wish. Otherwise, use experiment1 (defined in `define_experiment.py` from Task 2).
- Be wary of units.

mechpower

mechpower		
General Description		
This function computes the instantaneous mechanical power output by a *single* DC motor at each point in a given velocity profile.		
Calling Syntax		
`P = mechpower(v,rover)`		
Input Arguments		
`v`	1D numpy array OR scalar float/int	Rover velocity data obtained from a simulation [m/s]
`rover`	dict	Data structure containing rover definition
Return Arguments		
`P`	1D numpy array OR scalar float/int	Instantaneous power output of a single motor corresponding to each element in `v` [W]. Return argument should be the same size as input `v`.
Additional Specifications and Notes		
▪ This function should validate (a) that the first input is a scalar or vector of numerical values. If it is a vector, it must be expressed as a numpy array. Further, matrices are not allowed. and (b) that the second input is a dict. If any condition fails, call `raise Exception('<message here>')` with a meaningful message to the user. ▪ This function should call **tau_dcmotor** and **motorW**. ▪ Be wary of units.		

battenergy

battenergy		
General Description		
This function computes the total electrical energy consumed from the rover battery pack over a simulation profile, defined as time-velocity pairs. This function assumes all 6 motors are driven from the same battery pack (i.e., this function accounts for energy consumed by all motors). This function accounts for the inefficiencies of transforming electrical energy to mechanical energy using a DC motor. In practice, the result given by this function will be a lower bound on energy requirements since it is undesirable to run batteries to zero capacity and other losses exist that are not modeled in this project.		
Calling Syntax		
`E = battenergy(t,v,rover)`		
Input Arguments		
`t`	1D numpy array	N-element array of time samples from a rover simulation [s]
`v`	1D numpy array	N-element array of rover velocity data from a simulation [m/s]
`rover`	dict	Data structure containing rover definition
Return Arguments		
`E`	scalar	Total electrical energy consumed from the rover battery pack over the input simulation profile. [J]
Additional Specifications and Notes		

- This function should validate (a) that the first two inputs are equal-length vectors of numerical values and (b) that the third input is a dict. If any condition fails, call `raise Exception('<message here>')` with a meaningful message to the user.
- This function should call **mechpower** and **tau_dcmotor**.
- The rover dictionary should contain data points for efficiency as a function of torque as noted in Section 4 and also in Task 5. You will need to interpolate between these data points (as you did in Task 5) to evaluate the efficiency for a given torque.
- Be wary of units.
- You may use any built-in Python functions for the numerical integration portion. You need not code this from scratch, unless you want to.

simulate_rover

simulate_rover		
General Description		
This function integrates the trajectory of a rover.		
Calling Syntax		
`rover=simulate_rover(rover,planet,experiment,end_event)`		
Input Arguments		
`rover`	dict	Data structure containing the parameters of the rover
`planet`	dict	Data structure containing the planet definition
`experiment`	dict	Data structure containing parameters of the trajectory to be followed by the rover
`end_event`	dict	Data structure containing the conditions necessary and sufficient to terminate simulation of rover dynamics
Return Arguments		
`rover`	dict	Data structure containing the parameters of the rover, including updated telemetry information.
Additional Specifications and Notes		
This function integrates the trajectory of a rover according to the terrain and initial conditions contained in experiment1 (defined in the define_experiment.py file). It uses `end_event` to define the necessary and sufficient conditions to terminate the simulation. Note that you can use the Python ODE solvers. Please determine the best solver for the particular problem. Note that you need to provide the conditions to stop the simulation using the options structure to be fed into the ode solver. You will need to download `end_of_mission_event` function from Canvas and add it to your subfunctions.py file. This function requires the end_event structure. The return argument will be used as one of the inputs to your ODE solver. The function must check validity of inputs. The function should populate the rover['telemetry'] dictionary.		