Regosketch 3 : Simulate Compression of Lunar Regolith under a Wheel

Mick Goulish
19 April, 2023

Overview

I'm trying here to make a simple finite-element model of soil compression and see how realistic it is in predicting how far the wheels of a vehicle will sink into the lunar regolith. I have one way to compare it to reality: by looking at the depth of the tracks left by the Apollo Lunar Rover. (The "Moon Buggy".) So I simulate two vehicles: The Apollo Moon Buggy, which massed something like 660 kg fully loaded. And a hypothetical bulldozer for my power company, at 6000 kg.

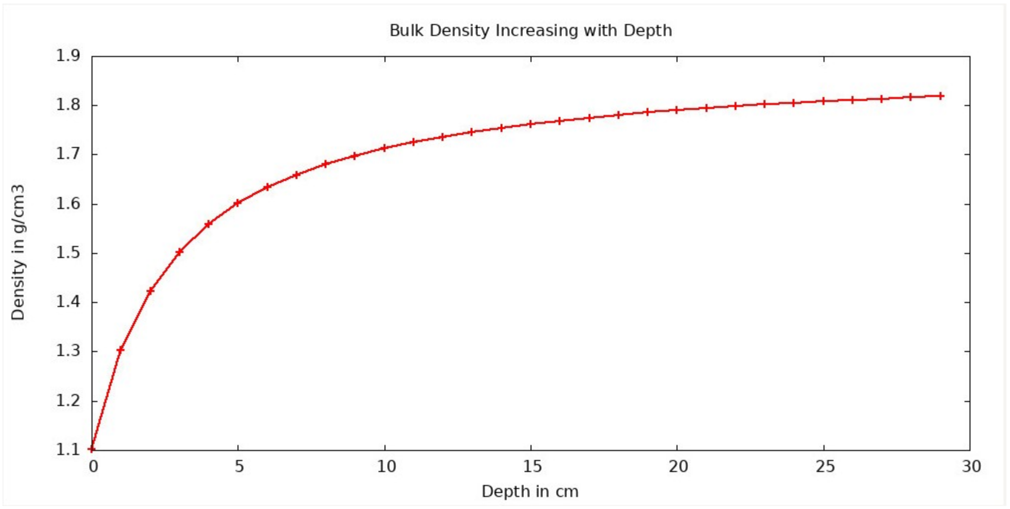
Doing the Simulation

I found a NASA page at https://ntrs.nasa.gov/citations/19720035207 that gives their estimate of the load-bearing strength of lunar regolith at different depths, in Newtons per square centimeter. They don't give a formula, but only values for two points: 0.02-0.04 N/sq cm at the surface, and 30-100 N/sq cm where the regolith is deep enough to reach 1.9 grams/cm3.

Since they give a range at both points, I took the average of both of those ranges: 0.03 N/cm2 at the surface and 65 N/cm2 at 30 cm depth. (Using the formula from our Quant 1 assignment for how density increases with depth.)

I just linearly increase load-bearing capacity between those two points, which could probably be improved.

Here is a graph of what that density increase looks like, with the formula from Quant 1.



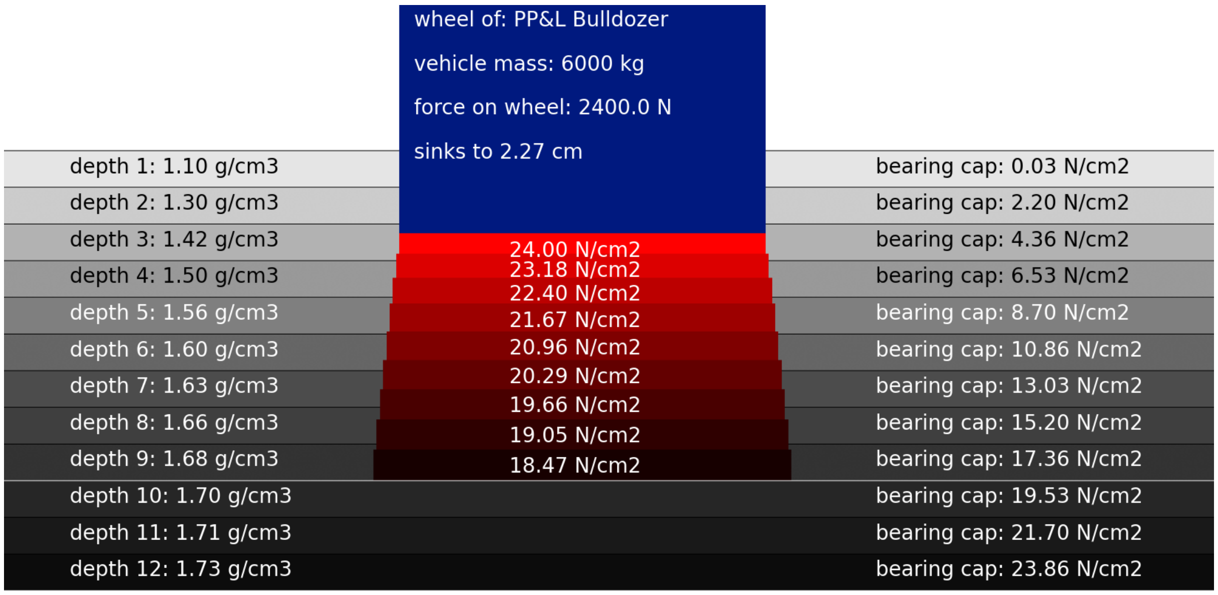
Then I do the simulation this way:

- Take the mass of the vehicle and multiply by lunar gravity to get total downward force in Newtons.
- Divide by 4 to get downward force per wheel.
- Assign a footprint to the wheel, which is just the squared width of the wheel. Not dealing with the wheel curvature here. Maybe it flattens out a little at the point of contact.
- Divide total force by footprint area to get Newtons per square centimeter.
- Look at the load-bearing strength of the top centimeter of regolith. If it is less than the footprint force per square cm, then collapse it. Compress it right down to its grain density of 1.9 g/cm3.
- Find out how thick that layer is after it collapses to grain density, and store that.
- Iterate downward, layer by layer in the same way, collapsing each layer to grain density if it is not strong enough to support the load.
- Except: as you go downward, the force footprint gets larger because there is some interaction between the particles on the edge of the footprint with particles just outside of it.
- It does not get larger very fast though, or we would see little depressed areas around the Apollo astronauts' footprints, and we don't. So I expand that force footprint just slowly, at an angle of 5 degrees downward.
- So the force per square cm decreases as you go downward, and the load-bearing capacity of the soil rises.
- Eventually you reach a point where the load bearing capacity of a layer exceeds the force per square cm of the force footprint. That layer does not collapse, and we stop there.
- Add up how much each collapsed layer was reduced in size (it decreases as you go downward) and the total is how far the wheel sank.
- Draw it all out as a graphic.

Results

My bulldozer is massive and has kind of thin wheels, at 10 cm.

Here is a graphic showing how each layer compresses beneath the wheel.



Peary Power and Light bulldozer wheel, compressing lunar regolith

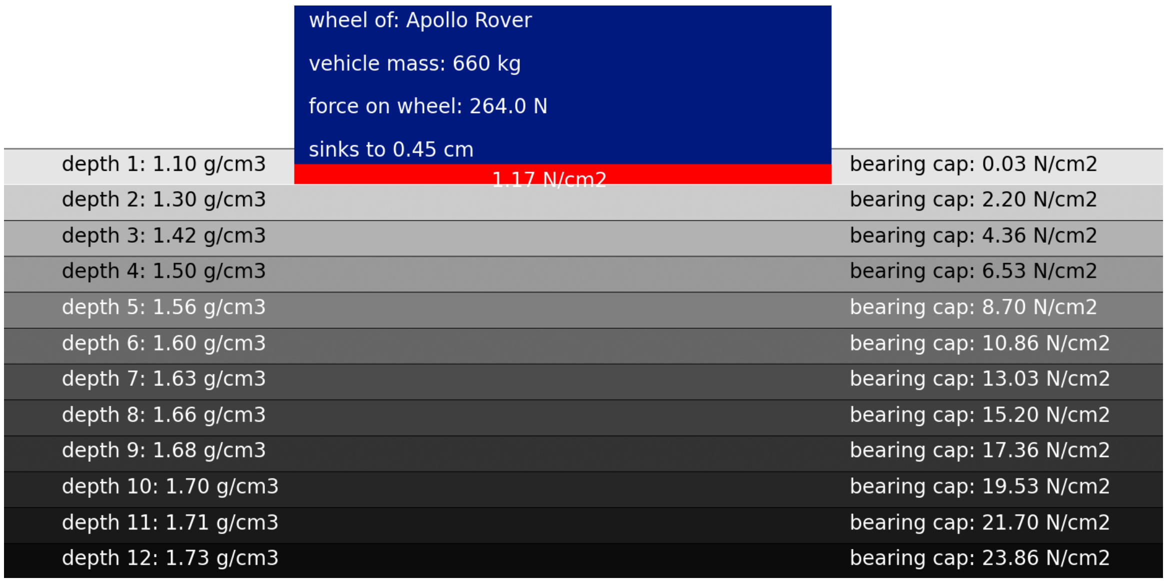
On the left you can read the depth of each regolith layer and its density. The blue thing at the top is the wheel cross-section (looking at it edgewise), showing the vehicle mass, force on this wheel, and how far it sank.

The red area beneath the wheel is the compressed layers of regolith, showing the force per cm2 on each layer. You can see how they slowly widen as you go down, with the 5 degree angle I chose for sideways spread.

On the right you can see the bearing capacity of each layer. The compression has stopped at 10 cm depth because the 10 cm layer has a load-bearing capacity higher than the footprint force.

That does not mean that the vehicle has sunk 10 cm. The sink-depth is the sum of how much each layer got smaller when it got compressed to grain-density. In this case total compression was 2.27 cm, almost an inch.

The Apollo Moon Buggy had really wide wheels, and it wasn’t very massive (compared to my bulldozer). It hardly sinks in at all: only about half a centimeter.



Apollo Moon Buggy wheel, barely there

I show this graphic second because it isn't as interesting. The Moon Buggy is only compressing a single layer and only sinks in about half a centimeter.

Is this realistic? It isn't very far off! In the pictures I have been able to find, there are no deep ruts visible behind the lunar rover. In some pictures it looks like it has only left tread-marks, so only sinking in to whatever the depth of those treads was.

How Do I Run It?

I'm including the code at the end of this paper, but it is also available at:

https://github.com/mgoulish/regosketch_3

To get it, do this: `git clone https://github.com/mgoulish/regosketch_3`

Then go into the directory it made, make sure the Python file is executable, and execute it. It will create an image file called 'image.png' which you can move to a new name if you want it to not get stepped on by the next run of the program.

The Code

Here is the actual code, in case this is the only way you can get it.

```
#! /usr/bin/env python3

from PIL import Image, ImageDraw, ImageFont, ImageOps
import matplotlib.pyplot as mpl
import math
import numpy as np
from skimage.draw import (line,
                          disk,
                          rectangle)


# Colors ---------------------------
white       = ( 1.0, 1.0, 1.0 )
black       = ( 0.0, 0.0, 0.0 )
wheel_color = ( 0.0, 0.1, 0.5 )


#------------------------------------
# Return colors to represent regolith
# density. Gray value gets darker as
# depth increases.
#------------------------------------
def gray_value_for_depth ( depth ) :
  if depth <= 6 :
    return 0.9 - 0.1 * depth
  elif depth <= 11 :
    return 0.25 - 0.05 * (depth - 7)
  else :
    return 0.0



#-------------------------------------------------------
# I got these constants by running this program and
# printing out the density values from 0 to 29 cm deep.
#-------------------------------------------------------
min_density = 1.101
max_density = 1.818
density_range = max_density - min_density


#-----------------------------------------------------
# There are two vehicles here. Uncomment whichever
# one you want to simulate. The wheel contact areas
# are an estimate based on wheel width.
# The PP&L bulldozer is fictional (so far). It has a
# larger contact area than the Apollo rover because
# it is a large, heavy construction vehicle.
#-----------------------------------------------------
```

```
# Choose one of these, and comment out the other one.
scenario = "Moon_Buggy"
#scenario = "PP&L"




if scenario == "Moon_Buggy" :
  # Apollo Lunar rover  -------------------------------------------
  vehicle_name       = "Apollo Rover"
  mass_of_vehicle    = 660  # kg
  wheel_width        = 15   # cm (will be squared)
  n_wheels           = 4
elif scenario == "PP&L" :
  # Peary Power and Light bulldozer -----------------------------
  vehicle_name       = "PP&L Bulldozer"
  mass_of_vehicle    = 6000  # kg
  wheel_width        = 10    # cm (will be squared)
  n_wheels           = 4
else :
  print ( "Bad scenario!" )
  sys.exit ( 1 )



lunar_gravity          = 1.6   # meters / second
force_to_support       = mass_of_vehicle * lunar_gravity  # Newtons
force_per_wheel        = force_to_support / n_wheels

# Shallow angle of force-spreading with depth, because
# we do not see little depressed areas around the Apollo
# astronauts' footprints.
angle_of_force_spread  = 5     # degrees
force_spread           = math.tan(math.radians(angle_of_force_spread))


# This is the density to which each overloaded
# layer will collapse.
grain_density = 2.0    # g/cm3

#-----------------------------------------------------------
# These constants are from NASA page:
# https://ntrs.nasa.gov/citations/19720035207
# The min value at near-surface and the max value at
# 30 cm is all they gave me. No formula.
# So I will scale it linearly with depth, which could
# probably be improved.
#-----------------------------------------------------------
min_bearing_cap   = 0.03   # Newtons per square cm
bearing_per_cm    = 65.0 / 30


def density_and_bearing_cap_at_depth ( depth ) :
  density  = 1.89 * (depth + 1.69) / (depth + 2.9)
  bearing_cap = min_bearing_cap + depth * bearing_per_cm
  return density, bearing_cap
```

```
#============================================
# Main
#============================================

#-------------------------------------------------
# Pre-calculate density and bearing capacity for
# each centimeter down to 30, and store the values
# in arrays.
#-------------------------------------------------
max_depth = 30
density     = []
bearing_cap = []

for depth in range(max_depth) :
  d, c = density_and_bearing_cap_at_depth ( depth )
  print ( f"depth {depth} cap {c}" )
  density.append(d)
  bearing_cap.append(c)


# Array to store each compressed layer's width,
# compressed height, and force per cm2
compressed_layers = []


#--------------------------------------------------
# Compress the layers!
# Check each layer to see if it can support the
# load on it. If the force per cm2 is too great,
# then the layer collapses to its grain density,
# and all force is transferred down to the next
# deeper layer.
# But the force footprint grows a little every
# time, because of some interlocking between
# particles offf to the side. The angle of growth
# is apparently not very great with lunar regolith
# or you would be able to see little depressed areas
# around the Apollo footprints.
# When you reach a point where the soil's load
# bearing capacity is greater than or equal to
# the load per cm2, compression stops.
#--------------------------------------------------
total_compression        = 0.0
first_uncompressed_layer = 0
max_pressure             = 0
min_pressure             = 10000
support_footprint_edge   = wheel_width

for depth in range(max_depth) :
  downward_force = force_per_wheel / support_footprint_edge ** 2
  print ( f"depth {depth} downward_force == {downward_force}" )

  # Store the max and min pressure,
  # for later display purposes.
  if downward_force > max_pressure :
    max_pressure = downward_force
  if downward_force < min_pressure :
    min_pressure = downward_force
  support = bearing_cap [ depth ]
  print ( f"  support {support}" )

  # Is the support at this level greater than
  # or equal to the load per square cm?
  if support >= downward_force :
    # Yes! This layer does not compress,
    # so its thickness remains at 1.0 cm.
    first_uncompressed_layer = depth
    #compressed_layers.append( (support_footprint_edge, 1.0, downward_force) )
    print ( "  load is supported." )
    break

  # This layer cannot support the load,
  # so it collapses to grain density.
  pre_collapse_density = density [ depth ]
  new_layer_thickness = pre_collapse_density / grain_density
  total_compression += (1.0 - new_layer_thickness)
  print ( f"  new_layer_thickness == {new_layer_thickness}" )
  print ( f"  total_compression   == {total_compression}" )

  compressed_layers.append( (support_footprint_edge, new_layer_thickness, 
downward_force) )

  # The footprint increases at a shallow angle as
  # it goes down. So for each new cm of depth, the
  # previous support-square edge length increases by
  # twice the spread amount, which was calculated
  # above.
  support_footprint_edge += 2 * force_spread




#=======================================================
# render the compression image
#=======================================================
# Make the image.
image_height = 1000
image_width  = 2000
img = np.full((image_height, image_width, 3), 200, dtype=np.double)
pixels_per_cm = 50

# Draw a line for each cm of depth, starting at ground level
ground_level = 200

mpl.rcParams.update({'font.size': 5})
y = ground_level
first_uncompressed_layer_y = 0


#-------------------------------------------------------------
# Draw all the original layers of regolith,
# with darker gray values representing increasing density.
#-------------------------------------------------------------
for i in range(12) :
  layer_density = density[i]
  layer_bc      = bearing_cap[i]
  percent_density = layer_density / grain_density
  percent_density *= 0.75  # I just want the gray values to be lighter.
  print ( f"density {layer_density} {percent_density}" )
  gray = gray_value_for_depth(i)
  print ( f"gray: {gray}" )
  # Draw rect -------
  rect_start  = (y, 0)
  rect_height = pixels_per_cm
  rect_width  = image_width
  rect_extent = (rect_height, rect_width)
  rr, cc = rectangle(rect_start, extent=rect_extent, shape=img.shape)
  img[rr, cc] = (gray, gray, gray)
  if i == first_uncompressed_layer :
    first_uncompressed_layer_y = y
  # Label this rect ------------
  left_label  = f"depth {i+1}: {layer_density:.2f} g/cm3"
  right_label = f"bearing cap: {layer_bc:.2f} N/cm2"
  g = white
  if i < 4 :
    g = black
  mpl.text ( 300,  y + 30, left_label,  color=g )
  mpl.text ( 1400, y + 30, right_label, color=g )
  # Draw line -----
  rr, cc = line ( y, 0, y, image_width-1)
  img[rr, cc] = black
  # Advance the y value one cm lower
  y += pixels_per_cm


#-------------------------------------------------
# Draw the compressed layers, in red.
# The topmost layer starts at the total
# compression amount, and we work downward
# from there.
#-------------------------------------------------
pressure_range = max_pressure - min_pressure
y = first_uncompressed_layer_y - pixels_per_cm
n_compressed_layers = len(compressed_layers)
image_x_center = image_width / 2
first_layer_width = 0
first_layer_left_edge = 0
y = ground_level + total_compression * pixels_per_cm

for i in range(n_compressed_layers) :
  print ( f" i == {i}" )
  layer = compressed_layers[i]
  layer_width  = layer[0] * pixels_per_cm
  layer_height = layer[1] * pixels_per_cm
  print ( f"draw compressed layer {i} height {layer_height}" )
  # Draw rect -------
  layer_left_edge = image_x_center - layer_width / 2
  if i == 0 :
    first_layer_left_edge = layer_left_edge
    first_layer_width = layer_width
  rect_start  = (y, layer_left_edge)
  rect_extent = (layer_height, layer_width)
  rr, cc = rectangle(rect_start, extent=rect_extent, shape=img.shape)

  # The brightest red will be the highest pressure
  pressure_here = layer[2]
  label = f"{pressure_here:.2f} N/cm2"
  mpl.text ( image_x_center - 100,  y + 30, label,  color=g )
  pressure_percent = (pressure_here - min_pressure) / pressure_range
  img[rr, cc] = (pressure_percent, 0.0, 0.0)

  y += layer_height

# Draw a white line at the top of the first
# uncompressed layer, so it's easier to see.
line_y = first_uncompressed_layer_y
rr, cc = line ( line_y, 0, line_y, image_width-1)
img[rr, cc] = white

# Draw the wheel that is causing all
# this compression.
wheel_bottom = ground_level + total_compression * pixels_per_cm
rect_start  = (0, first_layer_left_edge)
rect_extent = (wheel_bottom, first_layer_width)
rr, cc = rectangle(rect_start, extent=rect_extent, shape=img.shape)
img[rr, cc] = wheel_color


#----------------------------------------------
# Write a bunch of information on the wheel.
#----------------------------------------------
labels = []
labels.append ( f"wheel of: {vehicle_name}" )
labels.append (  "   " )
labels.append ( f"vehicle mass: {mass_of_vehicle} kg" )
labels.append (  "   " )
labels.append ( f"force on wheel: {force_per_wheel} N" )
labels.append (  "   " )
labels.append ( f"sinks to {total_compression:.2f} cm" )

x       = first_layer_left_edge + 20
y       = 30
spacing = 30
for i in range(len(labels)) :
  mpl.text ( x,  y, labels[i],  color=white )
  y += spacing


#--------------------------------
# Save out the image
#--------------------------------
filename = './image.png'
mpl.imshow ( img )
mpl.axis ( 'off' )
mpl.savefig ( filename, bbox_inches='tight', dpi=400 )


# My last regosketch.    :-(
```