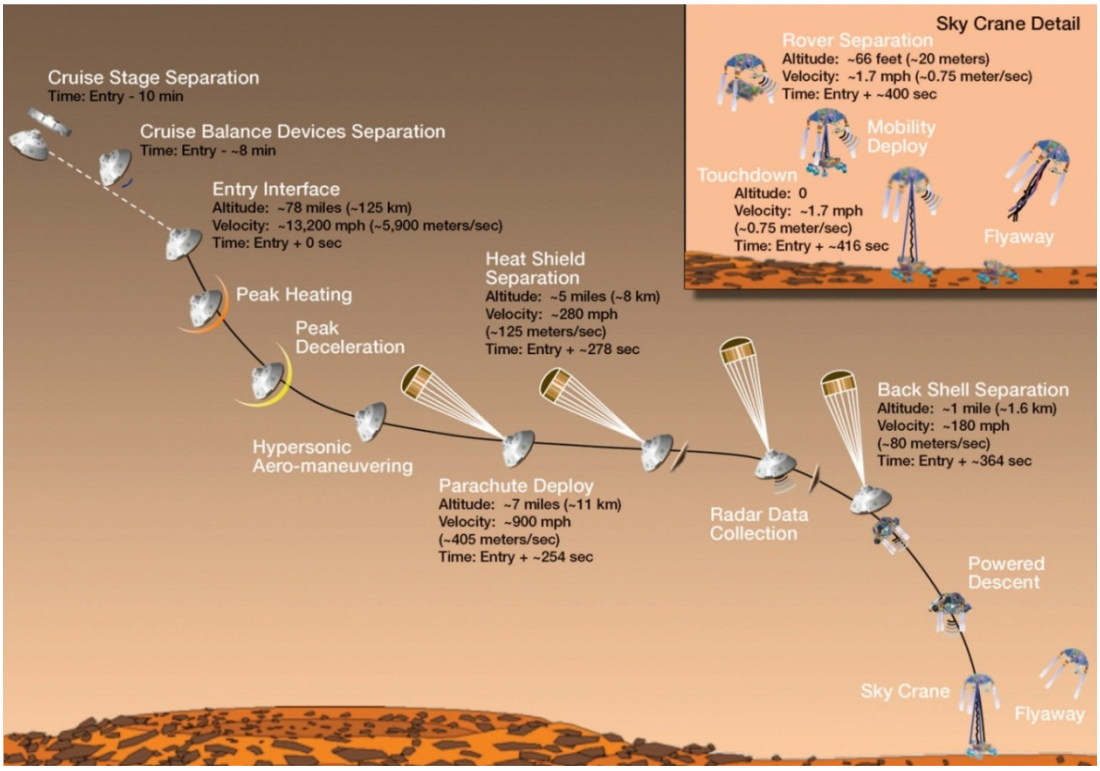
# MEEN 357 DESIGN PROJECT
# PHASE 1: SYSTEM MODELING AND PRELIMINARY ANALYSIS

| | | | |
|---|---|---|---|
| **Assigned:** | 9 September | **Submission:** | Electronic & Hardcopy |
| **Due Date:** | 5 October | **Collaboration Type:** | Groups of 2 or 3 |
| **Due Time:** | 11:59pm | **Grading:** | 75 points |

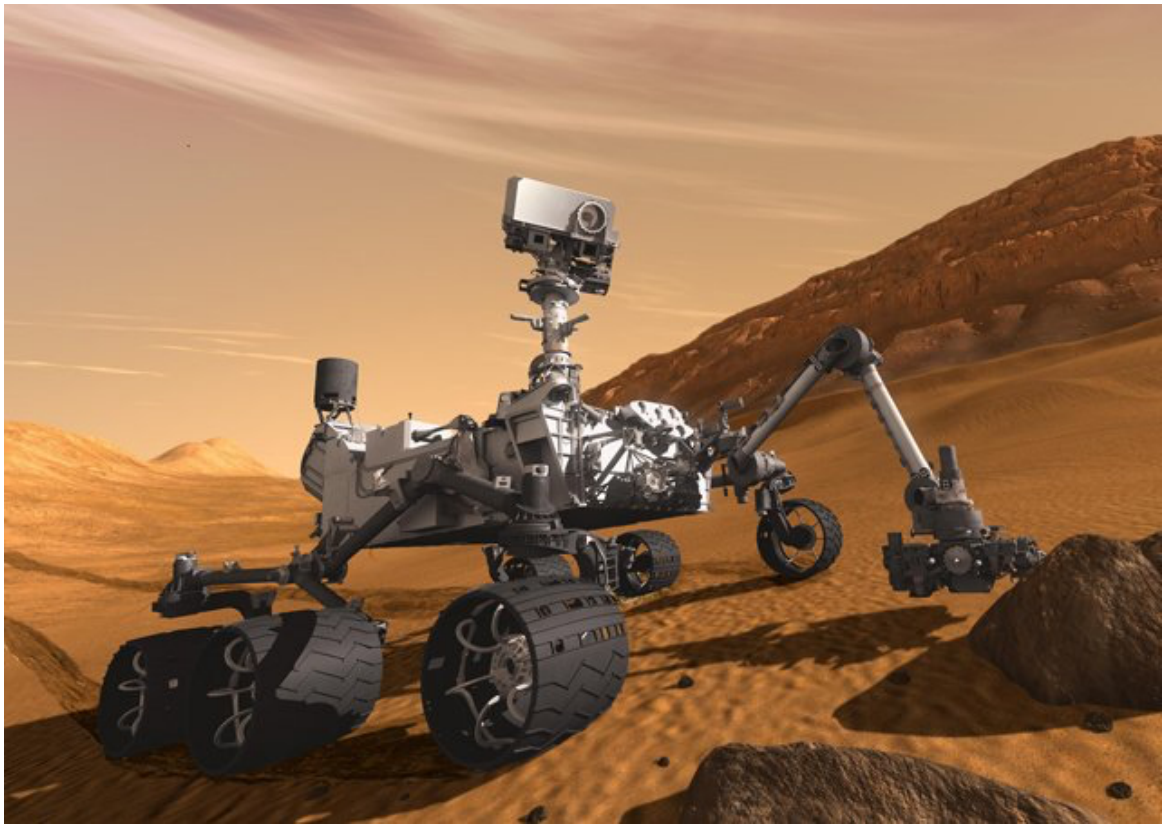
**Design Project Scenario**

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 OVERVIEW OF PHASE 1

In this phase of the project you are tasked with modeling the drive system for Marvin as well as performing some analysis of its performance characteristics. What follows is a description of the Marvin drive system, information required to analyze it, and specifications of the Phase 1 deliverables.

The motivation for Phase 1 tasks is to study the general behavior of the system as well as to develop a library of Python models that will be useful in subsequent tasks involving simulation and optimization.

**Phase 1 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: MODELING THE MARVIN ROVER DRIVE SYSTEM

Earlier in the development process, project leadership elected to go with a six-wheeled concept using a rocker-bogie suspension (http://en.wikipedia.org/wiki/Rocker-bogie) and independent drive motors in each wheel. This configuration has several advantages, including redundancy (the rover may be able to move even if one or more motors fail), mass reduction (a solution involving a single motor and a mechanical power transmission likely would be heavier), and terrain traversal capability (the rocker-bogie suspension reduces vertical travel of the center of mass and enables the rover to span gaps larger than the diameters of its wheels). Your firm has prior expertise with electric-powered robotics, so project leaders elected to use DC motors to drive the rover.

You may consider **all six** drive wheel assemblies to be identical. Each wheel assembly consists of a DC electric motor, a speed reducer, and the wheel itself. (The full wheel assembly also contains bearings, fasteners, the treads, and other hardware, but you are instructed not to consider these in your analysis.) The following is a summary of the functionality and behavior of three crucial elements of the assembly:

| Element | Functionality | Behavior (as relevant to this project) |
|---|---|---|
| **DC Motor** | Converts electrical energy into rotational mechanical energy (a torque, $\tau$, at a particular rotational speed, $\omega$). | Defined by characteristic curves that relate speed, power, and efficiency to torque output. |
| **Speed Reducer** | Converts rotational mechanical energy output from motor ($\tau_{in}, \omega_{in}$) into rotational mechanical energy at a slower speed and higher torque ($\tau_{out}, \omega_{out}$). | Defined by internal mechanics of device. |
| **Drive Wheel** | Converts rotational output of speed reducer into translational force on ground (and therefore on robot). | Defined by wheel radius. |

Figure 1 is a block diagram of the assembly. The following is a detailed description of each element, including how to model it.

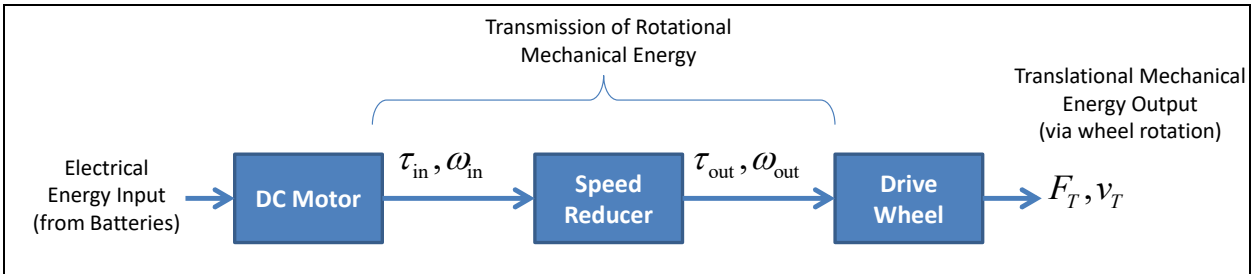



**Figure 1:** Energy flow and conversions through one side of the drive system.

## 2.1 DC Motor

A DC motor converts direct-current electrical energy to rotational mechanical energy. Although fairly detailed models of a motor are possible, you can accomplish your tasks for this project using benchmark performance curves that relate key motor characteristics to its output torque. Figure 2 contains the speed and power characteristic curves for a typical DC motor. The following are equations that approximate these curves:

**Torque vs. speed:** $\boldsymbol{\omega = \omega_{NL}\left(1 - \frac{(\tau - \tau_{NL})}{(\tau_S - \tau_{NL})}\right)}$ **or** $\tau = \tau_S - \left(\frac{\tau_S - \tau_{NL}}{\omega_{NL}}\right)\omega$

**Torque vs. power:** $P = -\left(\frac{\omega_{NL}}{\tau_S}\right)\tau^2 + \omega_{NL}\tau$

**Also recall:** $P = \tau\omega$

where:

- $\boldsymbol{\tau}$ is the motor shaft torque
- $\boldsymbol{\omega}$ is the motor shaft speed
- $\boldsymbol{P}$ is the mechanical power output of the motor
- $\tau_S$ is the stall torque
- $\tau_{NL}$ is the no-load torque
- $\omega_{NL}$ is the no-load speed
- $0 \leq \omega \leq \omega_{NL}$ [← important!]

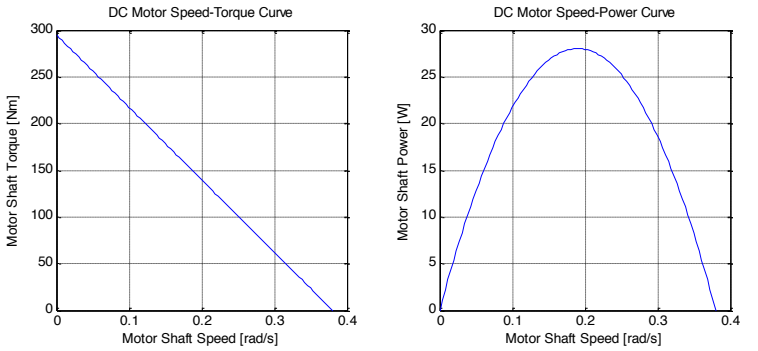



**Figure 2:** Torque-speed curve for a DC motor.

## 2.2 Speed Reducer

A speed reducer converts rotational energy at one speed and torque to rotational energy at a different (slower) speed and (higher) torque. Speed reducers usually are necessary when using motors because the output of the motor tends to be too fast / too low torque to be useful in the end application. Although other physical realizations are possible, gears are a common way to achieve a desired speed reduction.

You are considering a simple reverted gear train system such as the one depicted in Figure 3. This system consists of two identically-sized smaller gears (referred to as “pinions”) and two identically-sized larger gears. This configuration is such that the input and output shafts are co-linear. The input and output shafts spin in the same directions, but at different speeds-torques.

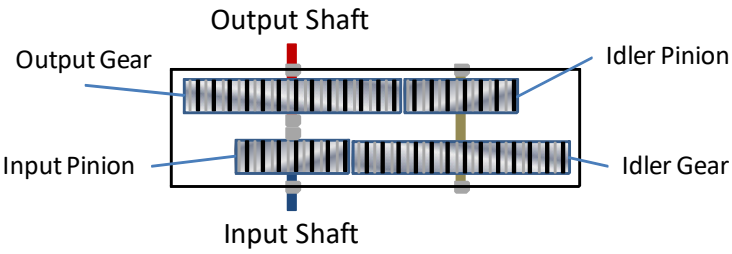



**Figure 3:** Simple Reverted Gear Train

For this phase of the project, the principal consideration relating to the speed reducer is the degree to which it reduces speed/increases torque, called its gear ratio. In future phases, you also will consider the size of your speed reducer. The critical speed reducer relationships are summarized as follows:

| | |
|---|---|
| **Overall gear ratio [-]** | $N_g = \left(\frac{N_2}{N_1}\right)^2 = \left(\frac{d_2}{d_1}\right)^2$ |
| **Input-output relationship, torque:** | $\tau_{out} = N_g \tau_{in}$ |
| **Input-output relationship, rotational speed:** | $\omega_{out} = \frac{\omega_{in}}{N_g}$ |

where:

- $\boldsymbol{d_1}$ is the diameter of the pinion
- $\boldsymbol{d_2}$ is the diameter of the larger gear
- $\boldsymbol{N_1}$ is the number of teeth on the pinion
- $\boldsymbol{N_2}$ is the number of teeth on the larger gear
- $\boldsymbol{\tau_{in}}, \boldsymbol{\tau_{out}}$ are, respectively, the torques present at the input and output shafts of the speed reducer
- $\boldsymbol{\omega_{in}}, \boldsymbol{\omega_{out}}$ are, respectively, the rotational speeds present at the input and output shafts of the speed reducer

## *2.3 Wheels*

The rover wheels are simple mechanically. The rotational mechanical output of the speed reducer acts on the axle of the wheel which, in turn, relates to the translational motion of the rover. Assuming there is no slippage of the wheel (which you are instructed to assume), you have sufficient information to calculate the force acting on the rover by an individual drive wheel and the translational speed of the rover. The key relationships are as follows:

| | |
|---|---|
| **Translational speed [m/s]** | $v_{rover} = r\omega$ |
| **Drive Force [N]** | $F_d = \frac{\tau}{r}$ |

where

- $\boldsymbol{r}$ is the radius of the wheel, in meters
- $\boldsymbol{\omega}$ is the rotational speed of the wheel, in radians per second
- $\boldsymbol{\tau}$ is the torque applied by the wheel, in Newton-meters

# 3 BACKGROUND: EXTERNAL INFLUENCES ON THE SYSTEM

External forces retard the progress of Marvin as it moves across the Martian landscape (see Figure 4). The two main forces you must consider are (1) gravitational and (2) rolling resistance. (In general, a body in motion will experience aerodynamic drag, which is a viscous retarding force. However, Marvin will travel slowly enough that this force is negligible.)

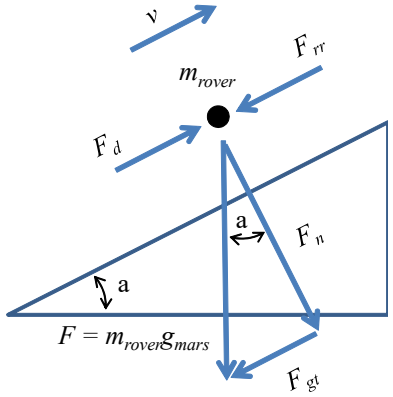



**Figure 4:** Free body diagram of rover on inclined terrain

## 3.1 Gravitational Forces

Gravitational forces may act to retard or speed up Marvin, depending on the slope of the terrain. Let $\alpha$ denote the angle of incline of the ground and $g_{mars}$ denote the acceleration due to Martian gravity (=3.72 m/s$^2$). Then the **magnitude** of gravitational force applied to Marvin is $F_{gt} = m_{rover} g_{mars} \sin\alpha$ [N] where $m_{rover}$ is Marvin's mass [kg]. The **sign** of this force is determined by your chosen sign convention for the project.

## 3.2 Rolling Resistance

Rolling resistance is a retarding force due to friction and other energy losses experienced as the treads travel over the ground. This resistance will be significant for Marvin.

A simple model for the rolling resistance force, $F_{rr}$, is the product of the rolling resistance coefficient, $C_{rr}$, and the normal force acting on the wheel, $F_n$. Thus, a constant force, $F_{rr,simple} = C_{rr} F_n$. Note that the normal force is not the same as the gravitational force that impacts translational motion directly, $F_{gt}$. For a rover on a slope of angle $\alpha$, the normal force is $F_n = m_{rover} g \cos\alpha$.

One limitation of this model is that it does not consider rover velocity. Strictly speaking, the rolling resistance force should be zero when the rover is not moving. Similarly, the sign of this force should always be opposite from the velocity (so that it always acts to reduce the magnitude of velocity). We can implement this correction in Python using something called the error function, **erf**:

$$F_{rr} = \mathrm{erf}(40 v_{rover})\, F_{rr,simple}$$

where $v_{rover}$ is the velocity of the rover [m/s]. Python already implements **erf** for you; take a look at its help file. It has a value of zero when its argument is zero and a value of 1 when the argument is greater than approximately 2.

Critically, the coefficient of rolling resistance depends on the properties of the rover wheels and the surface on which it is traveling. For instance, the value of $C_{rr}$ will be much higher on sandy soil than it is on a firmly compacted surface (such as smooth rock). Thus, we will have to design Marvin to perform well over a range of conditions.

# 4 TEAM DELIVERABLES, PART 1: PYTHON FUNCTIONS

Your team will implement several Python functions that will be useful for analyzing Marvin's capabilities. You will reuse these functions throughout the semester for several different analyses.

Table 1 is a summary of the Python functions you must implement for this phase of the project. A detailed specification of each function is given in the appendix. All of your Python functions should satisfy the following requirements.

- Your **functions** should run silently unless there is an error condition. This means you should comment out or remove any `print` statements, etc.
- Your functions should perform at least a rudimentary validation of the input arguments (e.g., to ensure that they are of the right types and of the right sizes). You can call `raise Exception()` if input validation fails.
- All functions should be saved in a single file named **subfunctions.py**

Your functions must follow **naming conventions** and any other specifications exactly.

**Table 1: Summary of functions your team must implement (detailed specs in the appendix).**

| Model | Description |
|---|---|
| **get_mass** | Computes the total mass of the rover. Uses information in the rover dict. |
| **get_gear_ratio** | Returns the speed reduction ratio for the speed reducer based on speed_reducer dict. |
| **tau_dcmotor** | Returns the motor shaft torque when given motor shaft speed and a dictionary containing important specifications for the motor. |
| **F_drive** | Returns the force applied to the rover by the drive system given information about the drive system (wheel_assembly) and the motor shaft speed. |
| **F_gravity** | Returns the magnitude of the force component acting on the rover in the direction of its translational motion due to gravity as a function of terrain inclination angle and rover properties. |
| **F_rolling** | Returns the magnitude of the force acting on the rover in the direction of its translational motion due to rolling resistances given the terrain inclination angle, rover properties, and a rolling resistance coefficient. |
| **F_net** | Returns the magnitude of net force acting on the rover in the direction of its translational motion. |

Please note that **you must follow the detailed specifications for the functions given in the appendix**. This specification defines the arguments, return values, and behavior in a very specific way. The appendix also includes definitions for the Python dicts that you will use to pass around data about your rover, its environment and specific simulation scenarios you might be investigating.

# 5 TEAM DELIVERABLES, PART 2: MARVIN ANALYSES W/ PYTHON SCRIPTS

In addition to submitting all of the code described above, you also must conduct the following analyses of the Marvin rover system. Your team must submit results from the following analyses in hard copy as well as the requested scripts.

### 5.1.1 Analysis of the DC Motor

In this analysis task, you will visualize the performance characteristics of a DC motor using the Python functions you have created. Based on these results, you will determine whether your Python functions are valid (i.e., trustable).

***graphs_motor.py***

Create a py-file script called **`graphs_motor.py`** according to the following specifications:

- It does not display anything to the console
- It plots the following three graphs **in a 3x1 array** (use the **`matplotlib.pyplot.subplot`** command to achieve this) in the following order top to bottom:
  - motor shaft speed [rad/s] vs. motor shaft torque [Nm] (use torque on the x-axis)
  - motor power [W] vs. motor shaft torque [Nm] (use torque on the x-axis)
  - motor power [W] vs. motor shaft speed [rad/s] (use speed on the x-axis)
- All graphs should have both axes labeled clearly (with units indicated). Use the **`matplotlib.pyplot.xlabel`** and **`matplotlib.pyplot.ylabel`** commands.
- Use the functions you created to generate the graphs

### 5.1.2 Impact of Speed Reducer

In this task, you will examine the impact of the speed reducer on the key rotational motion measures (power, torque, speed).

***graphs_sr.py***

Create a py-file script called **`graphs_sr.py`** that is similar to graphs_motor.py EXCEPT that it uses the speed, torque, and power ***of the speed reducer output shaft*** (the motor shaft now is the input to the speed reducer; your graphs should be of the torque, speed, and power of the speed reducer output). As in the previous case, please label your axes properly.

### 5.1.3 Maximum Attainable Rover Speed over Various Terrain Situations

In this task, you will examine the performance of the entire rover system under various physical conditions.

***analysis_terrain_slope.py***

Crete a py-file script called **analysis_terrain_slope.py** in which you use a root-finding method (e.g., bisection method, secant method, etc.) to determine the speed of the rover at various terrain slopes.

- Assume a coefficient of rolling resistance of $C_{rr} = 0.2$.
- Generate terrain angles to test with the following line of code:
  - `slope_array_deg = numpy.linspace(-10,35,25);`
  - Note that this gives you angles in DEGREES.
- Store the maximum velocity [m/s] at each angle in a vector called **`v_max`**.
- Plot `v_max` versus `slope_array_deg`. Make sure to label the axes and indicate their units.
- Do not display anything to the console
- *** Hint: Since we are not using a speed controller in this model, your rover will travel at the fastest speed it can in any given situation. Its top speed is the velocity it is at when it stops accelerating. This means you must look for the operating point of the motor at which the net force acting on the rover is zero. ***
- Hint: You can use the no-load and stall speeds of the motor to define an initial bracket for the root-finding method. Alternatively, provide an open method with any value on this range.

***analysis_rolling_resistance.py***

Create a py-file script called **analysis_rolling_resistance.py** in which you use a root-finding method (e.g., bisection method, secant method, etc.) to determine the speed of the rover at various values for the coefficient of rolling resistance. This analysis is very similar to what you must do for the terrain slope.

- Assume a terrain slope of 0 degrees (horizontal terrain)
- Generate rolling resistance coefficients to test with the following line of code:
  - `Crr_array = numpy.linspace(0.01,0.4,25);`
- Store the maximum velocity [m/s] at each angle in a vector called **`v_max`**.
- Plot `v_max` versus `Crr_array`. Make sure to label the axes and indicate their units.
- Do not display anything to the console
- The hints for the previous problem apply here as well.

***analysis_combined_terrain.py***

Crete a py-file script called **analysis_combined_terrain.py** in which you use a root-finding method (e.g., bisection method, secant method, etc.) to determine the speed of the rover at various values for the coefficient of rolling resistance AND terrain slope. You will plot the results as a surface.

*Important note: There are a few combinations of slope and rolling resistance for which no terminal speed is reached. This occurs when the slope is negative (rover traveling downhill) and the coefficient of rolling resistance is near zero (firm ground). Physically, in these cases the rover continues to accelerate as it descends the hill. Practically, your code should return NAN when it cannot find a root. Your code can continue with the NAN results in a few places. Python graphing commands will just leave blanks where the NAN points are in your data.*

Please follow the steps as outlined below in order to generate data compatible with a surface plotting function:

1. Generate rolling resistance coefficients using the following line of code:
   - `Crr_array = numpy.linspace(0.01,0.4,25);`
2. Generate an array of terrain angles using the following line of code:
   - `slope_array_deg = numpy.linspace(-10,35,25);`
3. Surface plotting functions often requires matrix inputs. Two of these matrices define values for the independent variables (coefficient of rolling resistance and slope). The other matrix contains the rover speed data. To generate the two matrices of independent variable data, use the following line of code:
   - `CRR, SLOPE = numpy.meshgrid(Crr_array, slope_array_deg)`
4. Create a matrix of zeros called **VMAX** that is the same size as the two matrices you created in the previous step. The following command will work:
   - `VMAX = numpy.zeros(numpy.shape(CRR), dtype = float)`
5. Now all the preliminaries are complete and you can analyze the rover. Create a double loop that iterates through the elements of the Crr and SLOPE matrices. It should look something like the following (you don't have to use these variable names or this exact code…this is just illustrative):

```
N = numpy.shape(CRR)[0]
for i in range(N):
    for j in range(N):
        Crr_sample = float(CRR[i,j])
        slope_sample = float(SLOPES[i,j])
        VMAX[i,j] = ... # here you put code to find the max speed at Crr_sample and
                        # slope_sample
```

6. Once you complete the double loop, you can call the surface plotting command as follows. Make sure to add axis labels and a descriptive title. Choose an appropriate view for the surface plot in your script.
    - `figure = matplotlib.pyplot.figure()`
    - `ax = Axes3D(figure, elev = N1, azim = N2) # where N1 and N2 will control the 3D view`
    - `ax.plot_surface(CRR, SLOPE, VMAX)`

As with the other analysis scripts: Do not display anything to the console.

# 6 TEAM DELIVERABLES, PART 3: QUESTIONS AND INTERPRETATION

Please submit a brief report in which you answer the following questions. Include any data and figures you need to support your answers. DO NOT assume that we will run your code to see something…if you need to refer to a figure or certain information, you must include it in your response.

## *6.1 Coding*

***Question 1:***

We had you define the acceleration due to gravity as a field in a structure that you had to pass as an input argument to several functions. Instead, we could have had you type the value for the constant, 3.72 $m/s^2$, directly in those functions. Do you believe there is an advantage to how we had you do it? Explain. Would you have done it differently? Explain why or why not.

***Question 2:***

What happens if you try to call **F_gravity** using a terrain slope of 110 degrees? Is this desirable behavior? Explain why you think this.

## *6.2 Motor and Speed Reducer Behavior*

***Question 3:***

What is the maximum power output by a single rover motor? At what motor shaft speed does this occur? Provide graphs or other data to support your answer.

***Question 4:***

What impact does the speed reducer have on the power output of the drive system? Again, provide any graphs or supporting data.

### *6.3 Rover Behavior*

***Question 5:***
Examine the graph you generated using **analysis_terrain_slope.py**. (Provide the graph in your response for reference.) Explain the trend you observe. Does it make sense physically? Why or why not? Please be precise. For example, if the graph appears linear or non-linear, can you explain why it should be the way you observed? Refer back to the rover model and how slope impacts rover behavior.

***Question 6:***
Examine the graph you generated using **analysis_rolling_resistance.py**. (Provide the graph in your response for reference.) Explain the trend you observe. Does it make sense physically? Why or why not? Please be precise. For example, if the graph appears linear or non-linear, can you explain why it should be the way you observed? Refer back to the rover model and how the coefficient of rolling resistance impacts rover behavior.

***Question 7:***
Examine the surface plot you generated using **analysis_combined_terrain.py**. (Provide the graph in your response for reference.) What does this graph tell you about the physical conditions under which it is appropriate to operate the rover? Based on what you observe, which factor, terrain slope or coefficient of rolling resistance, is the dominant consideration in how fast the rover can travel? Please explain your reasoning.

## 7 SUBMISSION PROCEDURES

| | |
|---|---|
| Before Submitting (and as soon as possible): | Someone from your team must reply to the discussion board on CANVAS with names of your team members. I will then assign an ID number for your team which you will use throughout the semester for project submissions. |
| To submit: | The Canvas submission will be by group number, so only one submission from each team is necessary. The submission is completely electronic. |
| File convention: | Please submit all your code and other files in a .zip archive. Please name this **P1_MEEN357_FA2021_TEAM**.zip**, where ** is your team number**. Please avoid having subfolders in your zip archive. The names of Python files are specified elsewhere in this document |
| What to submit: | 1) `subfunctions.py` containing **all** of the following function definitions<br>a) tau_dcmotor<br>b) get_gear_ratio<br>c) get_mass<br>d) F_rolling<br>e) F_gravity<br>f) F_drive<br>g) F_net<br><br>2) Script Files<br>a) graphs_motor.py<br>b) graphs_sr.py<br>c) analysis_terrain_slope.py<br>d) analysis_rolling_resistance.py<br>e) analysis_combined_terrain.py<br><br>3) Answers to questions, in a file called **Project1.pdf**.<br>4) Collaboration statement, in a file called **collaboration.pdf**. Inform us if you have consulted with any other teams on this project. |

# A. APPENDIX: IMPORTANT CONSTANTS

| Constant | Symbol | Value | Units | Description |
|---|---|---|---|---|
| **Wheel radius** | $r$ | 0.30 | m | |
| **Wheel mass** | | 1.0 | kg | Mass of one drive wheel |
| **Motor stall torque** | $\tau_S$ | 170 | Nm | |
| **Motor no-load torque** | $\tau_{NL}$ | 0 | Nm | |
| **Motor no-load speed** | $\omega_{NL}$ | 3.80 | rad/s | |
| **Motor mass** | | 5.0 | kg | Mass of one drive motor. |
| **Science Payload Mass** | | 75 | kg | Combined mass of all scientific instruments. |
| **RTG Mass (power subsystem)** | | 90 | kg | Radioisotope thermoelectric generator (RTG). Supplies electric power for rover. |
| **Chassis mass** | | 659 | kg | Mass of rover structure. |
| **Speed reducer pinion diameter** | $d_1$ | 0.04 | m | |
| **Speed reducer gear diameter** | $d_2$ | 0.07 | m | |
| **Speed reducer mass** | | 1.5 | kg | |
| **Acceleration due to gravity** | $g_{mars}$ | 3.72 | m/s$^2$ | |

# B. APPENDIX: DEFINITION OF DATA STRUCTURES

Oftentimes in programming, particularly when modeling a physical system, we encounter groups of variables that are related in some way. This is true when modeling your Marvin rover. For example, the DC motor component itself has several associated parameters—its no-load torque, no-load speed, stall torque. Rather than defining each of these things as individual variables, Python enables us to organize them using a construct called a **dict** (i.e., a dictionary structure).

Think of a Python dict as a container for other variables. Instead of having variables `wheel_mass` and `wheel_radius`, you can have a dict called `wheel` with "keys" called `mass` and `radius`. You can pass the entire dict to a function, and you can access the information within using these keys: e.g., for mass, you would use `rover['mass']`. Although this may seem odd to you at first, their benefits will become apparent to you as you use them. They often can save you typing (fewer arguments to functions) and they typically make the code simpler to write, thereby reducing coding mistakes. The dicts you must use for this phase of the project are defined in Table 2. They are organized according to system composition relationships. (Note: these are base definitions that we will expand upon in future project phases.)

**Table 2: Marvin dict definitions for Project Phase 1**

| **rover** | | |
|---|---|---|
| *Field Name* | *Type* | *Description* |
| wheel_assembly | dict | Wheel assembly dictionary. |
| chassis | dict | The chassis dictionary. |
| science_payload | dict | The science payload dictionary. |
| power_subsys | dict | The power subsystem dictionary. |
| **wheel_assembly** | | |
| *Field Name* | *Type* | *Description* |
| wheel | dict | The wheel dict |
| speed_reducer | dict | The speed reducer dict |
| motor | dict | The motor dict |
| **wheel** | | |
| *Field Name* | *Type* | *Description* |
| radius | scalar | Radius of drive wheel [m] |
| mass | scalar | Mass of one drive wheel [kg] |
| **speed_reducer** | | |
| *Field Name* | *Type* | *Description* |
| type | string | String of text defining the type of speed reducer. For Project Phase 1, the only valid entry is “reverted”. |
| diam_pinion | scalar | Diameter of pinion [m] |
| diam_gear | scalar | Diameter of gear [m] |
| mass | scalar | Mass of speed reducer assembly [kg] |
| **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] |
| **chassis** | | |
| *Field Name* | *Type* | *Description* |
| mass | scalar | Mass of chassis [kg] |
| **science_payload** | | |
| *Field Name* | *Type* | *Description* |
| mass | scalar | Mass of science payload [kg] |
| **power_subsys** | | |
| *Field Name* | *Type* | *Description* |
| mass | scalar | Mass of power subsystem [kg] |
| **planet** | | |
| *Field Name* | *Type* | *Description* |
| g | scalar | Acceleration due to gravity [m/s^2] |

Note that the dict **planet** is the only one we will use in this phase that is not a substructure of **rover**.

# C. APPENDIX: DEFINITIONS OF REQUIRED PYTHON FUNCTIONS

## *tau_dcmotor*

The intent of this function is to model DC motor behavior as a torque-speed curve.

| **tau_dcmotor** | | |
|---|---|---|
| ***General Description*** | | |
| This function returns the motor shaft torque in Nm given the shaft speed in rad/s and the motor specifications structure (which defines the no-load speed, no-load torque, and stall speed, among other things.<br><br>This function should operate in a "vectorized" manner, meaning that if given a vector of motor shaft speeds, it returns a vector of the same size consisting of the corresponding motor shaft torques. | | |
| ***Calling Syntax*** | | |
| `tau = tau_dcmotor(omega, motor)` | | |
| ***Input Arguments*** | | |
| `omega` | numpy array | Motor shaft speed [rad/s] |
| `motor` | dict | Data structure specifying motor parameters |
| ***Return Arguments*** | | |
| `tau` | numpy array | Torque at motor shaft [Nm]. Return argument is same size as first input argument. |
| ***Additional Specifications and Notes*** | | |

- This function should validate that (a) that the first input is a scalar or vector and (b) that the second input is a dict. If any of these conditions fail, call `raise Exception()`.
- Upon careful inspection of the DC motor model from Section 2.1, you will notice that the model is valid only for $0 \leq \omega \leq \omega_{NL}$. To determine what to do outside of this range, we must consider the physical meanings:
  - » $\omega > \omega_{NL}$ means the motor is spinning faster than it can spin on its own under no load. This implies that some other force is acting to speed the turning of the motor. Although in reality the motor might offer a little resistance to the turning force, this is minor compared to other retarding forces and losses. Thus, in this situation, set $\tau = 0$.
  - » $\omega < 0$ means the motor is turning backwards. Although it is possible to reverse the direction of a motor by reversing the direction of current flow, we will assume in our analyses that we always are trying to turn the motor in one direction. Thus, $\omega < 0$ implies there is some external torque that is greater than the torque generated by our motor. We'll model this situation as $\tau = \tau_S$, which means that our motor is applying the maximum torque it can in resistance to the external influence.

### *get_gear_ratio*

The intent of this function is to compute the gear ratio.

| **get_gear_ratio** | | |
|---|---|---|
| ***General Description*** | | |
| This function computes the gear ratio of the speed reducer.<br><br>In later project phases, you will extend this to work for various types of speed reducers. For now, it needs to work only with the simple reverted gear set described in Section 2.2. | | |
| ***Calling Syntax*** | | |
| `Ng = get_gear_ratio(speed_reducer)` | | |
| ***Input Arguments*** | | |
| `speed_reducer` | dict | Data structure specifying speed reducer parameters |
| ***Return Arguments*** | | |
| `Ng` | scalar | Speed ratio from input pinion shaft to output gear shaft. Unitless. |
| ***Additional Specifications and Notes*** | | |
| ▪ This function should validate that (a) the input is a dict. If this condition fails, call `raise Exception()`.<br><br>▪ This function should compare the string held in the `type` field to the string "reverted" using an appropriate string comparison function. Implement the comparison so that it is **not** case sensitive. For the current phase of the project, this function should return an error if the `type` field is anything other than "reverted". The error message should be informative.<br><br>▪ Although this function performs a very simple calculation, there is benefit in structuring the code this way. First, it helps you to avoid typographical errors later (once you test and validate this function, you can reuse it again and again with confidence). Second, it confines many (if not all) assumptions about the speed reducer's physical implementation to one file. This makes it much easier to introduce new speed reducer types later. | | |

### *get_mass*

The intent of this function is to sum the mass of all components of the rover and return the result.

| **get_mass** | | |
|---|---|---|
| ***General Description*** | | |
| This function computes rover mass in kilograms. It accounts for the chassis, power subsystem, science payload, and six wheel assemblies, which itself is comprised of a motor, speed reducer, and the wheel itself. | | |
| ***Calling Syntax*** | | |
| `m = get_mass(rover)` | | |
| ***Input Arguments*** | | |
| `rover` | dict | Data structure containing rover parameters |
| ***Return Arguments*** | | |
| `m` | scalar | Rover mass [kg] |
| ***Additional Specifications and Notes*** | | |
| ▪ This function should validate that (a) the input is a dict. If this condition fails, call `raise Exception()`.<br><br>▪ Be wary of units.<br><br>▪ Don't forget that there are six wheel assemblies in the rover. | | |

## *F_drive*

The intent of this function is to model the force on the rover due to the drive system.

| **F_drive** | | |
|---|---|---|
| ***General Description*** | | |
| This function computes the combined drive force, in Newtons, acting on the rover due to all six wheels. This force is a function of the motor shaft speed, and the properties of the motor, speed reducer, and drive track (all defined in the rover dict).<br><br>This function should be "vectorized" such that if given a vector of motor shaft speeds, it returns a vector of the same size consisting of the corresponding forces. | | |
| ***Calling Syntax*** | | |
| `Fd = F_drive(omega, rover)` | | |
| ***Input Arguments*** | | |
| `omega` | numpy array | Array of motor shaft speeds [rad/s] |
| `rover` | dict | Data structure specifying rover parameters |
| ***Return Arguments*** | | |
| `Fd` | numpy array | Array of drive forces [N] |
| ***Additional Specifications and Notes*** | | |
| ▪ This function should validate that (a) the first input is a scalar or vector and (b) that the second one is a dict. If any condition fails, call `raise Exception()`.<br><br>▪ Make sure to account for all six wheels.<br><br>▪ This function must call `tau_dcmotor` and `get_gear_ratio`. | | |

## *F_gravity*

The intent of this function is to model the force on the rover due to gravity.

| **F_gravity** | | |
|---|---|---|
| ***General Description*** | | |
| This function computes the component of force due to gravity, in Newtons, **acting in the direction of rover translation**. This force is a function of the angle the terrain makes with the horizon (in degrees) and the total mass of the rover (in kg).<br><br>This function should be "vectorized" such that if given a vector of terrain angles, it returns a vector of the same size consisting of the corresponding forces. | | |
| ***Calling Syntax*** | | |
| `Fgt = F_gravity(terrain_angle, rover, planet)` | | |
| ***Input Arguments*** | | |
| `terrain_angle` | numpy array | Array of terrain angles [deg] |
| `rover` | dict | Data structure containing rover parameters |
| `planet` | dict | Data structure containing planet gravity parameter |
| ***Return Arguments*** | | |
| `Fgt` | numpy array | Array of forces [N] |

| ***Additional Specifications and Notes*** |
|---|
| ▪ This function should validate that (a) the first input is a scalar or vector, (b) that all elements of the first argument are between -75 degrees and +75 degrees, and (c) that the last two inputs are dicts. If any condition fails, call `raise Exception()`. |
| ▪ Be wary of sign conventions. Positive angle (uphill) should yield negative force. |
| ▪ Be wary of units. |
| ▪ This function must call `get_mass`. |

## *F_rolling*

The intent of this function is to model the force on the rover due to rolling resistance—the retarding force resulting from the interaction between the wheels and the ground.

| **`F_rolling`** | | |
|---|---|---|
| ***General Description*** | | |
| This function computes the component of force due to rolling resistance, in Newtons, **acting in the direction of rover translation**. This force is a function of the angle the terrain makes with the horizon (in degrees), the total mass of the rover (in kg), and the rolling resistance coefficient (which is a unitless constant that depends on properties of both the wheels and the ground).<br><br>This function should be "vectorized" such that if given a vector of terrain angles, it returns a vector of the same size consisting of the corresponding forces.<br><br>This function computes the rolling resistance summed over all six wheels. Assume that 1/6$^{th}$ the rover normal force acts on each wheel. | | |
| ***Calling Syntax*** | | |
| `Frr = F_rolling(omega, terrain_angle, rover, planet, Crr)` | | |
| ***Input Arguments*** | | |
| `omega` | numpy array | Array of motor shaft speeds [rad/s] |
| `terrain_angle` | numpy array | Array of terrain angles [deg] |
| `rover` | dict | Data structure containing rover parameters |
| `planet` | dict | Data structure containing planet gravity parameter |
| `Crr` | scalar | Value of rolling resistance coefficient [-] |
| ***Return Arguments*** | | |
| `Frr` | numpy array | Array of forces [N] |
| ***Additional Specifications and Notes*** | | |
| ▪ This function should validate that (a) the first two inputs are scalars or vectors of the same size, (b) that all elements of the second argument are between -75 degrees and +75 degrees, (c) that the third and fourth inputs are dicts, and (d) that the fifth input is a positive scalar. If any condition fails, call `raise Exception()`. | | |
| ▪ Be wary of sign conventions. This force always should oppose rover motion (which we assume is in the positive direction). | | |
| ▪ Be wary of units. | | |
| ▪ This function must call `get_mass` and `get_gear_ratio`. | | |

## *F_net*

The intent of this function is to model the net force acting on the rover in the direction of its motion.

**F_net**

***General Description***

This function computes the total force, in Newtons, acting on the rover in the direction of its motion. This is a function of the motor shaft speed, terrain angle, rover properties, and the coefficient of rolling resistance.

This function should be "vectorized" such that if given same-sized vectors of motor shaft speeds and terrain angles, it returns a vector of the same size consisting of the corresponding forces.

***Calling Syntax***

```
F = F_net(omega, terrain_angle, rover, planet, Crr)
```

***Input Arguments***

| | | |
|---|---|---|
| `omega` | numpy array | Array of motor shaft speeds [rad/s] |
| `terrain_angle` | numpy array | Array of terrain angles [deg] |
| `rover` | dict | Data structure containing rover parameters |
| `planet` | dict | Data structure containing planet gravity parameter |
| `Crr` | scalar | Value of rolling resistance coefficient [-] |

***Return Arguments***

| | | |
|---|---|---|
| `Frr` | numpy array | Array of forces [N] |

***Additional Specifications and Notes***

- This function should validate that (a) the first two inputs are scalars or vectors of the same size, (b) that all elements of the second argument are between -75 degrees and +75 degrees, (c) that the third and fourth inputs are dicts, and (d) that the fifth input is a positive scalar. If any condition fails, call `raise Exception()`.
- This function must call `F_drive`, `F_gravity`, and `F_rolling`.
- Be wary of sign conventions used in your other functions.
- Be wary of units.
- Be sure to account for the forces (both driving and retarding) due to having six drive wheels.