tire_sim

November 15, 2023

[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
import numpy as np
# Sven Nebendahl

# Load the CSV file into a DataFrame
data = pd.read_csv('quickmap-profile-data.csv')

# Extract x, y, and z values from the DataFrame
distance = data['position']
longitude = data['lon']+360
latitude = data['lat']
slope = data['TerrainSlope']

# Define the spherical_to_cartesian function
def spherical_to_cartesian(theta, phi):
    r_moon_southpole = 1737.5
    x = r_moon_southpole * np.sin(np.radians(theta)) * np.cos(np.radians(phi))
    y = r_moon_southpole * np.sin(np.radians(theta)) * np.sin(np.radians(phi))
    z = r_moon_southpole * np.cos(np.radians(theta))
    return x, y, z

# Convert spherical coordinates to Cartesian coordinates
moon_x, moon_y, moon_z = spherical_to_cartesian(longitude, latitude)

# Create a grid cartesian
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)

# Create a grid spherical
xi_cart = np.linspace(min(moon_x), max(moon_x), 1000)
yi_cart = np.linspace(min(moon_y), max(moon_y), 1000)
xi_cart, yi_cart = np.meshgrid(xi_cart, yi_cart)

# Interpolate the z values on the grid cartesian
```

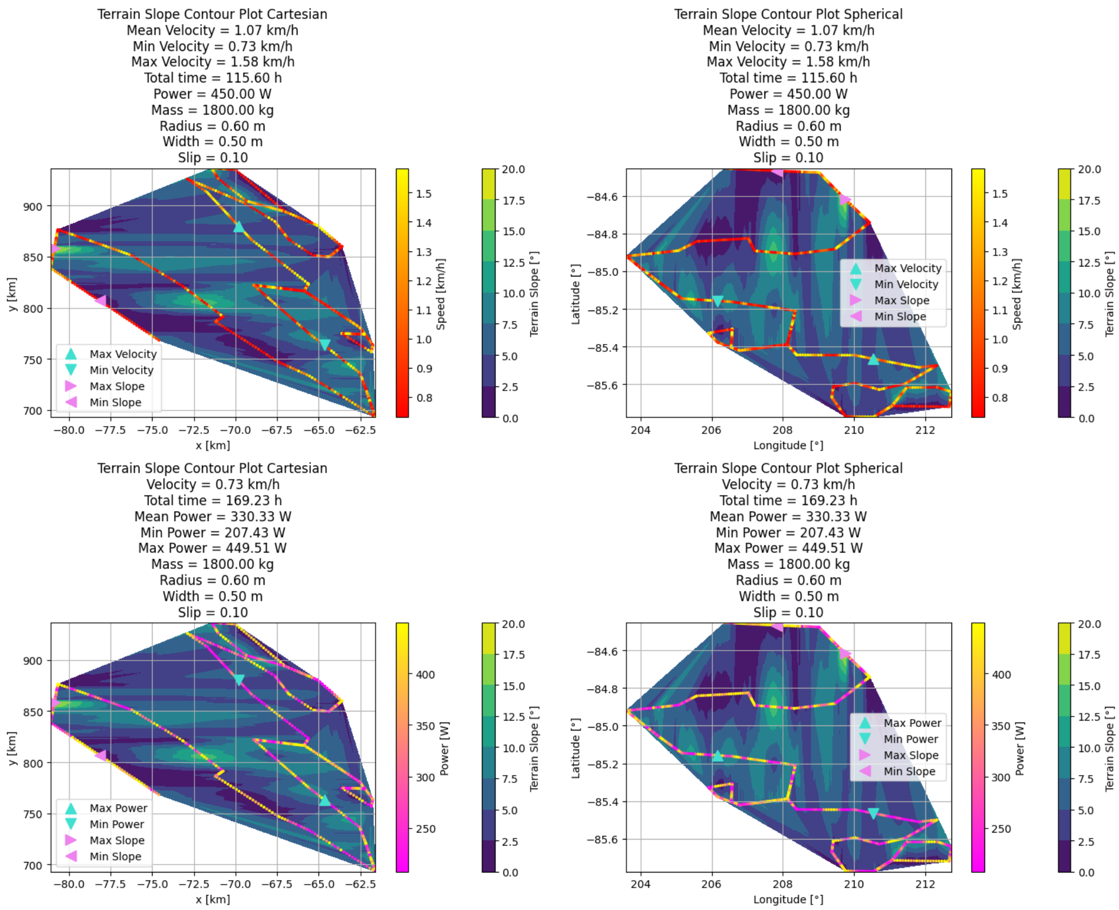
```
zi_cart = griddata((moon_x, moon_y), slope, (xi_cart, yi_cart), method='linear')

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

# Rover parameters
r = 0.6  # m
b = 0.25/0.3*r  # m
P = 450  # power for the whole rover with all 4 wheels
s = 0.1  # slip
g_moon = 1.62  # m/s^2
m = 1800
v = 0.73 # km/h
c_b = 0.017 * 100 ** 2  # N/cm^2 coefficient of soil/wheel cohesion
n = 1  # exponent of soil deformation and is dimensionless
k_c = 0.14 * 100 ** 2  # N/cm^2 cohesive modulus of soil deformation
k_phi = 0.82 * 100 ** 3  # N/cm^3 frictional modulus of soil deformation
phi_b = 35  # deg
alpha = slope  # Use the slope data from the CSV file

def rover(mass, slip, radius, power, width, gravity, speed):
    W = mass * gravity
    N = W * np.sin(alpha)
    D = 2 * radius
    k = k_c / width + k_phi
    z = (3 * W / ((3 - n) * (k_c + width * k_phi) * np.sqrt(D))) ** (2 / (2 * n + 1))
    theta_1 = np.arccos(1 - z / radius)
    theta_m = (0.45 + 0.24 * slip) * theta_1
    e = radius * np.sin(theta_1 - theta_m)
    l = radius * np.cos(theta_1 - theta_m)
    A = 2 * width * np.sqrt(D * z - z * z)  # contact area between the wheel and the ground
    R = width * k * z ** (n + 1) / (n + 1)
    H = A * c_b + W * np.tan(np.deg2rad(phi_b))
    DP = H - R
    T = DP * l + N * e
    velocity = power * radius * 3.6 / T
    PWR = (speed*T)/(r*3.6)
    return [velocity, PWR]

v_mean = np.mean(rover(m, s, r, P, b, g_moon, v)[0])
total_time = max(distance)/v_mean
v_min = np.min(rover(m, s, r, P, b, g_moon, v)[0])
v_max = np.max(rover(m, s, r, P, b, g_moon, v)[0])
```

```
# Create a figure with two subplots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 12))

# Subplot 1: Terrain Slope Contour Plot cartesian
ax1.grid(True)
contour = ax1.contourf(xi_cart, yi_cart, zi_cart, cmap='viridis')
fig.colorbar(contour, ax=ax1, label='Terrain Slope [°]')
scatter = ax1.scatter(moon_x, moon_y, c=rover(m, s, r, P, b, g_moon, v)[0],␣
 ↪linewidths=0.5, marker='o', s=5, cmap='autumn')
fig.colorbar(scatter, ax=ax1, label='Speed [km/h]')
ax1.axis('auto')
ax1.plot(moon_x[np.argmax(rover(m, s, r, P, b, g_moon, v)[0])], moon_y[np.
 ↪argmax(rover(m, s, r, P, b, g_moon, v)[0])], '^', color='turquoise',␣
 ↪markersize=10, label='Max Velocity')
ax1.plot(moon_x[np.argmin(rover(m, s, r, P, b, g_moon, v)[0])], moon_y[np.
 ↪argmin(rover(m, s, r, P, b, g_moon, v)[0])], 'v', color='turquoise',␣
 ↪markersize=10, label='Min Velocity')
ax1.plot(moon_x[np.argmax(alpha)], moon_y[np.argmax(alpha)], '>',␣
 ↪color='violet', markersize=10, label='Max Slope')
ax1.plot(moon_x[np.argmin(alpha)], moon_y[np.argmin(alpha)], '<',␣
 ↪color='violet', markersize=10, label='Min Slope')
legend = ax1.legend()
ax1.set_xlabel('x [km]')
ax1.set_ylabel('y [km]')
ax1.set_title('Terrain Slope Contour Plot Cartesian\n'
              'Mean Velocity = {:.2f} km/h\n'
              'Min Velocity = {:.2f} km/h\n'
              'Max Velocity = {:.2f} km/h\n'
              'Total time = {:.2f} h\n'
              'Power = {:.2f} W\n'
              'Mass = {:.2f} kg\n'
              'Radius = {:.2f} m\n'
              'Width = {:.2f} m\n'
              'Slip = {:.2f}'.format(v_mean, v_min, v_max, total_time, P, m, r,␣
 ↪b, s))

# Subplot 2: Terrain Slope Contour Plot spherical
ax2.grid(True)
contour = ax2.contourf(xi_sphere, yi_sphere, zi_sphere, cmap='viridis')
fig.colorbar(contour, ax=ax2, label='Terrain Slope [°]')
scatter = ax2.scatter(longitude, latitude, c=rover(m, s, r, P, b, g_moon,␣
 ↪v)[0], linewidths=0.5, marker='o', s=5, cmap='autumn')
fig.colorbar(scatter, ax=ax2, label='Speed [km/h]')
ax2.axis('auto')
```

```
ax2.plot(longitude[np.argmax(rover(m, s, r, P, b, g_moon, v)[0])], latitude[np.
 ↪argmax(rover(m, s, r, P, b, g_moon, v)[0])], '^', color='turquoise',␣
 ↪markersize=10, label='Max Velocity')
ax2.plot(longitude[np.argmin(rover(m, s, r, P, b, g_moon, v)[0])], latitude[np.
 ↪argmin(rover(m, s, r, P, b, g_moon, v)[0])], 'v', color='turquoise',␣
 ↪markersize=10, label='Min Velocity')
ax2.plot(longitude[np.argmax(alpha)], latitude[np.argmax(alpha)], '>',␣
 ↪color='violet', markersize=10, label='Max Slope')
ax2.plot(longitude[np.argmin(alpha)], latitude[np.argmin(alpha)], '<',␣
 ↪color='violet', markersize=10, label='Min Slope')
legend = ax2.legend()
ax2.set_xlabel('Longitude [°]')
ax2.set_ylabel('Latitude [°]')
ax2.set_title('Terrain Slope Contour Plot Spherical\n'
              'Mean Velocity = {:.2f} km/h\n'
              'Min Velocity = {:.2f} km/h\n'
              'Max Velocity = {:.2f} km/h\n'
              'Total time = {:.2f} h\n'
              'Power = {:.2f} W\n'
              'Mass = {:.2f} kg\n'
              'Radius = {:.2f} m\n'
              'Width = {:.2f} m\n'
              'Slip = {:.2f}'.format(v_mean, v_min, v_max, total_time, P, m, r,␣
 ↪b, s))

v_mean = v
total_time = max(distance)/v_mean
P_mean = np.mean(rover(m, s, r, P, b, g_moon, v)[1])
P_min = np.min(rover(m, s, r, P, b, g_moon, v)[1])
P_max = np.max(rover(m, s, r, P, b, g_moon, v)[1])
# Subplot 3: Terrain Slope Contour Plot cartesian
ax3.grid(True)
contour = ax3.contourf(xi_cart, yi_cart, zi_cart, cmap='viridis')
fig.colorbar(contour, ax=ax3, label='Terrain Slope [°]')
scatter = ax3.scatter(moon_x, moon_y, c=rover(m, s, r, P, b, g_moon, v)[1],␣
 ↪linewidths=0.5, marker='o', s=5, cmap='spring')
fig.colorbar(scatter, ax=ax3, label='Power [W]')
ax3.axis('auto')
ax3.plot(moon_x[np.argmax(rover(m, s, r, P, b, g_moon, v)[1])], moon_y[np.
 ↪argmax(rover(m, s, r, P, b, g_moon, v)[1])], '^', color='turquoise',␣
 ↪markersize=10, label='Max Power')
ax3.plot(moon_x[np.argmin(rover(m, s, r, P, b, g_moon, v)[1])], moon_y[np.
 ↪argmin(rover(m, s, r, P, b, g_moon, v)[1])], 'v', color='turquoise',␣
 ↪markersize=10, label='Min Power')
ax3.plot(moon_x[np.argmax(alpha)], moon_y[np.argmax(alpha)], '>',␣
 ↪color='violet', markersize=10, label='Max Slope')
```

```python
ax3.plot(moon_x[np.argmin(alpha)], moon_y[np.argmin(alpha)], '<', color='violet', markersize=10, label='Min Slope')
legend = ax3.legend()
ax3.set_xlabel('x [km]')
ax3.set_ylabel('y [km]')
ax3.set_title('Terrain Slope Contour Plot Cartesian\n'
              '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\n'
              'Slip = {:.2f}'.format(v_mean, total_time, P_mean, P_min, P_max, m, r, b, s))

# Subplot 4: Terrain Slope Contour Plot spherical
ax4.grid(True)
contour = ax4.contourf(xi_sphere, yi_sphere, zi_sphere, cmap='viridis')
fig.colorbar(contour, ax=ax4, label='Terrain Slope [°]')
scatter = ax4.scatter(longitude, latitude, c=rover(m, s, r, P, b, g_moon, v)[1], linewidths=0.5, marker='o', s=5, cmap='spring')
fig.colorbar(scatter, ax=ax4, label='Power [W]')
ax4.axis('auto')
ax4.plot(longitude[np.argmax(rover(m, s, r, P, b, g_moon, v)[1])], latitude[np.argmax(rover(m, s, r, P, b, g_moon, v)[1])], '^', color='turquoise', markersize=10, label='Max Power')
ax4.plot(longitude[np.argmin(rover(m, s, r, P, b, g_moon, v)[1])], latitude[np.argmin(rover(m, s, r, P, b, g_moon, v)[1])], 'v', color='turquoise', markersize=10, label='Min Power')
ax4.plot(longitude[np.argmax(alpha)], latitude[np.argmax(alpha)], '>', color='violet', markersize=10, label='Max Slope')
ax4.plot(longitude[np.argmin(alpha)], latitude[np.argmin(alpha)], '<', color='violet', markersize=10, label='Min Slope')
legend = ax4.legend()
ax4.set_xlabel('Longitude [°]')
ax4.set_ylabel('Latitude [°]')
ax4.set_title('Terrain Slope Contour Plot Spherical\n'
              '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\n'
                'Slip = {:.2f}'.format(v_mean, total_time, P_mean, P_min, P_max,␣
↪m, r, b, s))

# Adjust layout to prevent clipping of titles
plt.tight_layout()
plt.rcParams['figure.dpi'] = 1200
plt.rcParams['savefig.dpi'] = 1200
plt.show()
print(data)
```



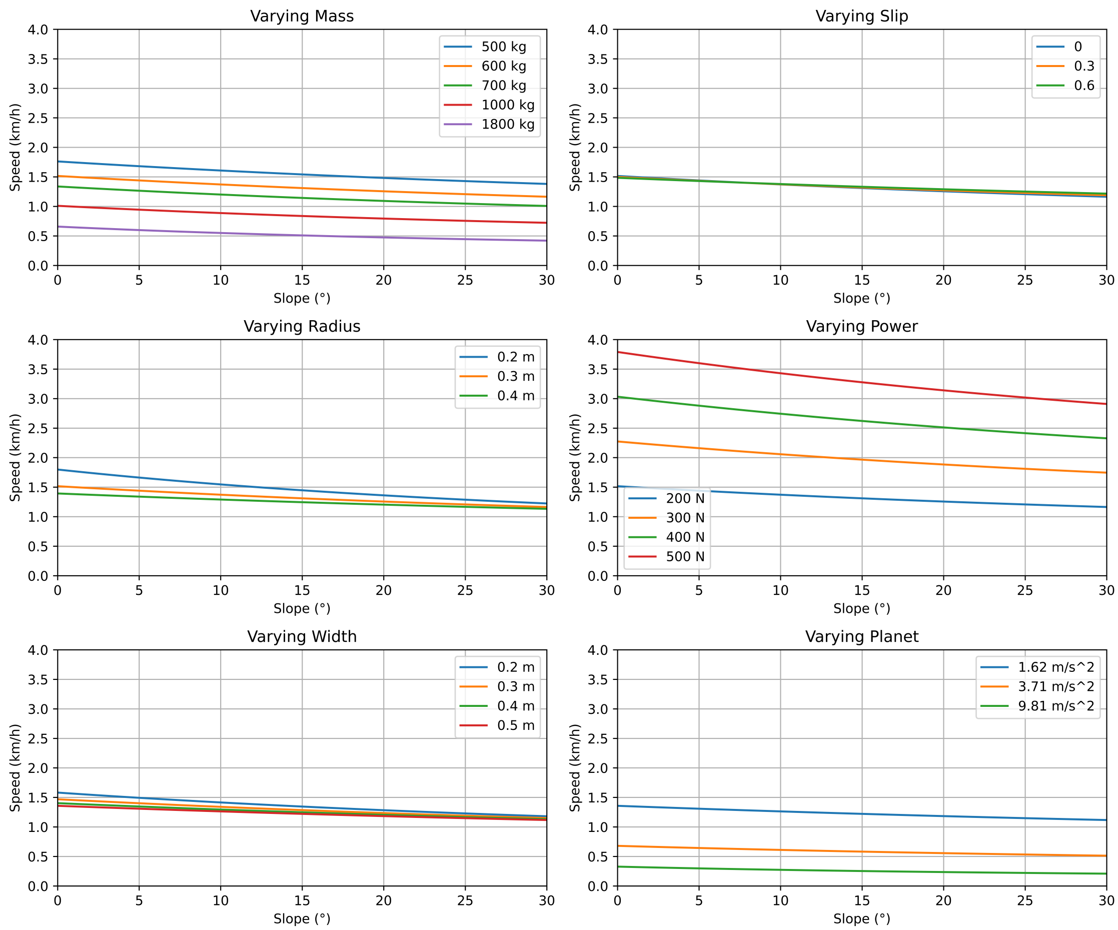

```
       position  TerrainSlope         lon        lat
0      0.000000      2.123901 -150.241801 -85.770015
1      0.176485      2.890792 -150.163092 -85.770429
2      0.352970      3.534304 -150.084367 -85.770836
3      0.529455      4.026555 -150.005627 -85.771235
4      0.705940      4.054826 -149.926873 -85.771625
..          …             …           …          …
717  122.837356      3.103476 -153.390440 -84.456294
```

```
718  123.013841      4.565214 -153.450317 -84.455654
719  123.190326      6.067159 -153.510181 -84.455008
720  123.366811      6.424311 -153.570031 -84.454356
721  123.539578      6.616184 -153.628606 -84.453712

[722 rows x 4 columns]
```

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

r = 0.3  # m
b = 0.25  # m
P = 200  # power for the whole rover with all 4 wheels
s = 0 # slip
g_moon = 1.62  # m/s^2
m = 600
c_b = 0.017 * 100 ** 2  # N/cm^2 coefficient of soil/wheel cohesion
n = 1  # exponent of soil deformation and is dimensionless
k_c = 0.14 * 100 ** 2  # N/cm^2 cohesive modulus of soil deformation
k_phi = 0.82 * 100 ** 3  # N/cm^3 frictional modulus of soil deformation
phi_b = 35  # deg
alpha = np.linspace(0, np.pi / 6, 100)  # slope

def fun(mass, slip, radius, power, width, gravity):
    W = mass * gravity
    N = W * np.sin(alpha)
    D = 2 * radius
    k = k_c / width + k_phi
    z = (3 * W / ((3 - n) * (k_c + width * k_phi) * np.sqrt(D))) ** (2 / (2 * n␣
 ↪+ 1))
    theta_1 = np.arccos(1 - z / radius)
    theta_m = (0.45 + 0.24 * slip) * theta_1
    e = radius * np.sin(theta_1 - theta_m)
    l = radius * np.cos(theta_1 - theta_m)
    A = 2 * width * np.sqrt(D * z - z * z)  # contact area between the wheel␣
 ↪and the ground
    R = width * k * z ** (n + 1) / (n + 1)
    H = A * c_b + W * np.tan(np.deg2rad(phi_b))
    DP = H - R
    T = DP * l + N * e
    velocity = power * radius * 3.6 / T
    return velocity

# Define values for mass, slip, radius, and power
mass_values = [500, 600, 700, 1000, 1800]
slip_values = [0, 0.3, 0.6]
radius_values = [0.2, 0.3, 0.4]
```

```
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
fig, axes = plt.subplots(3, 2, figsize=(12, 10))

# Varying mass subplot
for mass in mass_values:
    axes[0, 0].plot(np.rad2deg(alpha), fun(mass, s, r, P, b, g_moon),␣
 ↪label=f"{mass} kg")

axes[0, 0].legend()
axes[0, 0].set_title("Varying Mass")
axes[0, 0].set_xlabel('Slope (°)')
axes[0, 0].set_ylabel('Speed (km/h)')
axes[0, 0].grid(True)
axes[0, 0].set_xlim(0, 30)
axes[0, 0].set_ylim(0, 4)

# Varying slip subplot
for slip in slip_values:
    axes[0, 1].plot(np.rad2deg(alpha), fun(m, slip, r, P, b, g_moon),␣
 ↪label=f"{slip}")

axes[0, 1].legend()
axes[0, 1].set_title("Varying Slip")
axes[0, 1].set_xlabel('Slope (°)')
axes[0, 1].set_ylabel('Speed (km/h)')
axes[0, 1].grid(True)
axes[0, 1].set_xlim(0, 30)
axes[0, 1].set_ylim(0, 4)

# Varying radius subplot
for radius in radius_values:
    axes[1, 0].plot(np.rad2deg(alpha), fun(m, s, radius, P, b, g_moon),␣
 ↪label=f"{radius} m")

axes[1, 0].legend()
axes[1, 0].set_title("Varying Radius")
axes[1, 0].set_xlabel('Slope (°)')
axes[1, 0].set_ylabel('Speed (km/h)')
axes[1, 0].grid(True)
axes[1, 0].set_xlim(0, 30)
axes[1, 0].set_ylim(0, 4)

# Varying power subplot
```

```
for power in power_values:
    axes[1, 1].plot(np.rad2deg(alpha), fun(m, s, r, power, b, g_moon),␣
 ↪label=f"{power} N")

axes[1, 1].legend()
axes[1, 1].set_title("Varying Power")
axes[1, 1].set_xlabel('Slope (°)')
axes[1, 1].set_ylabel('Speed (km/h)')
axes[1, 1].grid(True)
axes[1, 1].set_xlim(0, 30)
axes[1, 1].set_ylim(0, 4)

# Varying width subplot
for width in width_values:
    axes[2, 0].plot(np.rad2deg(alpha), fun(m, s, r, P, width, g_moon),␣
 ↪label=f"{width} m")

axes[2, 0].legend()
axes[2, 0].set_title("Varying Width")
axes[2, 0].set_xlabel('Slope (°)')
axes[2, 0].set_ylabel('Speed (km/h)')
axes[2, 0].grid(True)
axes[2, 0].set_xlim(0, 30)
axes[2, 0].set_ylim(0, 4)

# Varying Planet subplot
for gravity in gravity_values:
    axes[2, 1].plot(np.rad2deg(alpha), fun(m, s, r, P, width, gravity),␣
 ↪label=f"{gravity} m/s^2")

axes[2, 1].legend()
axes[2, 1].set_title("Varying Planet")
axes[2, 1].set_xlabel('Slope (°)')
axes[2, 1].set_ylabel('Speed (km/h)')
axes[2, 1].grid(True)
axes[2, 1].set_xlim(0, 30)
axes[2, 1].set_ylim(0, 4)

plt.tight_layout()
plt.rcParams['figure.dpi'] = 1200
plt.rcParams['savefig.dpi'] = 1200
plt.show()
```

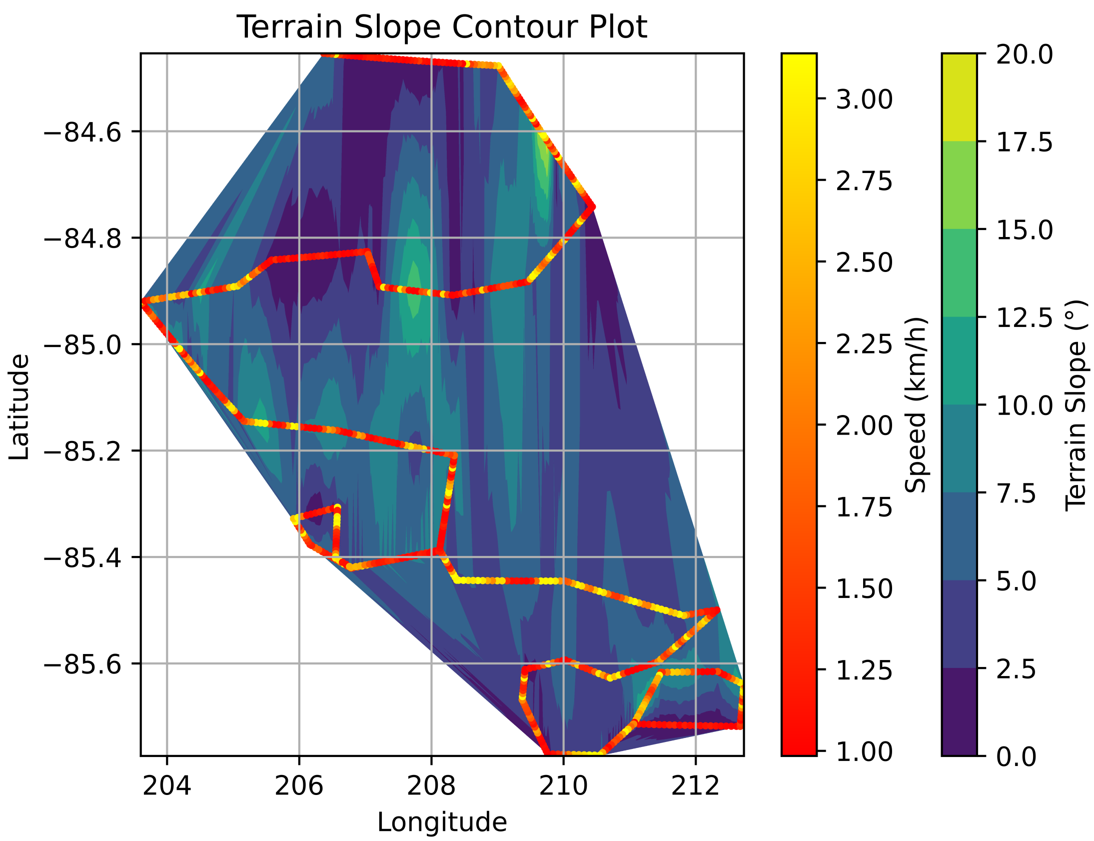


[ ]:

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
import numpy as np

# Load the CSV file into a DataFrame
data = pd.read_csv('quickmap-profile-data.csv')

# Extract x, y, and z values from the DataFrame
longitude = data['lon']+360
latitude = data['lat']
slope = data['TerrainSlope']

# Create a grid
xi = np.linspace(min(longitude), max(longitude), 10000)
yi = np.linspace(min(latitude), max(latitude), 10000)
xi, yi = np.meshgrid(xi, yi)
```

```python
# Interpolate the z values on the grid
zi = griddata((longitude, latitude), slope, (xi, yi), method='linear')

r = 0.3  # m
b = 0.25  # m
P = 200  # power for the whole rover with all 4 wheels
s = 0.3 # slip
g_moon = 1.62  # m/s^2
m = 600
c_b = 0.017 * 100 ** 2  # N/cm^2 coefficient of soil/wheel cohesion
n = 1  # exponent of soil deformation and is dimensionless
k_c = 0.14 * 100 ** 2  # N/cm^2 cohesive modulus of soil deformation
k_phi = 0.82 * 100 ** 3  # N/cm^3 frictional modulus of soil deformation
phi_b = 35  # deg
#alpha = np.linspace(0, np.pi / 6, 100)  # slope
alpha = slope

def fun(mass, slip, radius, power, width, gravity):
    W = mass * gravity
    N = W * np.sin(alpha)
    D = 2 * radius
    k = k_c / width + k_phi
    z = (3 * W / ((3 - n) * (k_c + width * k_phi) * np.sqrt(D))) ** (2 / (2 * n + 1))
    theta_1 = np.arccos(1 - z / radius)
    theta_m = (0.45 + 0.24 * slip) * theta_1
    e = radius * np.sin(theta_1 - theta_m)
    l = radius * np.cos(theta_1 - theta_m)
    A = 2 * width * np.sqrt(D * z - z * z)  # contact area between the wheel and the ground
    R = width * k * z ** (n + 1) / (n + 1)
    H = A * c_b + W * np.tan(np.deg2rad(phi_b))
    DP = H - R
    T = DP * l + N * e
    velocity = power * radius * 3.6 / T
    return velocity

# Create a contour plot
plt.grid(True)
contour = plt.contourf(xi, yi, zi, cmap='viridis')
plt.colorbar(contour, label='Terrain Slope (°)')
scatter = plt.scatter(longitude, latitude, c=fun(m, s, r, P, b, g_moon), linewidths=0.5, marker='o', s=5, cmap='autumn')
plt.colorbar(scatter, label='Speed (km/h)')
plt.axis('auto')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
```

```python
plt.title('Terrain Slope Contour Plot')
plt.rcParams['figure.dpi'] = 1200
plt.rcParams['savefig.dpi'] = 1200
plt.show()
print(data)
```




```
        position  TerrainSlope         lon        lat
0       0.000000      2.123901 -150.241801 -85.770015
1       0.176485      2.890792 -150.163092 -85.770429
2       0.352970      3.534304 -150.084367 -85.770836
3       0.529455      4.026555 -150.005627 -85.771235
4       0.705940      4.054826 -149.926873 -85.771625
..           ...           ...         ...        ...
717   122.837356      3.103476 -153.390440 -84.456294
718   123.013841      4.565214 -153.450317 -84.455654
719   123.190326      6.067159 -153.510181 -84.455008
720   123.366811      6.424311 -153.570031 -84.454356
721   123.539578      6.616184 -153.628606 -84.453712

[722 rows x 4 columns]
```