car_model:

Class: car()

Arguments: none

Methods:

curve_accel(v, r, transmission_gear='optimal')

Returns the maximum possible forward acceleration in g's that the vehicle can provide while traveling on a curve of given radius at a given speed.

Arguments:

v – (float) *the velocity of the vehicle (in/s)*

r – (float) *radius of curve the vehicle is travelling along in inches*

transmission_gear – (int) *set as 'optimal' by default, this will automatically assume the car is utilizing whichever transmission gear maximizes the power delivered to the axle. May also be set as an integer between 0 and 5 inclusive to select a different gear with 0 being the largest gear and 5 being the smallest gear.*

curve_brake(v, r)

Returns the maximum possible braking deceleration in g's that the vehicle can provide while traveling on a curve of given radius at a given speed.

Arguments:

v – (float) *the velocity of the vehicle (in/s)*

r – (float) *radius of curve the vehicle is travelling in inches*

curve_idle(v)

Returns the deceleration due to drag in g's for the vehicle traveling at a given speed

Arguments: **v** – *Velocity of vehicle in in/s*

adjust_weight(w)

Alters the weight of the car to a specified value. Accordingly adjusts all weight sensitive values.

Arguments: **w** – The new weight of the car in lb

adjust_height(h)

Alters the height of the center of mass of the car to a specified value. Accordingly adjusts all additional values sensitive to center of mass height

Arguments: **h** – *The new height of the car's center of mass in inches*

traction_curve()

Plots a friction ellipse for the entire vehicle. This displays the max forward and braking accelerations for every possible cornering acceleration the car can handle

Arguments: None

tire_model

Class: tire(self, cornering_data_file, acceleration_data_file = None)

Arguments:

cornering_data_file – *Raw cornering data from TTC database*

acceleration_data_file – *Raw acceleration data from TTC database. Set to None by default. Must input TTC data to analyze acceleration in x direction or to use tire object for lapsim.*

Methods:

traction(set_type, load, camber)

Returns either max possible cornering force or max longitudinal force of tires in lb given applied load and camber.

Arguments:

set_type – (string) *Set as 'corner' or 'cornering' to return max cornering force from tire. Set as 'accel' or 'acceleration' to return max longitudinal force from tire*

load – (float) *Applied vertical load on tire*

camber – (float) *inclination* angle/camber of tire

lateral_force_plot()

Generates a plot displaying *the maximum lateral force which may be provided by the tire at different cambers and applied loads*

Arguments: None

lateral_coeff_plot()

Generates a plot displaying coefficient of friction in the lateral direction at different cambers and applied loads

Arguments: None

SA_FY_plot(camber)

Generates a plot of slip angle vs lateral force curves across all the different applied loads which were tested in the TTC data at a specified camber

Arguments:

> ***camber*** *– either 0, 2, or 4. Represents camber/inclination angle of tire*

SA_MZ_plot(camber)

Generates a plot of slip angle vs aligning torque curves across all the different applied loads which were tested in the TTC data at a specified camber

Arguments:

> **camber** – (int) *either 0, 2, or 4. Represents camber/inclination angle of tire*

axial_force_plot()

Generates a plot displaying the maximum axial force which may be provided by the tire at different cambers and applied loads

Arguments: None

axial_coeff_plot()

Generates a plot displaying coefficient of friction in the axial direction at different cambers and applied loads

Arguments: None

SR_FX_plot(camber)

Generates a plot of slip ratio vs axial force curves across all the different applied loads which were tested in the TTC data at a specified camber

Arguments:

camber – (int) *either* 0, 2, *or* 4. *Represents camber/inclination angle of tire*

spline_track:

class: track(p1x, p1y, p2x, p2y)

Generates a track from a series of gates. Gates are defined as a set of two points the car must travel between. The initial track is typically not very good but can be refined using the adjust() *method.*

Variables:

All variables are only defined after the lapsim is run using the track with either the run_sim() *or* plt_sim() *methods.*

t – (float) *The total travel time of the car from the last sim which was run*

nds – (list) *The nodespace of the last sim which was run. Contains a list of values. This nodespace is comprised of a list of floats where each float represents a location on the track. Each index of the list corresponds to a specific node. The value of the list at that index represents how far the car must travel from the start of the track to reach that node.*

v3 – (list) *The velocities of the car at each individual node within the node space.*

Arguments:

p1x – *A list of x coordinates for the first point of each gate*

p1y – *A list of y coordinates for the first point of each gate*

p2x – *A list of x coordinates for the second point of each gate*

p2y - *A list of y coordinates for the second point of each gate*

Methods:

adjust_track(itterations, step)

An iterative function that slightly improves the track with each iteration.

Arguments:

itterations – *the number of iterations the function will run*

step – *Increasing step increases the amount the track will be altered with each iteration. Increasing this value will allow the track to be adjusted faster but will also reduce the precision of the adjustments being made. When the track is far from optimal and hasn't been adjusted much, its typically better to start with a high step value and decrease to a lower value as the track becomes more refined*

Additional Notes: *Running this method for 40 iterations with a step size of 100, and then again for 30 iterations with a step size of 30, and finally for another 30 iterations with a step size of 10 is typically enough to get near ideal track.*

run_sim(car, nodes=5000, start=0, end=0)

Runs a lapsim simulation for the track. Updates nds, t, and v3 variables in accordance to simulation results

Arguments:

car – *any car object from the* car_model *program*

nodes – (int) *set to* 5000 *by default. The number of nodes the track is discretized into for the lapsim simulation. More nodes allow for greater simulation precision but requires longer computational time*

start – (int) *the starting curve of the simulation. Set to 0 by default. Set to zero to run sim from beginning of track. Set to 1 to start at end of first curve. Set to 2 to start at end of second curve and so on.*

end – (int) *the final curve of the simulation. Setting to 5 will end the end of the track before the fifth curve. Setting to 6 will set the end of the track before the sixth curve and so on. Set to zero by default. This will set the end of the track to be after the final curve.*

plt_sim(car, nodes=5000, start=0, end=0)

Exactly the same as run_sim() *but also displays a position vs velocity plot after running the sim.*

Arguments:

car – *any car object from the* car_model *program*

nodes – (int) *set to* 5000 *by default. The number of nodes the track is discretized into for the lapsim simulation. More nodes allow for greater simulation precision and accuracy but requires longer computational time*

start – (int) *the starting curve of the simulation. Set to 0 by default. Set to zero to run sim from beginning of track. Set to 1 to start at end of first curve. Set to 2 to start at end of second curve and so on.*

end – (int) *the final curve of the simulation. Setting to 5 will set the end of the track before the fifth curve. Setting to 6 will set the end of the track before the sixth curve and so on. Set to zero by default. This will set the end of the track to be after the final curve.*

Lapsim:

Class: four_wheel(t_len_tot, t_rad, car, n)

A four-wheel-model simulation of a vehicle traveling across a track as fast as possible. Tracks are defined as a series of arcs with constant radii with each arc defined as a radius and an arc length.

Arguments:

t_len_tot – *A list of float values containing the lengths of each constant radius arc that comprises the track. Index 0 corresponds to the first arc, index 1 corresponds to the second arc and so on.*

t_rad – *A list of float values containing the radii of each constant radius arc that comprises the track. Index 0 corresponds to the first arc, index 1 corresponds to the second arc and so on.*

car – *must be a car object from the car_model program*

n – (int) *the number of nodes used to discretize the track for the lapsim. Increasing this value will yield higher precision and accuracy for simulation results but will also increase computational time.*

Methods:

run()

Runs the sim.

Arguments: None

Returns:

t – (float) *The total travel time of the car*

nds – (list) *The nodespace developed to discretize the track. This nodespace is comprised of a list of floats where each float represents a location on the track. Each index of the list corresponds to a specific node. The value of the list at that index represents how far along the track, in ft, the node is located.*

v3 – (list) *The velocities of the car at each individual node within the node space. Creating an nds vs v3 plot will result in a velocity vs distance traveled plot displaying the vehicle's speed at different parts of the track*

Drivetrain Model

Class: drivetrain(final_drive = 4.8)

Models the drivetrain of the vehicle

Arguments:

final_drive – (float) *set to 4.8 by default. Final drive of drivetrain. This represents the ratio between the gear reduction ratio across the two sprockets connecting the transmission and the rear axle*

Methods:

get_engn_pwr(rpm)

returns the power provided by the engine in horsepower at a specified rpm.

Arguments: **rpm –** (float) *rpm of the engine*

get_engn_T(rpm)

returns the torque provided by the engine in ft-lb at a specified rpm.

Arguments: **rpm –** (float) *rpm of the engine*

get_F_accel(self, mph, gear='optimal')

Returns the maximum possible acceleratory force the engine can provide at a specified travel speed and transmission gear.

Arguments:

mph – (float) travel speed of vehicle in

gear – (int) *set as 'optimal' by default, this will automatically assume the car is utilizing whichever transmission gear maximizes the power delivered to the axle. May also be set as an integer between 0 and 5 inclusive to*

select a different gear with 0 being the largest gear and 5 being the smallest gear.