tire_sim

November 26, 2023

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
import numpy as np
from PIL import Image, ImageDraw, ImageFont
import scienceplots
plt.style.use('science')

# Sven Nebendahl
# Load the CSV file into a DataFrame
data = pd.read_csv('quickmap-profile-data.csv') # https://quickmap.lroc.asu.edu/

# Extract x, y, and z values from the DataFrame
distance = data['position']
longitude = -data['lon']
latitude = -data['lat']
slope_table = data['TerrainSlope']

# Create a grid spherical
xi_sphere = np.linspace(min(longitude), max(longitude), 1000)
yi_sphere = np.linspace(min(latitude), max(latitude), 1000)
xi_sphere, yi_sphere = np.meshgrid(xi_sphere, yi_sphere)

# Interpolate the z values on the grid spherical
zi_sphere = griddata((longitude, latitude), slope_table, (xi_sphere,␣
 ↪yi_sphere), method='linear')

# Rover parameters
n_wheel = 4 # number of wheels
radius = 0.3 # wheel radius in m
width = 0.25 # wheel width in m
power_constant = 400/n_wheel # constant power per wheel
velocity_constant = 1 # constant velocity
c_b = 0.017 * 100 ** 2 # N/m^2 coefficient of soil/wheel cohesion
n = 1 # exponent of soil deformation and is dimensionless
k_c = 0.14 * 100 ** 2 # N/m^2 cohesive modulus of soil deformation
k_phi = 0.82 * 100 ** 3 # N/m^3 frictional modulus of soil deformation
```

```
K = 1.78 * 100 # coefficient of soil slip in m
gamma = 1500 # kg/m3 lunar soil density
phi = 35 # deg soil internal friction angle
phi_rad = np.deg2rad(phi) # rad soil internal friction angle
mu = np.tan(phi_rad) # static friction coefficient
N_c = 57.8 # Soil Bearing factors of Terzaghi (1943) for phi=35°
N_gamma = 42.4 # Soil Bearing factors of Terzaghi (1943) for phi=35°
mass = 1800/n_wheel # mass of the rover per wheel
gravity = 1.62 # gravitational constant of the moon
#slip = 0 # wheel slip ratio
slip = 0.07+(0.0215*slope_table) # wheel slip ratio per deg slope linear fit␣
 ↪based on experimental data from a paper

def rover(mass, slip, radius, power_constant, width, gravity,␣
 ↪velocity_constant, slope):
    F_weight = mass * gravity
    F_normal = F_weight * np.cos(slope)
    k = (k_c / width) + k_phi
    diameter = 2 * radius
    z_sinkage = (3 * F_normal/ ((3 - n) * (k_c + width * k_phi) * np.
 ↪sqrt(diameter))) ** (2/(2*n+1))
    theta_1 = np.arccos(1 - z_sinkage / radius)
    theta_m = (0.45 + 0.24 * slip) * theta_1
    e_lever = radius * np.sin(theta_1 - theta_m)
    l_lever = radius * np.cos(theta_1 - theta_m)
    l_contact_length = diameter/2*np.arccos(1-2*z_sinkage/diameter)
    A_vehicle_contact = width * l_contact_length

    C_rr = np.sqrt(z_sinkage / diameter) # coefficient of rolling resistance
    R_rolling = F_normal * C_rr
    R_compaction = (width * k / (n+1)) * z_sinkage ** (n + 1)

    alpha_angle_of_approach = np.arccos(1-2*z_sinkage/diameter)
    l_0 = z_sinkage*np.tan(np.pi/4-phi_rad/2)**2 # distance of rupture
    K_c = (N_c-np.tan(phi_rad))*np.cos(phi_rad)**2

    K_gamma = (2*N_gamma/np.tan(phi_rad)+1)*np.cos(phi_rad)**2
    R_bulldozing = (width*np.sin(alpha_angle_of_approach+phi_rad)/(2*np.
 ↪sin(alpha_angle_of_approach)*np.
 ↪cos(phi_rad)))*(2*z_sinkage*c_b*K_c+gamma*z_sinkage**2*K_gamma) + np.
 ↪pi*gamma*l_0**3*(np.pi-phi_rad)/540+np.pi*c_b*l_0**2/180+c_b*l_0**2*np.
 ↪tan(np.pi/4+phi_rad/2)
    R_gravitational = F_weight * np.sin(slope)

    R_sum = R_compaction + R_bulldozing + R_rolling + R_gravitational
    H_sum = (A_vehicle_contact * c_b + mu * F_normal) * (1-K/
 ↪l_contact_length*(1-np.exp(-slip*l_contact_length/K)))
```

```
    drawbar_pull = -H_sum + R_sum
    torque = drawbar_pull * l_lever + F_normal * e_lever
    velocity = power_constant * radius * 3.6 / torque
    power = (velocity_constant * torque)/(radius * 3.6)
    rpm = velocity * 60 / (2 * np.pi * radius)
    return [velocity, power, torque, drawbar_pull, rpm]

solution = rover(mass, slip, radius, power_constant, width, gravity,␣
 ↪velocity_constant, np.deg2rad(slope_table))
slip_mean = np.mean(slip)
slip_max = np.max(slip)
slope_table_mean = np.mean(slope_table)
slope_table_max = np.max(slope_table)
drawbar_pull_mean = np.mean(solution[3])
drawbar_pull_max = np.max(solution[3])
v_mean = np.mean(solution[0])
total_time = max(distance)/v_mean
v_min = np.min(solution[0])
v_max = np.max(solution[0])
torque_mean = np.mean(solution[2])
torque_min = np.min(solution[2])
torque_max = np.max(solution[2])
P_mean = np.mean(solution[1])
P_min = np.min(solution[1])
P_max = np.max(solution[1])
rpm_mean = np.mean(solution[4])
rpm_min = np.min(solution[4])
rpm_max = np.max(solution[4])
# Values for the text
labels = ['Slip Mean', 'Slope Table Mean', 'Drawbar Pull Mean', 'Mean␣
 ↪Velocity', 'Torque Mean', 'RPM Mean',
          'Slip Max', 'Slope Table Max', 'Drawbar Pull Max', 'Max Velocity',␣
 ↪'Torque Max', 'RPM Max']
values = [slip_mean * 100, slope_table_mean, drawbar_pull_mean, v_mean,␣
 ↪torque_mean, rpm_mean,
          slip_max * 100, slope_table_max, drawbar_pull_max, v_max, torque_max,␣
 ↪rpm_max]
units = ['%', '°', 'N', 'km/h', 'Nm', '1/min'] * 2  # Repeat units for each␣
 ↪parameter

# Creating an image with white background
img = Image.new('RGB', (200, 400), color='white')
draw = ImageDraw.Draw(img)

# Adding a title with LaTeX font
```

```
font = ImageFont.load_default()  # Change the font path and size as needed
draw.text((80, 10), "Design Parameters", fill='black', font=font)

text_y = 40
for label, value, unit in zip(labels, values, units):
    text = f'{label}: {value:.2f} {unit}'
    draw.text((20, text_y), text, fill='black', font=font)
    text_y += 30

# Saving the image as a PNG file
img.save('design_parameters.png')

# Displaying the image (optional)
img.show()

print("slip_mean: {:.2f} %".format(slip_mean*100))
print("slip_max: {:.2f} %".format(slip_max*100))
print("slope_table_mean: {:.2f} deg".format(slope_table_mean))
print("slope_table_max: {:.2f} deg".format(slope_table_max))
print("drawbar_pull_mean: {:.2f} N".format(drawbar_pull_mean))
print("drawbar_pull_max: {:.2f} N".format(drawbar_pull_max))
print("v_mean: {:.2f} km/h".format(v_mean))
print("total_time: {:.2f} h".format(total_time))
print("v_min: {:.2f} km/h".format(v_min))
print("v_max: {:.2f} km/h".format(v_max))
print("torque_mean: {:.2f} Nm".format(torque_mean))
print("torque_min: {:.2f} Nm".format(torque_min))
print("torque_max: {:.2f} Nm".format(torque_max))
print("P_mean: {:.2f} W".format(P_mean))
print("P_min: {:.2f} W".format(P_min))
print("P_max: {:.2f} W".format(P_max))
print("rpm_mean: {:.2f} 1/min".format(rpm_mean))
print("rpm_min: {:.2f} 1/min".format(rpm_min))
print("rpm_max: {:.2f} 1/min".format(rpm_max))


fig, (ax1) = plt.subplots(1, 1, figsize=(6, 6))

# Subplot 1: Terrain Slope Contour Plot constant power variable velocity
ax1.grid(True)
contour1 = ax1.contourf(xi_sphere, yi_sphere, zi_sphere, cmap='viridis')
fig.colorbar(contour1, ax=ax1, label='Terrain Slope [°]')
scatter1 = ax1.scatter(longitude, latitude, c=solution[0], linewidths=0.5,␣
 ↪marker='o', s=5, cmap='autumn')
fig.colorbar(scatter1, ax=ax1, label='Speed [km/h]')
ax1.axis('auto')
# Adjust x-axis and y-axis limits to add margin
```

```
margin = 0.2
ax1.set_xlim(min(longitude) - margin, max(longitude) + margin)
ax1.set_ylim(min(latitude) - margin, max(latitude) + margin)
ax1.plot(longitude[np.argmax(solution[0])], latitude[np.argmax(solution[0])],␣
 ↪'^', color='turquoise', markersize=10, label='Max Velocity')
ax1.plot(longitude[np.argmin(solution[0])], latitude[np.argmin(solution[0])],␣
 ↪'v', color='turquoise', markersize=10, label='Min Velocity')

# Add markers for every k hours
k = 20  # hours
tolerance = 0.1  # Adjust this value based on your desired tolerance
marker_added = False  # Reset the marker flag at the start

for i in range(len(distance)):
    if i > 0:
        v_mean_up_to = np.cumsum(solution[0])[i] / i
        time_passed = distance[i] / v_mean_up_to
        # Check if the current time_passed is within a new k-hour interval
        if abs((time_passed % k) / k - 1) < tolerance and not marker_added:
            label_text = '{}h'.format(round(time_passed / k) * k)
            #ax1.plot(longitude[i], latitude[i], 'o', color='white',␣
 ↪markersize=6, markeredgecolor='black')
            ax1.text(longitude[i], latitude[i], label_text, ha='center',␣
 ↪va='center', color='black', fontsize=8, alpha=1,␣
 ↪bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3',␣
 ↪alpha=1))
            marker_added = True  # Set the flag to True after adding the marker␣
 ↪and label
        elif abs((time_passed % k) / k - 1) >= tolerance:
            marker_added = False  # Reset the flag if outside the tolerance

legend = ax1.legend()
ax1.set_xlabel('Longitude [°]')
ax1.set_ylabel('Latitude [°]')
ax1.set_title('Constant Power Variable Velocity\n'
              'Mean Velocity = {:.2f} km/h\n'
              'Min Velocity = {:.2f} km/h\n'
              'Max Velocity = {:.2f} km/h\n'
              'Total time = {:.2f} h\n'
              'Constant Power = {:.2f} W\n'
              'Mass = {:.2f} kg\n'
              'Radius = {:.2f} m\n'
              'Width = {:.2f} m'
              .format(v_mean, v_min, v_max, total_time, power_constant*n_wheel,␣
 ↪mass*n_wheel, radius, width))
plt.tight_layout()
```

```
#plt.rcParams['figure.dpi'] = 600
#plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_constant_power_variable_velocity.png")
plt.show()

fig, (ax2) = plt.subplots(1, 1, figsize=(6, 6))
total_time = max(distance) / velocity_constant

# Subplot 2: Terrain Slope Contour Plot constant velocity variable power
ax2.grid(True)
contour2 = ax2.contourf(xi_sphere, yi_sphere, zi_sphere, cmap='viridis')
fig.colorbar(contour2, ax=ax2, label='Terrain Slope [°]')
scatter2 = ax2.scatter(longitude, latitude, c=solution[1]*n_wheel, linewidths=0.
 ↪5, marker='o', s=5, cmap='spring')
fig.colorbar(scatter2, ax=ax2, label='Power [W]')
ax2.axis('auto')
ax2.set_xlim(min(longitude) - margin, max(longitude) + margin)
ax2.set_ylim(min(latitude) - margin, max(latitude) + margin)
ax2.plot(longitude[np.argmax(solution[1])], latitude[np.argmax(solution[1])],␣
 ↪'^', color='turquoise', markersize=10, label='Max Power')
ax2.plot(longitude[np.argmin(solution[1])], latitude[np.argmin(solution[1])],␣
 ↪'v', color='turquoise', markersize=10, label='Min Power')

for i in range(len(distance)):
    if i > 0:
        time_passed = distance[i] / velocity_constant
        # Check if the current time_passed is within a new k-hour interval
        if abs((time_passed % k) / k - 1) < tolerance and not marker_added:
            label_text = '{}h'.format(round(time_passed / k) * k)
            #ax2.plot(longitude[i], latitude[i], 'o', color='white',␣
 ↪markersize=6, markeredgecolor='black')
            ax2.text(longitude[i], latitude[i], label_text, ha='center',␣
 ↪va='center', color='black', fontsize=8, alpha=1,␣
 ↪bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3',␣
 ↪alpha=1))
            marker_added = True  # Set the flag to True after adding the marker␣
 ↪and label
        elif abs((time_passed % k) / k - 1) >= tolerance:
            marker_added = False  # Reset the flag if outside the tolerance

legend = ax2.legend()
ax2.set_xlabel('Longitude [°]')
ax2.set_ylabel('Latitude [°]')
ax2.set_title('Constant Velocity Variable Power\n'
              'Constant Velocity = {:.2f} km/h\n'
              'Total time = {:.2f} h\n'
```

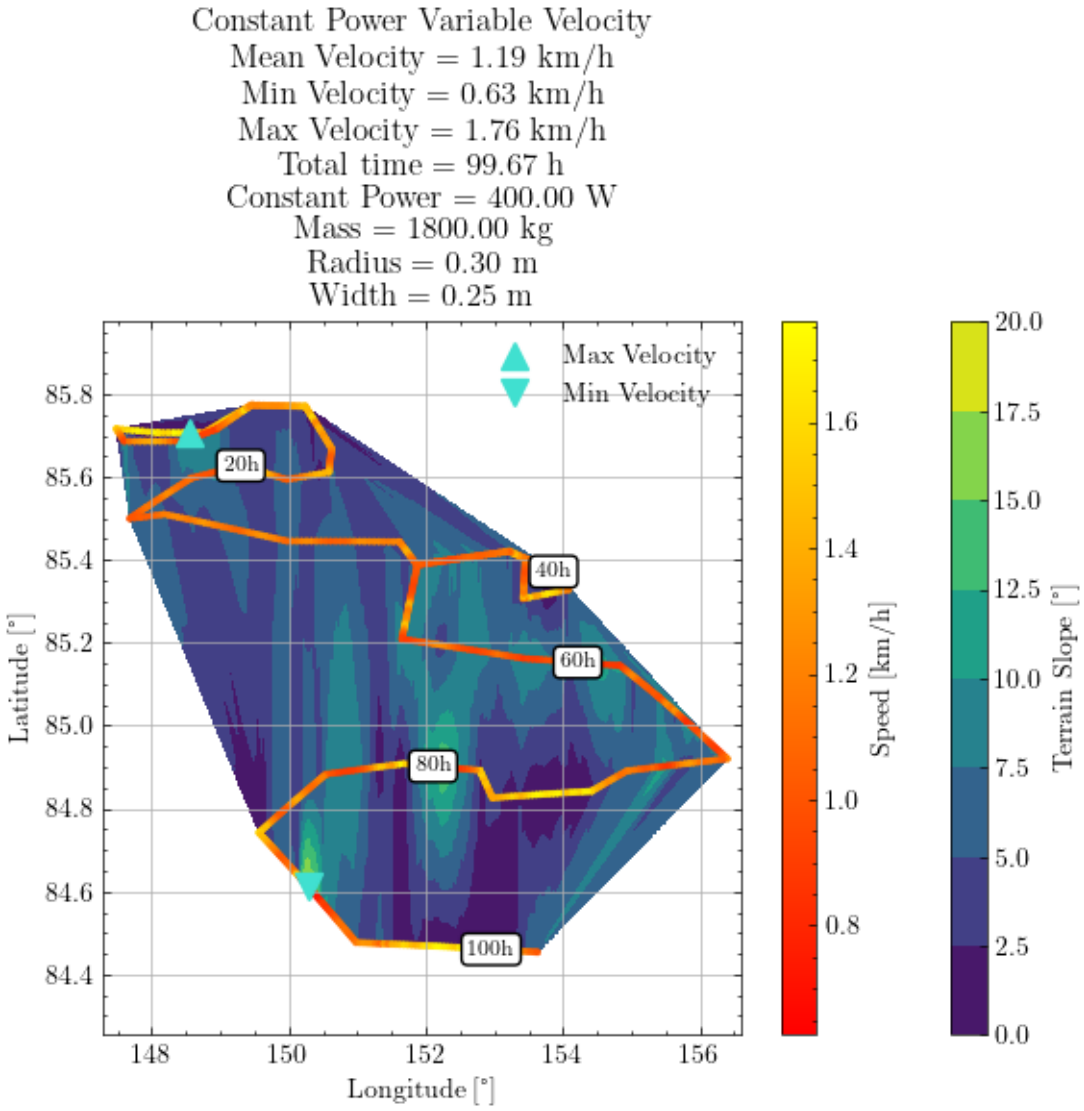
```
                'Mean Power = {:.2f} W\n'
                'Min Power = {:.2f} W\n'
                'Max Power = {:.2f} W\n'
                'Mass = {:.2f} kg\n'
                'Radius = {:.2f} m\n'
                'Width = {:.2f} m'
                .format(velocity_constant, total_time, P_mean*n_wheel,␣
 ↪P_min*n_wheel, P_max*n_wheel, mass*n_wheel, radius, width))
plt.tight_layout()
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_constant_velocity_variable_power.png")
plt.show()

fig, (ax3) = plt.subplots(1, 1, figsize=(6, 6))

# Subplot 3: Terrain Slope Contour Plot constant power variable torque
ax3.grid(True)
contour1 = ax3.contourf(xi_sphere, yi_sphere, zi_sphere, cmap='viridis')
fig.colorbar(contour1, ax=ax3, label='Terrain Slope [°]')
scatter1 = ax3.scatter(longitude, latitude, c=solution[2], linewidths=0.5,␣
 ↪marker='o', s=5, cmap='autumn')
fig.colorbar(scatter1, ax=ax3, label='Torque [Nm]')
ax3.axis('auto')
# Adjust x-axis and y-axis limits to add margin
margin = 0.2
ax3.set_xlim(min(longitude) - margin, max(longitude) + margin)
ax3.set_ylim(min(latitude) - margin, max(latitude) + margin)
ax3.plot(longitude[np.argmax(solution[2])], latitude[np.argmax(solution[2])],␣
 ↪'^', color='turquoise', markersize=10, label='Max Torque')
ax3.plot(longitude[np.argmin(solution[2])], latitude[np.argmin(solution[2])],␣
 ↪'v', color='turquoise', markersize=10, label='Min Torque')

# Add markers for every k hours
k = 20  # hours
tolerance = 0.1  # Adjust this value based on your desired tolerance
marker_added = False  # Reset the marker flag at the start

for i in range(len(distance)):
    if i > 0:
        v_mean_up_to = np.cumsum(solution[0])[i] / i
        time_passed = distance[i] / v_mean_up_to
        # Check if the current time_passed is within a new k-hour interval
        if abs((time_passed % k) / k - 1) < tolerance and not marker_added:
            label_text = '{}h'.format(round(time_passed / k) * k)
            #ax1.plot(longitude[i], latitude[i], 'o', color='white',␣
 ↪markersize=6, markeredgecolor='black')
```

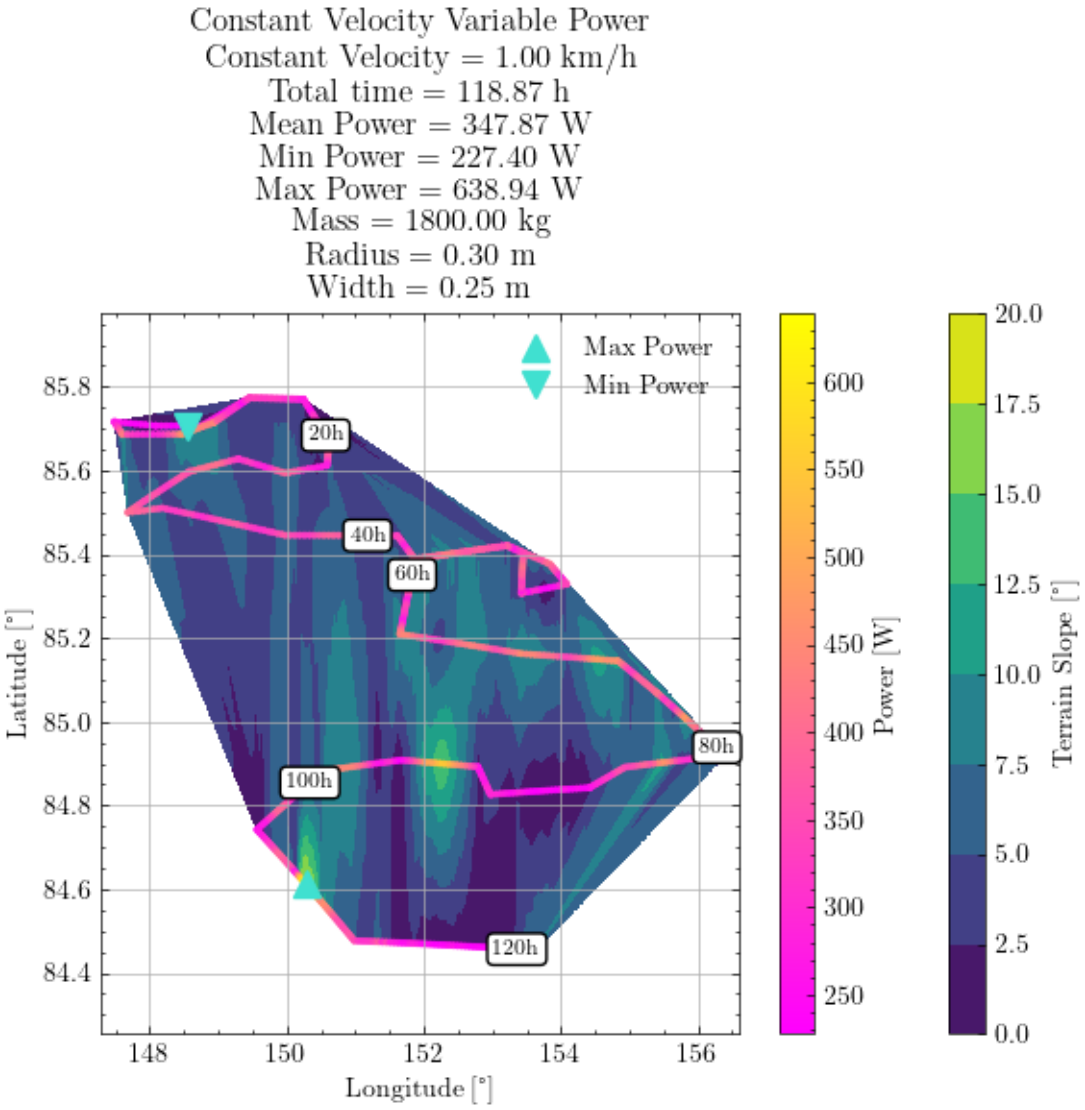
```
            ax3.text(longitude[i], latitude[i], label_text, ha='center',␣
 ↪va='center', color='black', fontsize=8, alpha=1,␣
 ↪bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3',␣
 ↪alpha=1))
            marker_added = True  # Set the flag to True after adding the marker␣
 ↪and label
        elif abs((time_passed % k) / k - 1) >= tolerance:
            marker_added = False  # Reset the flag if outside the tolerance

legend = ax3.legend()
ax3.set_xlabel('Longitude [°]')
ax3.set_ylabel('Latitude [°]')
ax3.set_title('Constant Power Variable Torque\n'
              'Mean Torque = {:.2f} Nm\n'
              'Min Torque = {:.2f} Nm\n'
              'Max Torque = {:.2f} Nm\n'
              'Total time = {:.2f} h\n'
              'Constant Power = {:.2f} W\n'
              'Mass = {:.2f} kg\n'
              'Radius = {:.2f} m\n'
              'Width = {:.2f} m'
              .format(torque_mean, torque_min, torque_max, total_time,␣
 ↪power_constant*n_wheel, mass*n_wheel, radius, width))
plt.tight_layout()
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_constant_power_variable_torque.png")
plt.show()

fig, (ax4) = plt.subplots(1, 1, figsize=(6, 6))

# Subplot 4: Terrain Slope Contour Plot constant power variable RPM
ax4.grid(True)
contour1 = ax4.contourf(xi_sphere, yi_sphere, zi_sphere, cmap='viridis')
fig.colorbar(contour1, ax=ax4, label='Terrain Slope [°]')
scatter1 = ax4.scatter(longitude, latitude, c=solution[4], linewidths=0.5,␣
 ↪marker='o', s=5, cmap='autumn')
fig.colorbar(scatter1, ax=ax4, label='RPM [1/min]')
ax4.axis('auto')
# Adjust x-axis and y-axis limits to add margin
margin = 0.2
ax4.set_xlim(min(longitude) - margin, max(longitude) + margin)
ax4.set_ylim(min(latitude) - margin, max(latitude) + margin)
ax4.plot(longitude[np.argmax(solution[2])], latitude[np.argmax(solution[2])],␣
 ↪'^', color='turquoise', markersize=10, label='Max RPM')
ax4.plot(longitude[np.argmin(solution[2])], latitude[np.argmin(solution[2])],␣
 ↪'v', color='turquoise', markersize=10, label='Min RPM')
```

```
# Add markers for every k hours
k = 20  # hours
tolerance = 0.1  # Adjust this value based on your desired tolerance
marker_added = False  # Reset the marker flag at the start

for i in range(len(distance)):
    if i > 0:
        v_mean_up_to = np.cumsum(solution[0])[i] / i
        time_passed = distance[i] / v_mean_up_to
        # Check if the current time_passed is within a new k-hour interval
        if abs((time_passed % k) / k - 1) < tolerance and not marker_added:
            label_text = '{}h'.format(round(time_passed / k) * k)
            #ax1.plot(longitude[i], latitude[i], 'o', color='white',␣
 ↪markersize=6, markeredgecolor='black')
            ax4.text(longitude[i], latitude[i], label_text, ha='center',␣
 ↪va='center', color='black', fontsize=8, alpha=1,␣
 ↪bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3',␣
 ↪alpha=1))
            marker_added = True  # Set the flag to True after adding the marker␣
 ↪and label
        elif abs((time_passed % k) / k - 1) >= tolerance:
            marker_added = False  # Reset the flag if outside the tolerance

legend = ax4.legend()
ax4.set_xlabel('Longitude [°]')
ax4.set_ylabel('Latitude [°]')
ax4.set_title('Constant Power Variable RPM\n'
              'Mean RPM = {:.2f} 1/min\n'
              'Min RPM = {:.2f} 1/min\n'
              'Max RPM = {:.2f} 1/min\n'
              'Total time = {:.2f} h\n'
              'Constant Power = {:.2f} W\n'
              'Mass = {:.2f} kg\n'
              'Radius = {:.2f} m\n'
              'Width = {:.2f} m'
              .format(rpm_mean, rpm_min, rpm_max, total_time,␣
 ↪power_constant*n_wheel, mass*n_wheel, radius, width))
plt.tight_layout()
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_constant_power_variable_rpm.png")
plt.show()
```

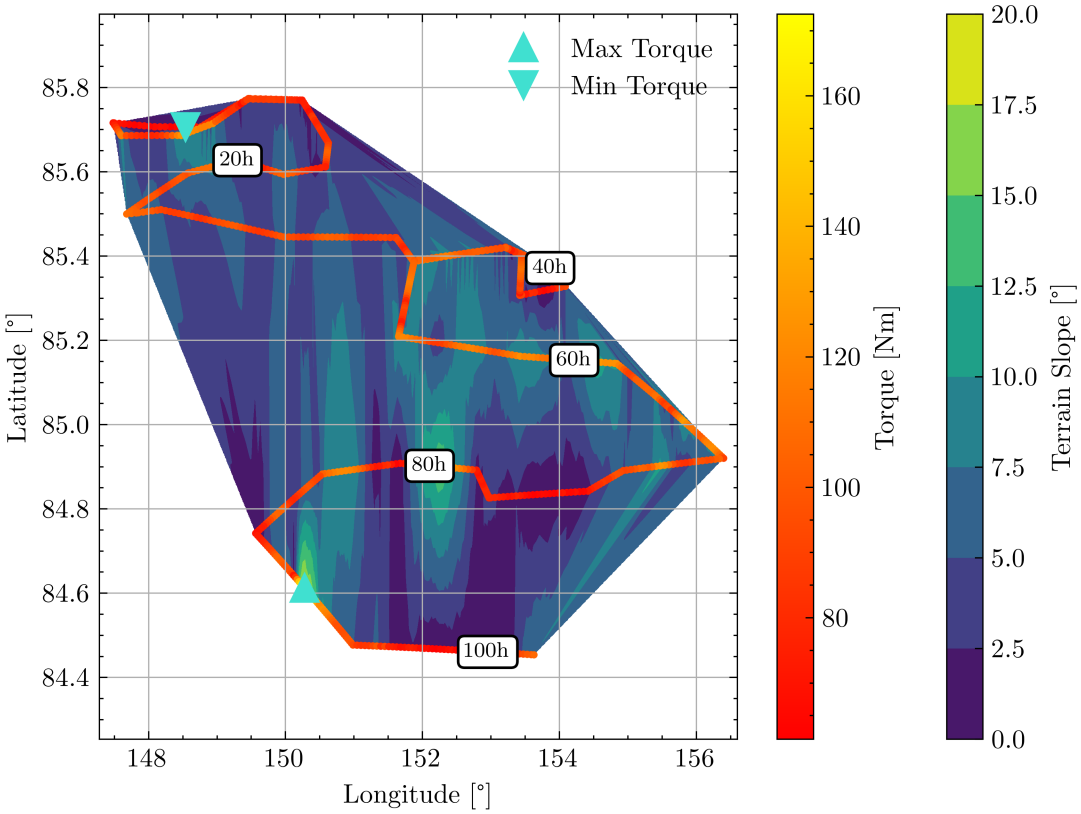
```
slip_mean: 18.39 %
slip_max: 47.06 %
slope_table_mean: 5.30 deg
```

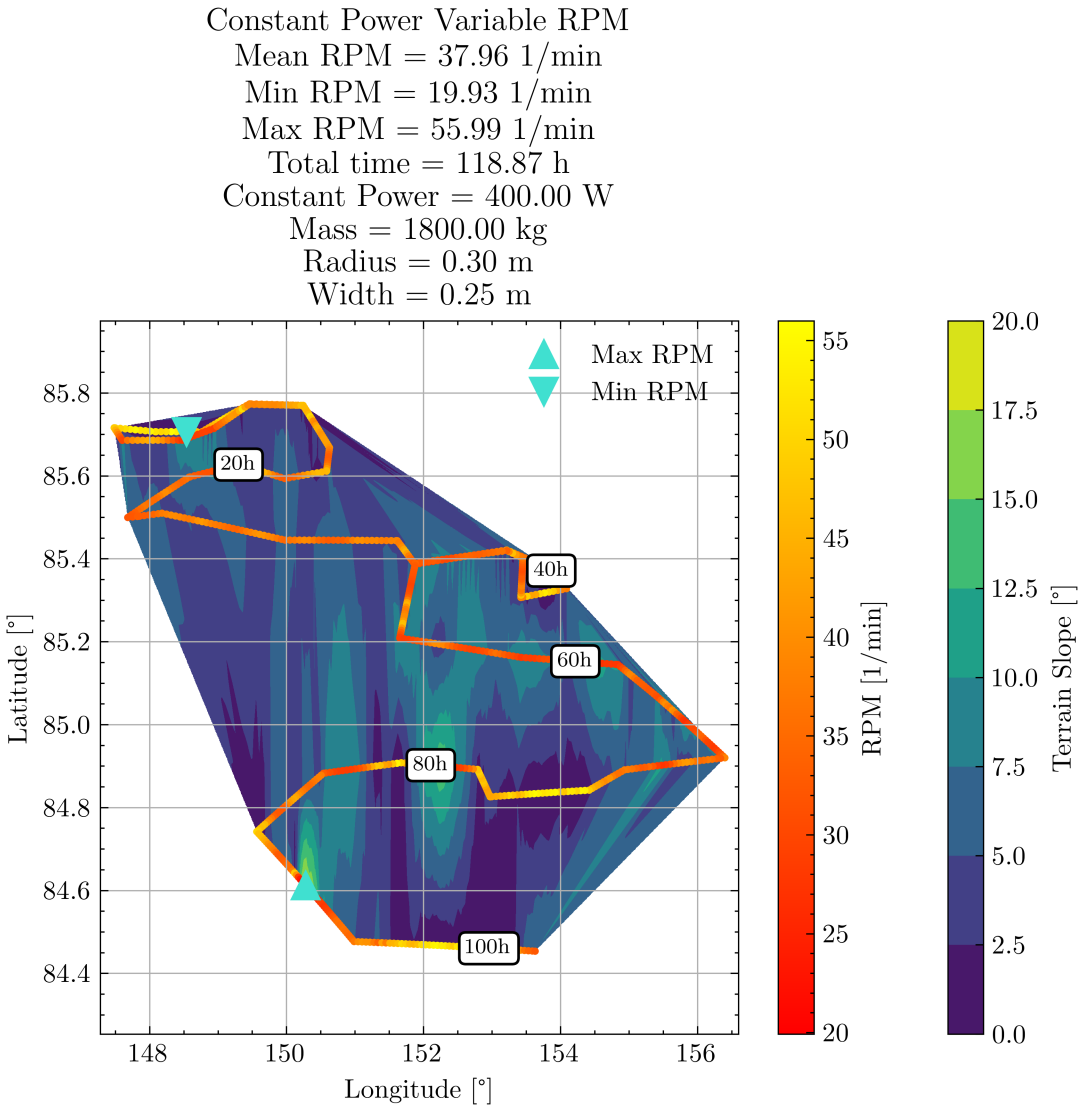
```
slope_table_max: 18.63 deg
drawbar_pull_mean: 137.84 N
drawbar_pull_max: 439.24 N
v_mean: 1.19 km/h
total_time: 99.67 h
v_min: 0.63 km/h
v_max: 1.76 km/h
torque_mean: 93.92 Nm
torque_min: 61.40 Nm
torque_max: 172.51 Nm
P_mean: 86.97 W
P_min: 56.85 W
P_max: 159.74 W
rpm_mean: 37.96 1/min
rpm_min: 19.93 1/min
rpm_max: 55.99 1/min
```




Constant Velocity Variable Power
Constant Velocity = 1.00 km/h
Total time = 118.87 h
Mean Power = 347.87 W
Min Power = 227.40 W
Max Power = 638.94 W
Mass = 1800.00 kg
Radius = 0.30 m
Width = 0.25 m
Max Power
Min Power
20h
40h
60h
80h
100h
120h
Latitude [°]
Longitude [°]
Power [W]
Terrain Slope [°]

Constant Power Variable Torque
Mean Torque = 93.92 Nm
Min Torque = 61.40 Nm
Max Torque = 172.51 Nm
Total time = 118.87 h
Constant Power = 400.00 W
Mass = 1800.00 kg
Radius = 0.30 m
Width = 0.25 m



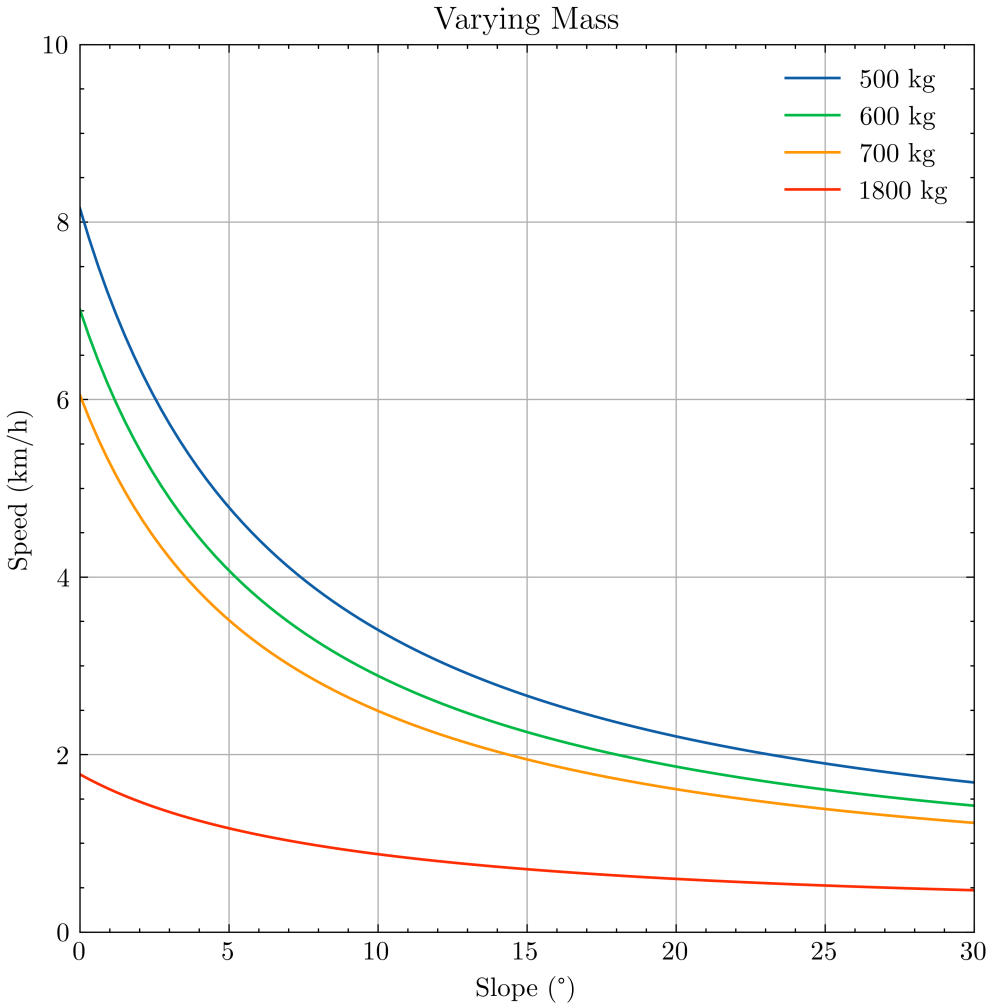


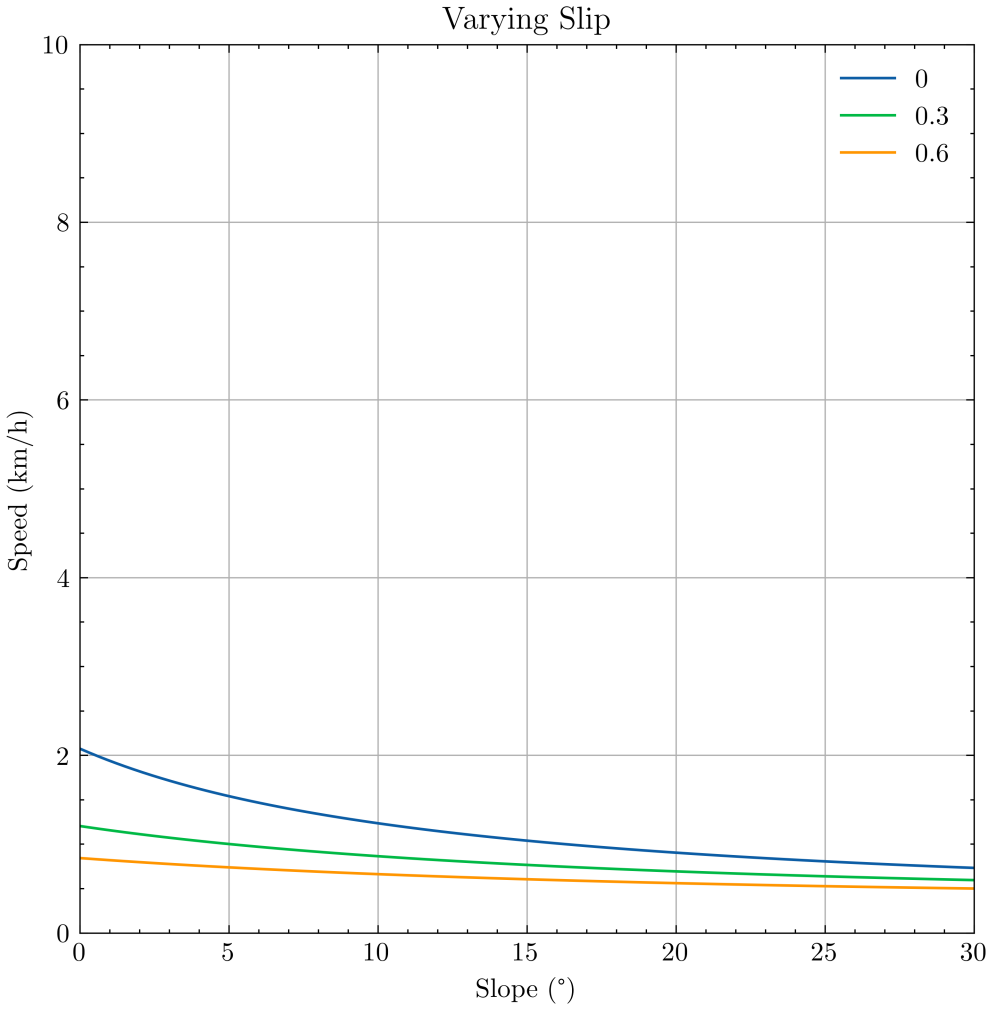

```
[ ]: slope_range = np.linspace(0, np.pi/6, 100) # range of slope values
     #slip = 0 # wheel slip ratio
     slip = 0.07+(0.0215*np.rad2deg(slope_range)) # wheel slip ratio

     # Define values for mass, slip, radius, and power
     mass_values = [500, 600, 700, 1800]
     slip_values = [0, 0.3, 0.6]
     radius_values = [0.3, 0.4, 0.5]
     power_values = [200, 300, 400, 500]
     width_values = [0.2, 0.3, 0.4, 0.5]
     gravity_values = [1.62, 3.71, 9.81]

     # Create subplots for Varying Mass
```

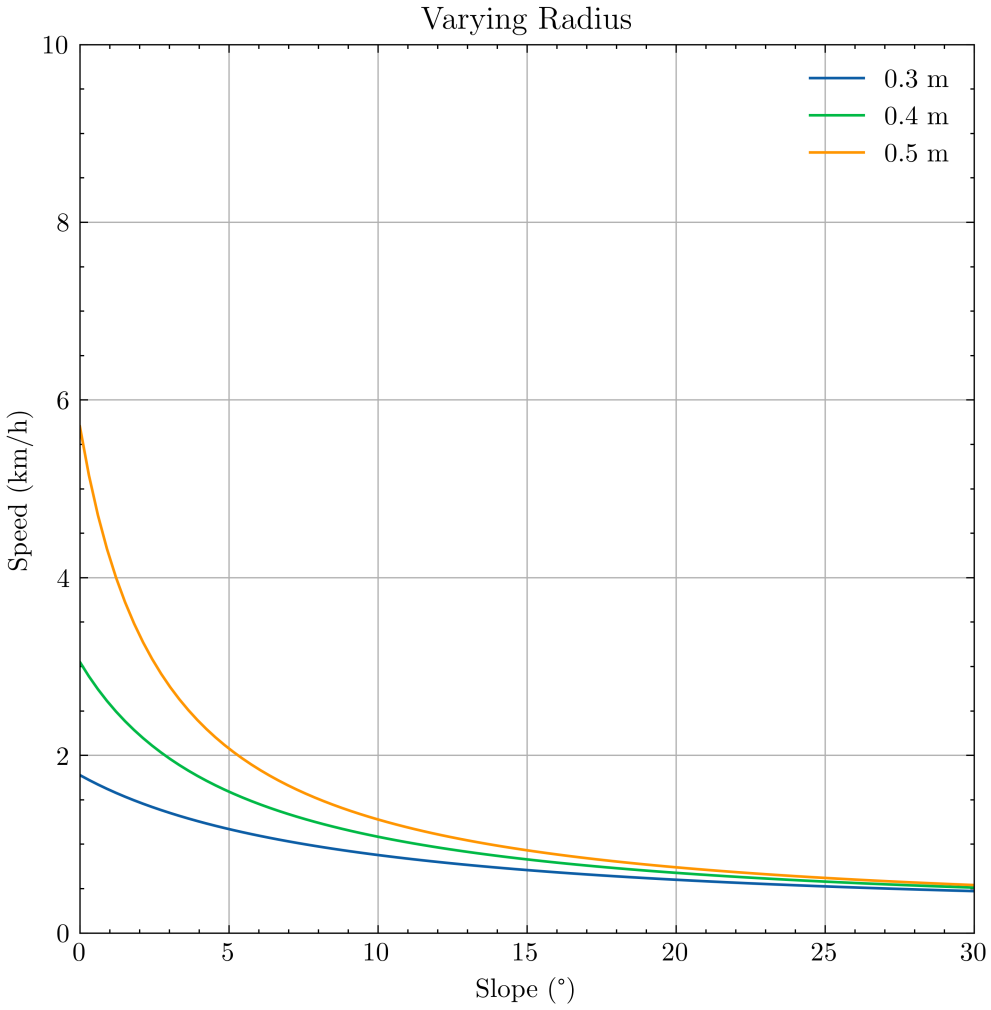
```
fig1, ax1 = plt.subplots(figsize=(6, 6))
for mass_n in mass_values:
    ax1.plot(np.rad2deg(slope_range), rover(mass_n/n_wheel, slip, radius,␣
 ↪power_constant, width, gravity, velocity_constant, slope_range)[0],␣
 ↪label=f"{mass_n} kg")
ax1.legend()
ax1.set_title("Varying Mass")
ax1.set_xlabel('Slope (°)')
ax1.set_ylabel('Speed (km/h)')
ax1.grid(True)
ax1.set_xlim(0, 30)
ax1.set_ylim(0, 10)
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_varying_mass.png")
plt.show()

# Create subplots for Varying Slip
fig2, ax2 = plt.subplots(figsize=(6, 6))
for slip_n in slip_values:
    ax2.plot(np.rad2deg(slope_range), rover(mass, slip_n, radius,␣
 ↪power_constant, width, gravity, velocity_constant, slope_range)[0],␣
 ↪label=f"{slip_n}")
ax2.legend()
ax2.set_title("Varying Slip")
ax2.set_xlabel('Slope (°)')
ax2.set_ylabel('Speed (km/h)')
ax2.grid(True)
ax2.set_xlim(0, 30)
ax2.set_ylim(0, 10)
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_varying_slip.png")
plt.show()

# Create subplots for Varying Radius
fig3, ax3 = plt.subplots(figsize=(6, 6))
for radius_n in radius_values:
    ax3.plot(np.rad2deg(slope_range), rover(mass, slip, radius_n,␣
 ↪power_constant, width, gravity, velocity_constant, slope_range)[0],␣
 ↪label=f"{radius_n} m")
ax3.legend()
ax3.set_title("Varying Radius")
ax3.set_xlabel('Slope (°)')
ax3.set_ylabel('Speed (km/h)')
ax3.grid(True)
ax3.set_xlim(0, 30)
```

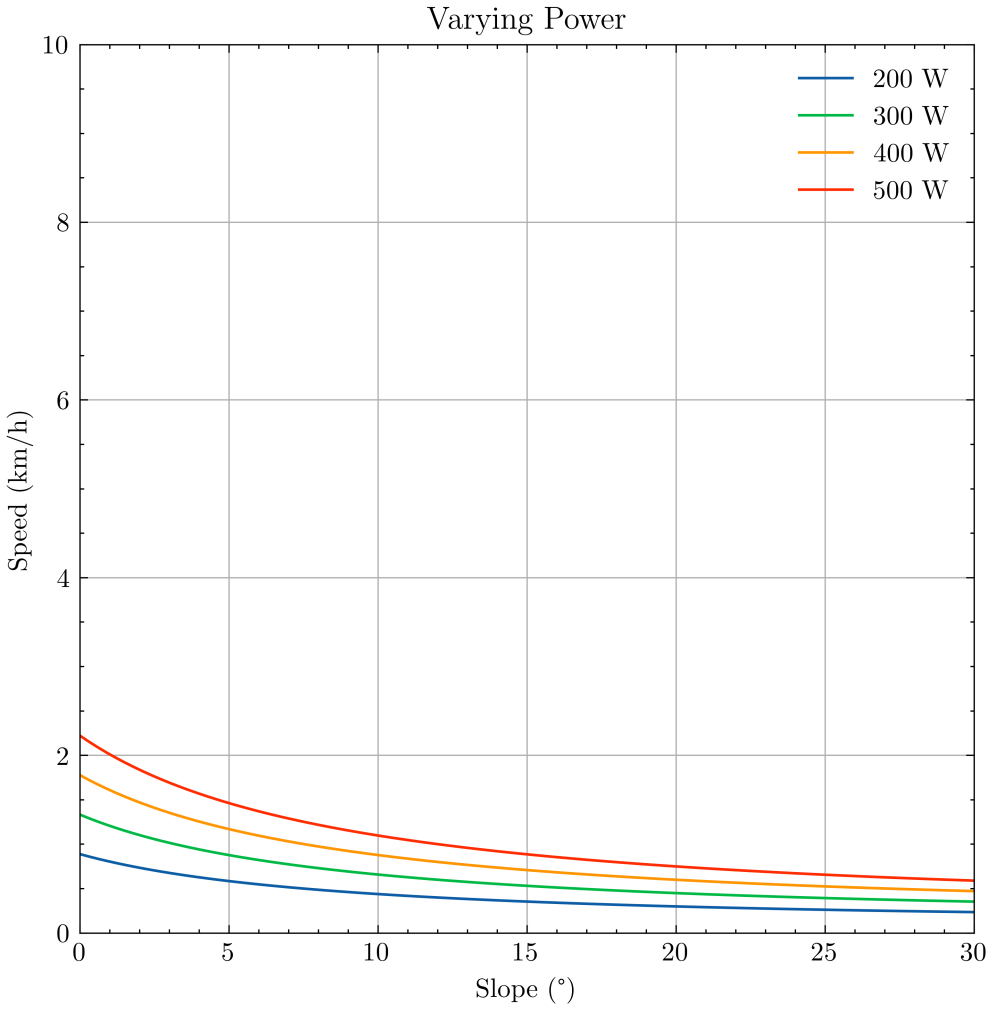
```
ax3.set_ylim(0, 10)
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_varying_radius.png")
plt.show()

# Create subplots for Varying Power
fig4, ax4 = plt.subplots(figsize=(6, 6))
for power_n in power_values:
    ax4.plot(np.rad2deg(slope_range), rover(mass, slip, radius, power_n/
 ↪n_wheel, width, gravity, velocity_constant, slope_range)[0],␣
 ↪label=f"{power_n} W")
ax4.legend()
ax4.set_title("Varying Power")
ax4.set_xlabel('Slope (°)')
ax4.set_ylabel('Speed (km/h)')
ax4.grid(True)
ax4.set_xlim(0, 30)
ax4.set_ylim(0, 10)
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_varying_power.png")
plt.show()

# Create subplots for Varying Width
fig5, ax5 = plt.subplots(figsize=(6, 6))
for width_n in width_values:
    ax5.plot(np.rad2deg(slope_range), rover(mass, slip, radius, power_constant,␣
 ↪width_n, gravity, velocity_constant, slope_range)[0], label=f"{width_n} m")
ax5.legend()
ax5.set_title("Varying Wheel Width")
ax5.set_xlabel('Slope (°)')
ax5.set_ylabel('Speed (km/h)')
ax5.grid(True)
ax5.set_xlim(0, 30)
ax5.set_ylim(0, 10)
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_varying_width.png")
plt.show()

# Create subplots for Varying Planet
fig6, ax6 = plt.subplots(figsize=(6, 6))
for gravity_n in gravity_values:
    ax6.plot(np.rad2deg(slope_range), rover(mass, slip, radius, power_constant,␣
 ↪width, gravity_n, velocity_constant, slope_range)[0], label=f"{gravity_n} m/
 ↪s2")
```

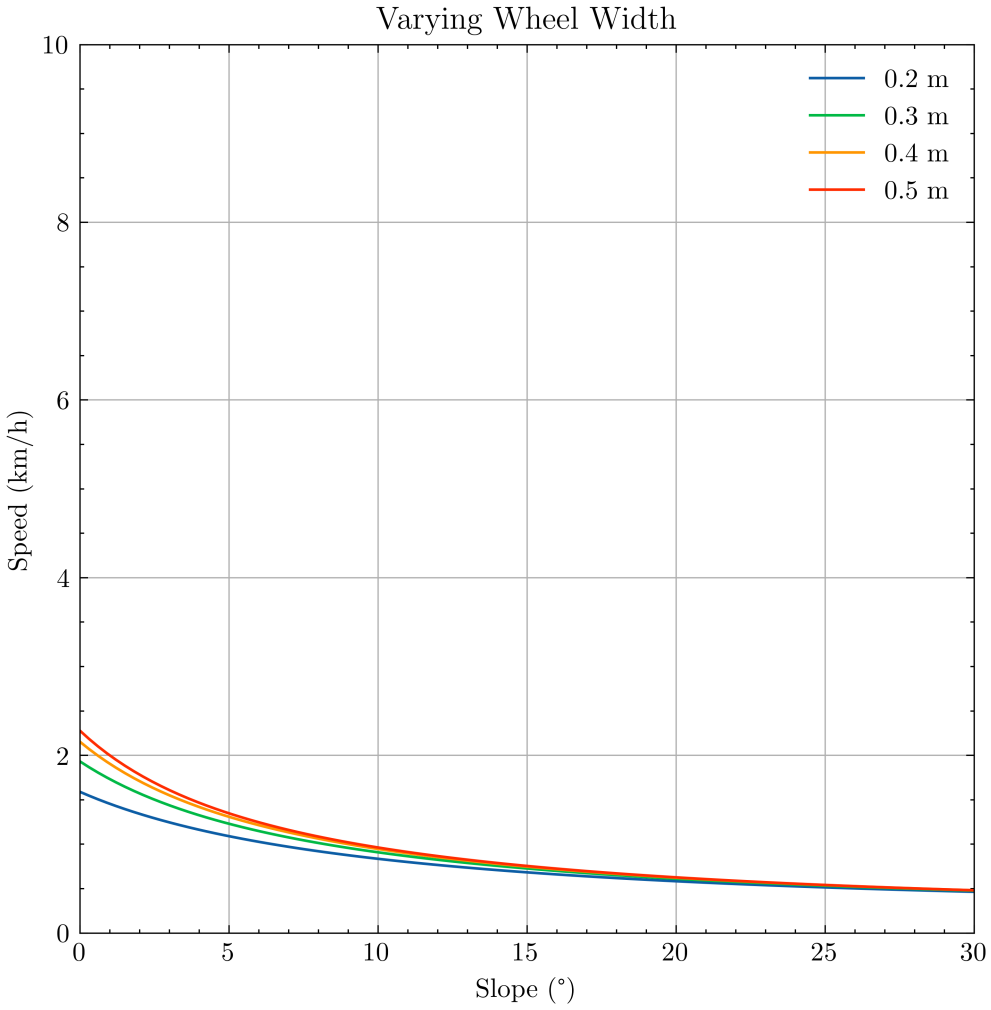
```
ax6.legend()
ax6.set_title("Varying Astronomical Body")
ax6.set_xlabel('Slope (°)')
ax6.set_ylabel('Speed (km/h)')
ax6.grid(True)
ax6.set_xlim(0, 30)
ax6.set_ylim(0, 10)
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
plt.savefig("tire_sim_varying_gravity.png")
plt.show()
```

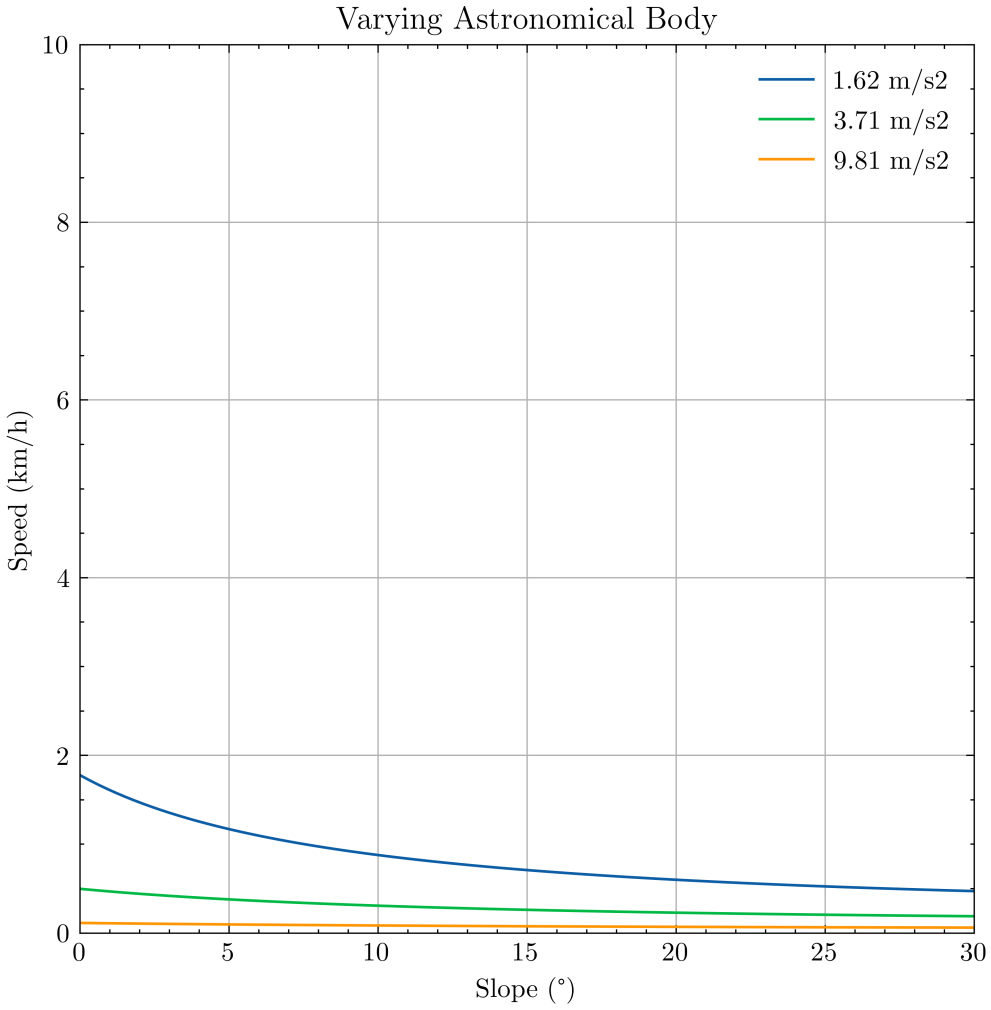



Varying Slip
0
0.3
0.6
Speed (km/h)
Slope (°)
0
2
4
6
8
10
0
5
10
15
20
25
30


Varying Radius
0.3 m
0.4 m
0.5 m
Speed (km/h)
Slope (°)
0
2
4
6
8
10
0
5
10
15
20
25
30


Varying Power
200 W
300 W
400 W
500 W
Speed (km/h)
Slope (°)
0
2
4
6
8
10
0
5
10
15
20
25
30


Varying Wheel Width
0.2 m
0.3 m
0.4 m
0.5 m
Speed (km/h)
Slope (°)
0
2
4
6
8
10
0
5
10
15
20
25
30


Varying Astronomical Body
1.62 m/s2
3.71 m/s2
9.81 m/s2
Speed (km/h)
Slope (°)
0
2
4
6
8
10
0
5
10
15
20
25
30