Isabelle Ho
Luis Dorfmann
Numerical Methods
17 December 2025

**Title:**
Moon Rover Wheel Coefficient

**Description:**
This program solves for the minimum viable value of static friction coefficient between a rover's wheels and the moon that allows the rover to surmount a hill whose profile is modeled by a user set function. The final value is dependent on a series of parameters that have both a default value and are settable by the user before running the script.

**Inputs:**
Separated by entity, here are the largely customizable inputs:

- Rover: Wheel diameter, friction coefficient, initial velocity, hill profile, hill distance, motor torque, force of gravity of the moon (CONST), weight
- Simulation: maximum timestep, minimum and maximum $\mu$

Note: gravitational force of the moon is set as a constant to maintain the use case of the program. A maximum hill size is not assumed to be large enough for the gravitational pull of the moon to vary with height.

**Outputs:**

- Calculates torque and net forces in order to determine if the rover can surmount the hill
    - Recurses the problem with a higher/different coefficient of friction if the rover is unable to make it up the hill
    - States the problem problem is not possible with the chosen inputs if $\mu_k = 1$
- Prints out all final calculated values
- Creates plots of the distance the rover was able to make it up the hill relative to the coefficient of friction of its wheels for various methods

**Appendix I, Simulation and Mathematical Details:**
The coefficient of friction is static because there is no slippage in the wheel at the moment it turns. The net force on the rover (positive directionally defined as moving up the hill) is given by the following equation:

$$F_{net} = F_{motor} - F_{drag} - F_{gravity}$$

The uphill motor force is limited by the coefficient of friction, as low contact/interactions with the hill leads to slippage and inefficient use of the motor. Thus, the MATLAB minimum function is used to determine the upwards force:

$$\min\left(\frac{\tau}{r}, \mu_s mg\cos\theta\right)$$

The drag force is defined:

$$F_D = C_D A\frac{\rho V^2}{2}$$

However, space is in a vacuum and therefore there is no fluid for the rover to be moving through. $\rho = 0$. Thus, drag force can be discounted and the surface area/relative size of the rover is irrelevant. The gravitational force downwards along the slope:

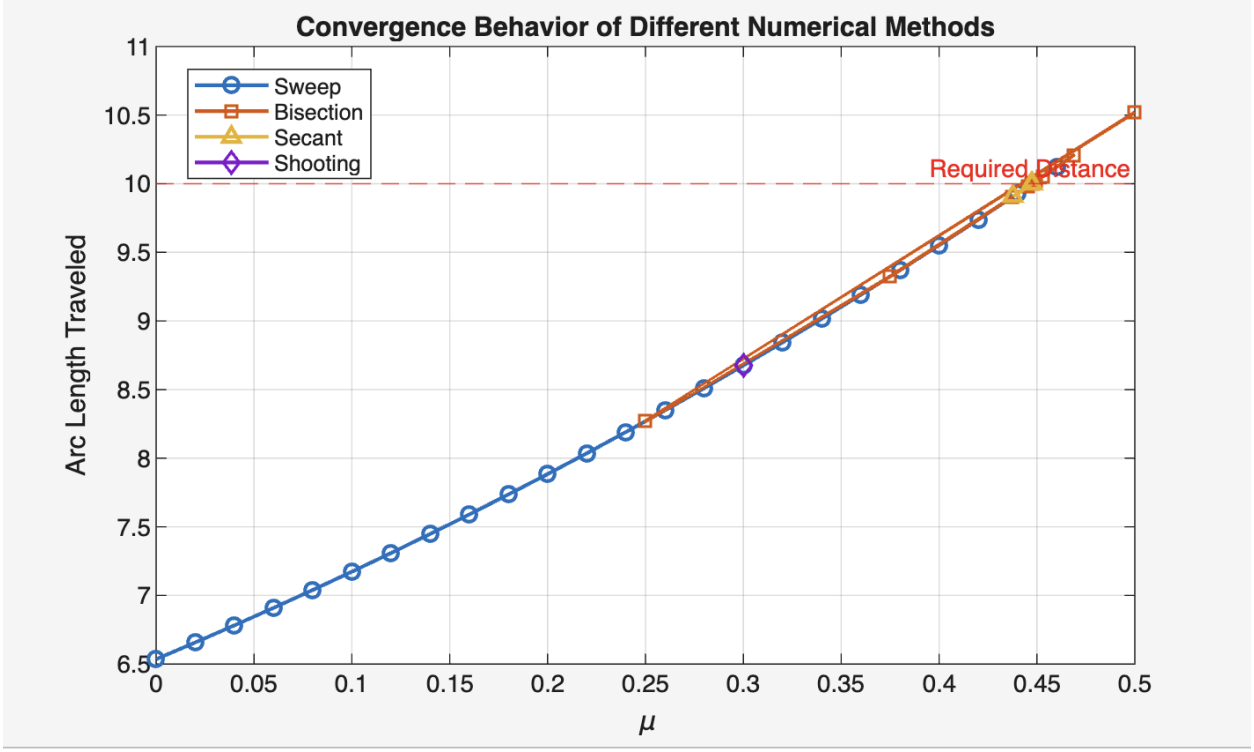
$$mg\sin\theta$$

$m$ and $\theta$ are user defined, and $g = 1.62$ m/s by Nasa's published research.

**Numerical Methods:**
Runge Kutta (Dormand–Prince) through ode45() MATLAB function
Bisection Method
Secant Method for root finding
Newton Style Shooting

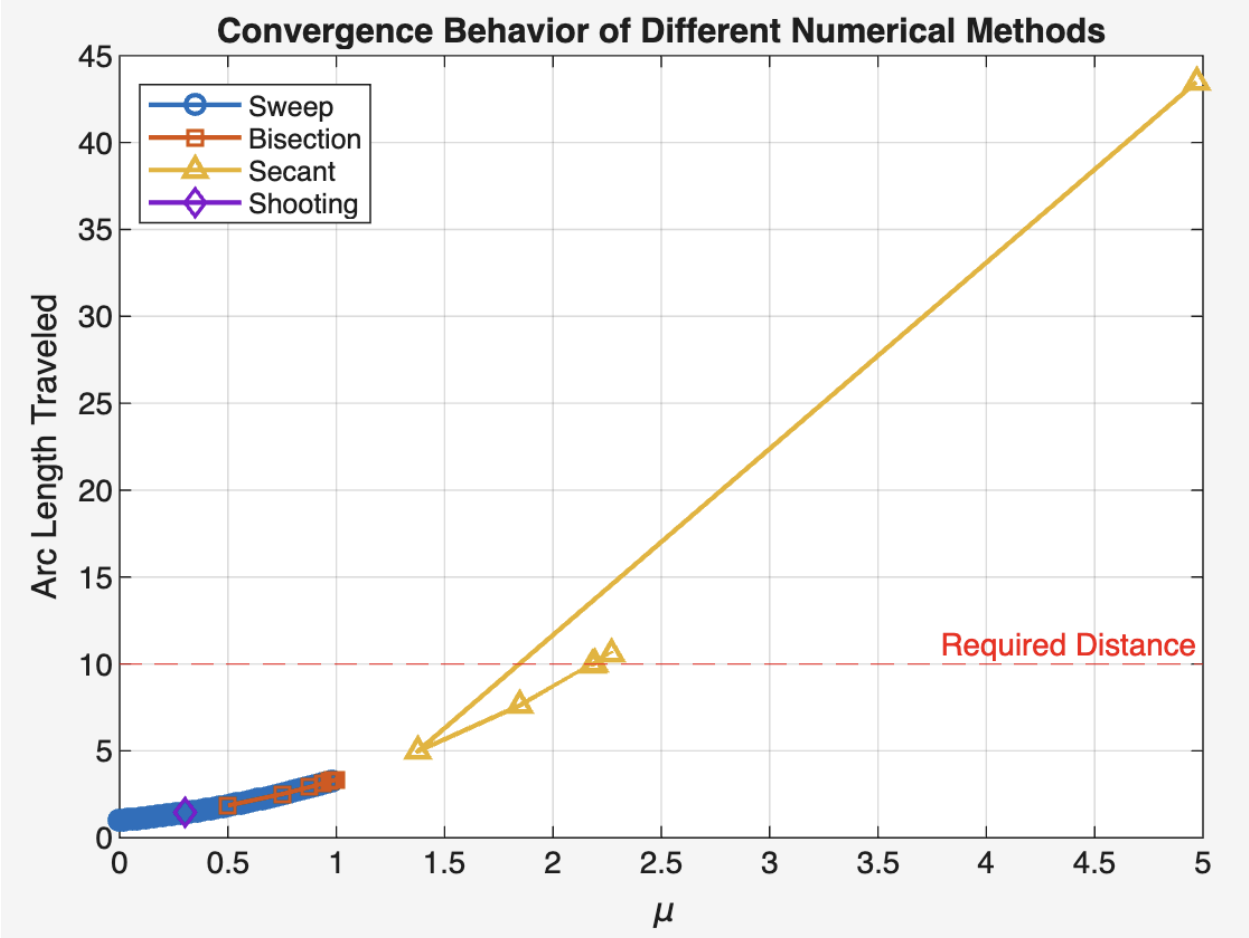
**Appendix II, Outputs:**

Case 1: All methods converge and a $\mu$ of 0.45 is estimated. Initial values: mass = 50 kg, radius = 0.1 meters, torque = 100 Nm, initial velocity = 3.5 meters per second, $\Delta\mu$ = 0.02, required distance = 10 meters, function = $0.3x^2$, derivative of function = $0.3x$




RESULTS:
Sweep Method: mu = 0.46000
Bisection: mu = 0.44745
Secant Method: mu = 0.44750
Shooting Method: mu = 0.30000

Case 2: Steeper slope and slower initial velocity lead to a $\mu$ outside of the target range. Attempts to converge shown. Initial values: mass = 50 kg, radius = 0.1 meters, torque = 100 Nm, initial velocity = 1.2 meters per second, $\Delta\mu$ = 0.02, required distance = 10 meters, function = $0.6x^2$, derivative of function = $1.2x$




Sweep method did not converge
Bisection method did not converge
Shooting method did not converge

RESULTS:
Sweep Method: mu = 0.98000
Bisection: mu = 0.99994
Secant Method: mu = 2.19081
Shooting Method: mu = 0.30000

The best guesses for the sweep and bisection method are near one, as that was the set upper limit for $\mu$. The secant method is open boundary, allowing it to solve for an accurate $\mu$.

**Appendix III, Code:**

Ho.m

```
clear; clc;

% SET AND INITIAL VALUES
params.g  = 1.62; % gravitational force of the moon
tol = 1e-4;       % tolerance for secant and shooting methods
maxIter = 50;     % maximum iterations for secant and shooting methods

% USER ADJUSTABLE PARAMETERS
params.m  = 50;   % rover mass in kilograms
params.r  = 0.1;  % wheel radius in meters
params.Tm = 100;  % motor torque in newton meters
params.v0 = 1.2;  % intial velocity before the hill
dmu = 0.02;       % delta mu for ode45()
params.s_required = 10;       % required hill distance for a successful run
params.yfun = @(x) 0.6*x.^2;  % function profile of the hill
params.dydx = @(x) 1.2*x;     % derivative of the function profile of the hill

% CALL FUNCTIONS
[mus, mu_s_log, s_s_log]   = sweep(params, dmu);
[mub, mu_b_log, s_b_log]   =  bisection(params, tol);
[muse, mu_se_log, s_se_log] = secant(0.1, 0.5, params, tol, maxIter);
[mush, mu_sh_log, s_sh_log] = shooting(0.3, params, tol, maxIter, dmu);

% PLOT RESULTS
figure;
plot(mu_s_log, s_s_log, 'o-', 'LineWidth', 1.5);
hold on;
plot(mu_b_log,   s_b_log,   's-', 'LineWidth', 1.5);
plot(mu_se_log,   s_se_log,   '^-', 'LineWidth', 1.5);
plot(mu_sh_log, s_sh_log, 'd-', 'LineWidth', 1.5);
yline(params.s_required,'--r','Required Distance');
xlabel('\mu');
ylabel('Arc Length Traveled');
title('Convergence Behavior of Different Numerical Methods');
legend('Sweep','Bisection','Secant','Shooting','Location','best');
grid on;

% PRINT RESULTS
fprintf('\nRESULTS:\n');
fprintf('Sweep Method:    mu = %.5f\n', mu_s_log(end));
fprintf('Bisection:       mu = %.5f\n', mu_b_log(end));
fprintf('Secant Method:   mu = %.5f\n', mu_se_log(end));
fprintf('Shooting Method: mu = %.5f\n', mu_sh_log(end));
```

simulateRover.m

```
function s_end = simulateRover(mu, params)
   params.mu = mu; % set mu parameter if fed directly into function

   y0 = [0; params.v0; 0];   % horizontal position, velocity, arc distance
travelled
   tspan = [0 100];

   options = odeset('Events', @(t,y) stopEvents(t,y,params));

   [t,y] = ode45(@(t,y) roverODE(t,y,params), tspan, y0, options);

   s_end = y(end,3);
end
```

---

roverODE.m

```
function dydt = roverODE(~, y, params)
x = y(1);   % horizontal position
v = y(2);   % velocity along slope
s = y(3);   % arc length
yp = params.dydx(x);
theta = atan(yp); % angle of slope

% FORCES
N = params.m * params.g * cos(theta);                  % normal force
F_drive = min(params.Tm / params.r, params.mu * N);    % motor force

% ACCELERATION
dvdt = (F_drive - params.m * params.g * sin(theta)) / params.m;

% POSITION
dxdt = v * cos(theta);    % horizontal position
dsdt = v;                 % arc position
dydt = [dxdt; dvdt; dsdt];
end
```

---

stopEvents.m

```
function [value, isterminal, direction] = stopEvents(~, y, params)
   s = y(1); % pull arc position from array
   v = y(2); % pull velocity from array

   value = [v - 1e-4; params.s_required - s];
   isterminal = [1; 1];  % terminate if speed = 0
   direction = [-1; -1]; % terminate if reaches required distance
end
```

---

distanceResidual.m

```
function res = distanceResidual(mu, params)
   % DETERMINES IF THE ROVER MADE IT FAR ENOUGH TO CONSIDER A SUCCESS
   s_end = simulateRover(mu, params);
   res = s_end - params.s_required;
end
```

---

sweep.m

```
function [mu, mu_s_log, s_s_log] = sweep(params, dmu)
   % INITIALIZE MU, PLOT ARRAYS
   mu = 0;
   mu_s_log = [];
   s_s_log  = [];
   while mu <= 1
       s_end = simulateRover(mu, params);
       mu_s_log(end+1) = mu;
       s_s_log(end+1)  = s_end;
       if s_end >= params.s_required % exit out and return mu if appropriate
distance
           return
       end
       mu = mu + dmu; % iterate mu by increasing it by user defined amount
   end
   disp('Sweep method did not converge')
end
```

---

bisection.m

```
function [mu, mu_b_log, s_b_log] = bisection(params, tol)
mu_low = 0;
mu_high = 1;
mu_b_log = [];
s_b_log  = [];
if distanceResidual(mu_high, params) < 0
   disp('Bisection method did not converge');
end
while (mu_high - mu_low) > tol
   mu_mid = (mu_low + mu_high)/2;
   s_mid = simulateRover(mu_mid, params);
   mu_b_log(end+1) = mu_mid;
   s_b_log(end+1)  = s_mid;
   if distanceResidual(mu_mid, params) >= 0 % pick the upper half to
       % bisect if mu is greater than the midpint
       mu_high = mu_mid;
   else
       mu_low = mu_mid; % % pick the lower half to bisect if mu is less
       % than the midpint
```

```
    end
end
mu = mu_high;
end
```

---

secant.m

```
function [mu, mu_se_log, s_se_log] = secant(mu0, mu1, params, tol, maxIter)
% EMPTY ARRAYS FOR LATER PLOT
mu_se_log = [];
s_se_log  = [];
f0 = distanceResidual(mu0, params);
f1 = distanceResidual(mu1, params);
for k = 1:maxIter
    mu2 = mu1 - f1*(mu1 - mu0)/(f1 - f0);

    % POPULATES PLOT ARRAYS
    s2 = simulateRover(mu2, params);
    mu_se_log(end+1) = mu2;
    s_se_log(end+1)  = s2;

    % RECURSES UNTIL TOLERANCE IS MET
    if abs(mu2 - mu1) < tol
        mu = mu2;
        return
    end
    mu0 = mu1;
    f0  = f1;
    mu1 = mu2;
    f1  = distanceResidual(mu1, params);
end
disp('Secant method did not converge'); % display message if able to exit
% out of loop without finding solution
end
```

---

shooting.m

```
function [mu, mu_sh_log, s_sh_log] = shooting(mu_guess, params, tol, maxIter,
dmu)
    mu_sh_log = [];
    s_sh_log  = [];    % empty array for plotting
    mu = mu_guess; % intial shot
    s_end = simulateRover(mu, params);
    mu_sh_log(end+1) = mu; % populate array with mu/distance travelable values
    s_sh_log(end+1)  = s_end;

    for k = 1:maxIter
        res = distanceResidual(mu, params);
```

```
        if abs(res) < tol
            return
        end

        % FINITE DIFFERENCE SLOPE
        res2 = distanceResidual(mu + dmu, params);
        dres_dmu = (res2 - res)/dmu;

        % NEWTON UPDATING
        mu = mu - res/dres_dmu;

        % CLAMPING
        mu = max(0, min(mu, 1));
    end
    disp('Shooting method did not converge');
end
```

**Validation:**
The model's estimation seems sound. It varies as expected: more difficult terrain requires a higher coefficient of friction in order to better utilize the force from the engine. The model displays the error ranges in computation between the various numerical methods and why bisection may be best for an undefined $\mu$ range. In practice, a $\mu$ value of over 2 is not practical; instead, one could aim for higher initial velocity or attempt to circumvent the slope by winding up it. Real life estimations of friction coefficients range from 0.1 to 0.7, making 0.45 within a reasonable range. 1 as an upper bound for $\mu$ is likely overambitious, as it is difficult to maintain traction with the moon's surface.

**References:**


Dorfmann, Luis. "Various Canvas Materials." Fa25-ES-0055-01-Numerical Methods Dec. 2025, Tufts University, Medford, Canvas, https://canvas.tufts.edu/courses/67228.

National Aeronautics and Space Administration. *Lunar Surface Data Book: Baseline*. 2025. NASA Technical Reports Server, ntrs.nasa.gov.

National Aeronautics and Space Administration. *Natural Environmental Design Criteria Guidelines for Use in Aerospace Vehicle Development*. NASA, 1969.

Wong, J. Y. *Theory of Ground Vehicles*. 4th ed., John Wiley & Sons, 2008.

Hall, Nancy. "Moon | Glenn Research Center | NASA." Glenn Research Center | NASA, 20 Nov. 2023, www1.grc.nasa.gov/beginners-guide-to-aeronautics/moon/.

ChatGPT, 17 December 2025. version 5.1, OpenAI, 9 Dec. 2025, chat.openai.com/chat.

Papakonstantinou, Joanna M, and Richard A Tapia. "Origin and Evolution of the Secant Method in One Dimension." American Mathematical Monthly, vol. 120, no. 6, 1 Jan. 2013, pp. 500–500, https://doi.org/10.4169/amer.math.monthly.120.06.500. Cited through AI Overview.