Individual Project Report

Hayden Supple

020: Team 02

ENGR 13300

William C. Oakes

4 December 2024

1. Project Introduction

1807, French brothers Claude and Nicéphore Niépce invented the internal combustion engine, a mechanical operative that uses the atomization of fuel, the mechanical compression of a piston, and an electrical spark to complete a combustion reaction, turning chemical potential energy into a useable kinetic energy (photo-museum). Over years, engine modifications and alterations have been made to add more pistons, more compression, faster combustion and others to complete work such as transportating, field working, and reaching the skies more efficiently and powerfully, completing tasks that would otherwise be difficult for humans to complete.

A dynamometer is a modern tool used to measure the outputted power of an engine given an input of crankshaft rotation (Setra). This tool uses a physical resistant force against the rotation to record the resistance created at different revolution speeds (RPM) to measure torque. This value is then inputted in the equation:

$$\text{Horsepower} = (\text{Torque} \times \text{RPM}) / 5252$$

This equation takes in Torque in lb-ft multiplied by RPMs in Revolutions / Minutes and divided by 5252, the constant to convert the units to horsepower, 550 lb•ft/s (powertestdyno).

My project is a simulation of a dynamometer. Using inputs from the user of engine variables, the program calculates intended gas octane as well as a simulated injector value map, which returns a color-coded map of injector values in grams of gasoline based on engine load and RPM. This program will allow a user to best understand the base horsepower and torque their engine should make.

2. Project Overview of Inputs and Outputs

The program begins with requesting a bore width (Cylinder Diameter) in millimeters which much be less than 200 mm and greater than 60mm, based on real minimums and maximums in produced engines. It then requests for a stroke length (Diameter of Crankshaft) in millimeters which much result in a Stroke:Bore ratio greater than 0.5 and less than 1.5, also based on real magnitudes in produced engines. These measurements will allow for the computation of the volume when the engine is in bottom dead center, or then lowest position for the piston.

Next, a piston type is requested (flat, flat-top with reliefs, hemi), which will further calculate the volume. Reliefs will result in a greater volume by a third the diameter of the valves, which is calculated by a rule of thumb:

$$\text{Valve Diameter} = \sqrt{((6000 \times \text{Stroke} \times \text{Bore}^2)/2286000)}$$

(BCSTValve)

Next, a compression ratio will be requested which much be greater than 5:1 and less than 15:1, based on produced engines. Lastly, the program asks for a number of cylinders.

Each request has its own error messages for upper and lower limits. A Warning message is automatically outputted to ensure the user recognizes simulatory limits and recognizes assumed constants within the experiment. After inputs, the program outputs the octane requirement for the engine and proposed compression ratio. The maximum RPM is outputted as it corresponds to the cylinder amount.

An output to explain the simulation is expressed.

After computations are completed, an excel sheet will be a result with color-coding based on the magnitude of injector values.

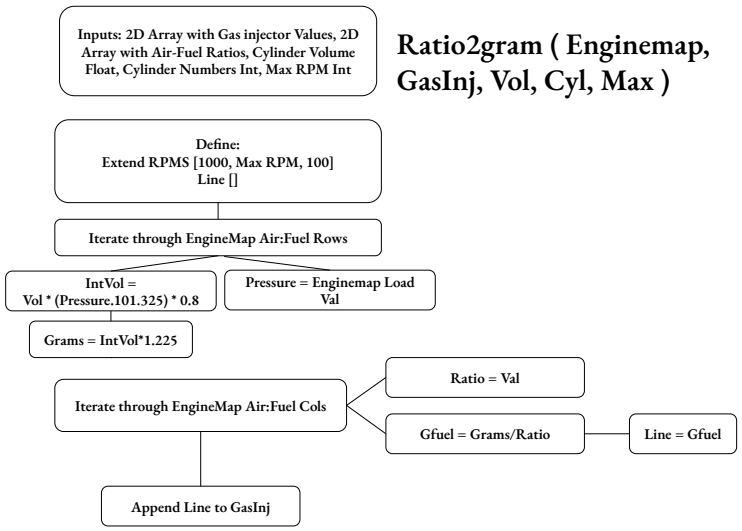
The result of the simulation will be a graph. This figure will represent two lines, one which is dashed and pink, Torque, and one solid and red, HP. The Y-Axis will represent Horsepower on the left and Torque in ft•lb on the right. Engine RPMs will be the x axis and the maximums will be outputted in the top left.

3. User-defined Functions

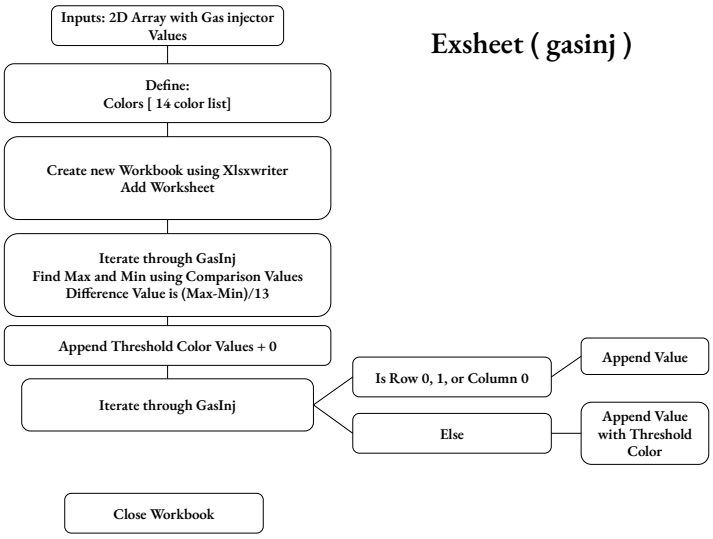
Within my program, I have three user defined functions: Ratio2gram, Exsheet, and EngineMapSim, an importable function from DynamometerSim.py.

My first program UDF is Ratio2gram, a function that imports data of enginemap, a preset 2D array of air-fuel ratios based on engine load and RPM, gasinj, another 2D array that is set up to add gasoline values in grams per load and RPM, vol, a float representing the volume of the engine, cyl, an int of the number of cylinders, and max, an integer representing the maximum RPM.

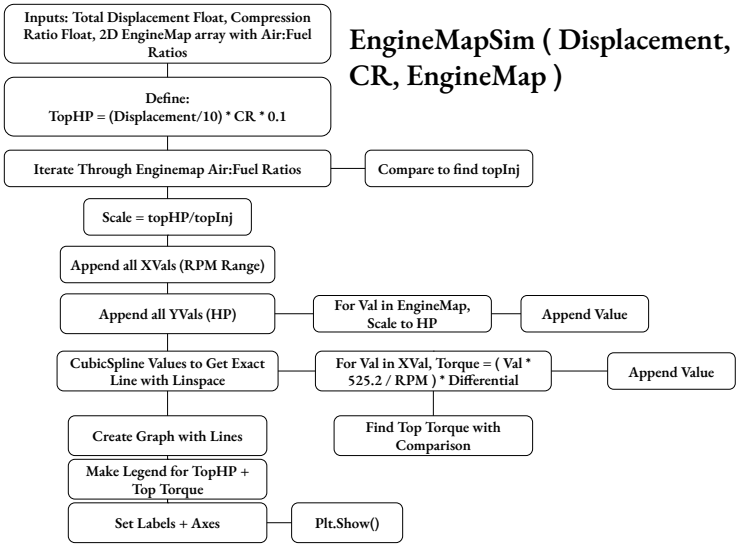
Ratio2gram takes these numbers and, while cycling through enginemap to compare air-fuel ratios, calculates the grams of gasoline needed to efficiently conduct combustion. The output of this is the updated array of gasinj, displaying gasoline values needed in grams.



My second UDF is Exsheet, a function that imports gasinj, an array that is occupied with gasoline in grams by the load and RPM. Exsheet defines an array of 14 stages of colors as well as a minimum and maximum value from gasinj. Following this, exsheet uses xlsxwriter, a python library that may relate python values to excel sheets, to write the entire contents of gasinj to an excel sheet, color-coding the values based on the magnitude of the number. The output is the excel sheet.



My third UDF is EngineMapSim, a function within python file DynamometerSim.py that imports displacement, a float of the volume in cubic inches of the engine, CR, the compression ratio, and gasinj, the values of gasoline injected as used to compare for magnitude. This function takes these values and using the trend of the gasinj values at top load, calculates and estimates a curve for horsepower for the specific vehicle. These values are then used with equation 1 above to calculate and graph torque. Enginemapsim outputs a graph relating horsepower and torque to RPM along with the maximum values of the trial.



4. References

Lish, Tom. “What Is a Dynamometer and How Does It Work?” *Premium Sensing Solutions*, www.setra.com/blog/test-and-measurement-dynamometer. Accessed 5 Dec. 2024.

“The Pyréolophore: A New Engine Principle.” *Nicéphore Niépce’s House Museum*, 7 Dec. 2022, photo-museum.org/the-pyreolophore-a-new-engine-principle/.

Studer, Peter. “How to Calculate Horsepower.” *Power Test, LLC*, 18 Jan. 2024, powertestdyno.com/how-to-calculate-horsepower/.

Valves, BSCT. “How to Calculate and Select the Diameter of the Control Valve?” *Control Valve & on off Valve ,Solenoid-Valve Manufacturer*, 11 May 2022, bcstvalve.com/the-diameter-of-the-control-valve/.

5. Appendix

1. User manual

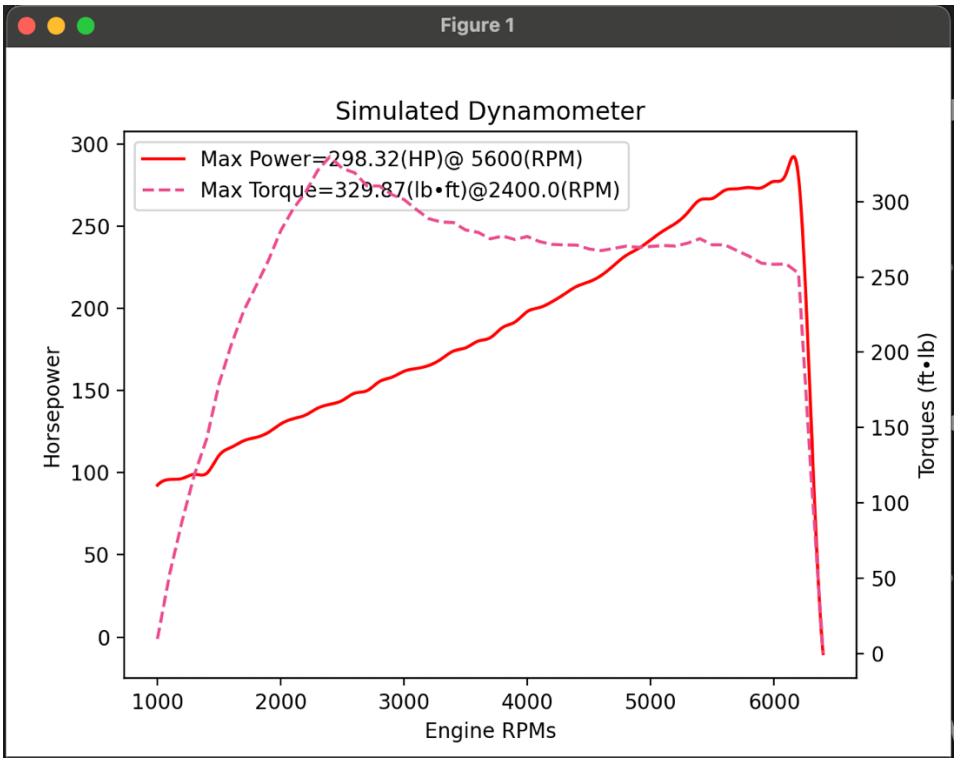
Program will begin by requesting for a bore size in millimeters that must be between 200 and 60, which is to be typed in by the user. Program will then request for a stroke size in millimeters, which, when calculated as the divisor under the bore size, must be between 0.5 and 1.5 inclusive. A piston type will be requested which is to be from the allowed styles. A compression ratio between 5 and 5 exclusive will be asked to be inputted from the user and finally a cylinder number allotment must be entered.

Program will output required RPM Maximum, octane of gasoline, and all precautionary warnings about the accuracy of the data. Output of an xlsx Excel spreadsheet with color coding based on injector value magnitude will be resulted from calculations up to maximum RPM.

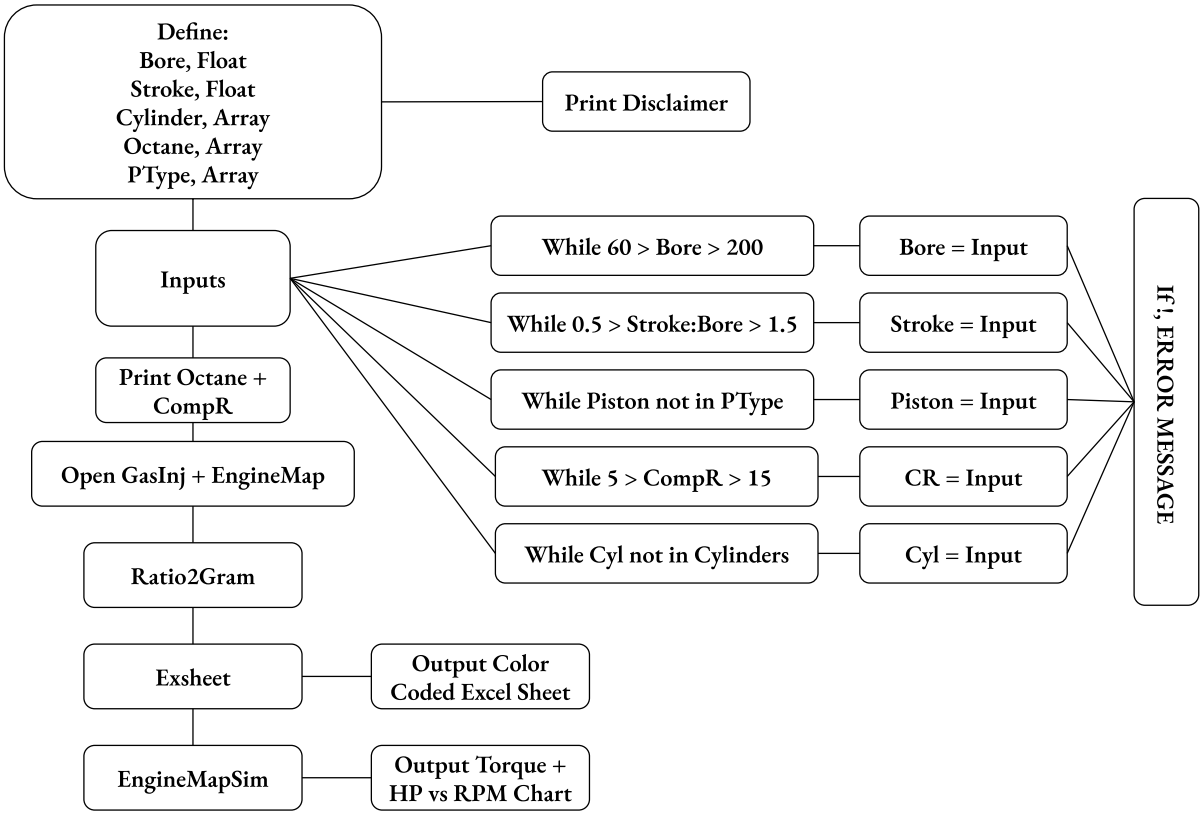
Program will begin a simulation and output a graph displaying RPM on the X Axis, HP with a solid line on the left Y Axis, and Torque in Lb•Ft on the right Y Axis. Below will be a sample input from a 2003 BMW 540i with a 4.4 Liter 8 Cylinder engine rated for 282 HP and 325 Lb•Ft of torque.

```
ds/FinalProjectENGR133.py
**Warning: Values displayed in simulation follow optimal conditions of a 14.7:1 AFR
with no consideration of airflow to engine. Assume optimal boost pressures and forced
induction**
Enter Bore Width (mm): 92
Enter Stroke Length (mm): 82.7
Enter Piston Type (flat, flat-top with reliefs, hemi): flat-top with reliefs
Enter Your Desired Compression Ratio (Format as X in X:1) 10
**Engine will use 92 octane gasoline**
**Engine has a Compression Ratio of 10.0:1**
Enter the amount of Cylinders: 8
**Maximum engine speed determined to be 6200 RPM**
**Simulation will Report Base Horsepower (Hp) and Torque (Ft-Lbs)**
2024-12-05 15:36:50.424 python[20351:788292] +[IMKClient subclass]: chose IMKClient_Modern
2024-12-05 15:36:50.424 python[20351:788292] +[IMKInputSession subclass]: chose IMKInputSession
_Modern
```

RPMS														
Gasoline (g	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000	2100	2200	2
50	0.1462	0.1462	0.1462	0.1462	0.1492	0.1514	0.1524	0.1535	0.1546	0.1557	0.1569	0.158	0.1592	0
58	0.1696	0.1696	0.1696	0.1696	0.1731	0.1768	0.1781	0.1794	0.1807	0.182	0.1833	0.1847	0.186	0
66	0.193	0.193	0.193	0.1957	0.197	0.2026	0.2041	0.2056	0.2071	0.2086	0.2101	0.2117	0.2133	0
74	0.2164	0.2164	0.2164	0.2194	0.2209	0.2288	0.2305	0.2322	0.2339	0.2356	0.2374	0.2392	0.241	0
82	0.2398	0.2398	0.2398	0.2431	0.2448	0.2554	0.2573	0.2592	0.2611	0.263	0.265	0.267	0.2691	0
90	0.2632	0.2632	0.2632	0.2668	0.2705	0.2824	0.2845	0.2866	0.2887	0.2909	0.2931	0.2953	0.2976	0
98	0.2866	0.2866	0.2866	0.2905	0.2946	0.3097	0.312	0.3144	0.3167	0.3191	0.3216	0.324	0.3265	0
106	0.31	0.31	0.3142	0.3164	0.3186	0.3426	0.3452	0.3452	0.3452	0.3478	0.3505	0.3532	0.356	0
114	0.3333	0.3356	0.3379	0.3403	0.3427	0.3684	0.3684	0.3712	0.3741	0.3769	0.3799	0.3828	0.3858	0
122	0.3567	0.3592	0.3617	0.3642	0.3667	0.3943	0.3973	0.4003	0.4034	0.4065	0.4097	0.4129	0.4162	0



2. Flowchart



3. Code

FinalProjectENGR133.py

```
from DynamometerSim import enginemapsim

import math
import numpy as np
import xlsxwriter as excel

def exsheet(gasinj):
   #Define all variables. Colors will be reference array for excel sheet.
   colors = ['#bdeb34','#d9fa05','#fafa05','#ffec1c','#ffca1c','#eda011', '#ed7f11', '#ed4c11',
'#c94210', '#fc5603', '#a31212','#820c0c', '#7d0b02']
   thresholds = []
   min = 0
```

```
max = 0
compmax = 0
compmin = 100
#Open Workbook
workbook = excel.Workbook("/Users/haydensupple/Downloads/GasInjValues.xlsx")
worksheet = workbook.add_worksheet()
#Iterate through gasinj to gain max and min values
for i in range(2,len(gasinj)):
   for j in range(1,len(gasinj[i])):
      if float(gasinj[i][j]) < compmin:
         min = float(gasinj[i][j])
         compmin = float(gasinj[i][j])
      elif float(gasinj[i][j]) > compmax:
         max = float(gasinj[i][j])
         compmax = float(gasinj[i][j])
#Threshold values will be every iteration of the range/13
val = (max-min)/13
#Append 0 for minimum
thresholds.append(0)
#Create thresholds based on the minimum + val iterated
for i in range(1,14):
   thresholds.append(min+(i*val))

#Iterate through gasinj and append to worksheet
for i in range(len(gasinj)):
   for j in range(len(gasinj[i])):
      value = gasinj[i][j]
      if i != 0 and i != 1 and j != 0:
         color = None
         for o in range(1, 14):
            if thresholds[o-1] <= float(value) <= thresholds[o]:
               color = colors[o-1]
               break
         if color:
            #If its a value for gasoline, format box to be a color from threshold
            cf = workbook.add_format({'bg_color': color})
            worksheet.write(i, j, value, cf)
         else:
            #If no color, append value
            worksheet.write(i, j, value)
```

```
            else:
                worksheet.write(i, j, value)
        workbook.close()

    def ratio2gram(enginemap, gasinj, vol, cyl, max):
        #Define volume of displacement per cylinder * cylinder
        vol = cyl*vol
        RPMS = [1000]
        #Create RPM range up to max
        RPMS.extend(range(1100, max+100, 100))

        line = []

        #Iterate through range of enginemap
        for i in range(2, len(enginemap)):
            #Get load variable from column 0
            pressure = int(enginemap[i][0])
            #Calculate Intake Volume
            intvol = vol*(pressure/101.325)*0.8

            #Calculate grams of gasoline per ratio
            grams = intvol*1.225
            line = []
            #Iterate through the range of enginemap to calculate the Air:fuel Ratio, grams of fuel, and
    append to a line
            for j in range(1, len(enginemap[i])):
                ratio = float(enginemap[i][j])
                gfuel = grams/ratio
                line = np.append(line, str(round(gfuel,4)))
            #Append line to gasinj
            for num in line:
                gasinj[i] = np.append(gasinj[i], num)
        for i in range(1, len(gasinj)):
            gasinj[i] = gasinj[i][:len(gasinj[1])]

    def main():
        #Define all variables
        bore = 300
        stroke = 350
        piston = ""
```

```
    ph = 0
    cr = 0
    cyl = 0
    #Define list for cylinders, piston style and octane
    cylinders = [4,6,8,10,12]
    octane = [5,72,6,72,7,80,8,85,9,87,10,92,11,92,12,96,13,100,14,108,15,112]
    pstyles = ["flat", "flat-top with reliefs", "dish", "hemi"]
    print("**Warning: Values displayed in simulation follow optimal conditions of a 14.7:1 AFR
\nwith no consideration of airflow to engine. Assume optimal boost pressures and forced
\ninduction**")
    #Make limitations for maximmum and minimum bore size with errors
    while bore > 200 or bore < 60:
        bore = float(input("Enter Bore Width (mm): "))
        if bore > 200:
            print("Bore must be below 200 millimeters!")
        if bore < 60:
            print("Bore must be higher than 60 millimeters!")
    #Make limitations for bore/stroke rastio with errors
    while bore/stroke <= 0.5 or bore/stroke >= 1.5:
        stroke = float(input("Enter Stroke Length (mm): "))
        if bore/stroke <= 0.5:
            print("Warning: Stroke:Bore Ratio is too high oversquare!")
        if bore/stroke >= 1.5:
            print("Warning: Stroke:Bore Ratio is too low undersquare!")
    #Make limiitations for piston style
    while piston not in pstyles:
        piston = input("Enter Piston Type (flat, flat-top with reliefs, hemi): ").lower()
    #Calulate estimated valve diameter and use volume to substract from pistons
    vdiam = math.sqrt((6000*stroke*(bore**2))/(2286000))
    if piston == "flat":
        ph = 0
    elif piston == "flat-top with reliefs":
        ph = -0.25*((vdiam**2)*math.pi)*4

    elif piston == "hemi":
        ph = -0.25*vdiam
    vol = ((math.pi*((0.5*bore)**2)*stroke)-ph)*(10**(-6))
    while cr < 5 or cr > 15:
        cr = float(input(f"Enter Your Desired Compression Ratio (Format as X in X:1) "))
    #Request for compression ratio with errors
```

```
        if cr < 5:
            print("Warning: Compression Ratio too low for combustion!")
        if cr > 15:
            print("Warning: Compression Ratio too high! Engine will detonate!")

    print(f"**Engine will use {octane[octane.index(math.floor(cr))+1]} octane gasoline**\n**Engine
has a Compression Ratio of {cr:.1f}:1**")
    #Request for cylinder numbers
    while cyl not in cylinders:
        cyl = int(input("Enter the amount of Cylinders: "))
    if cyl == 4:
        rpm = 8000
        print(f"**Maximum engine speed determined to be {rpm} RPM**")
    #For each cylinder, define RPM values
    if cyl == 6:
        rpm = 6800
        print(f"**Maximum engine speed determined to be {rpm} RPM**")
    if cyl == 8:
        rpm = 6200
        print(f"**Maximum engine speed determined to be {rpm} RPM**")
    if cyl == 10:
        rpm = 8500
        print(f"**Maximum engine speed determined to be {rpm} RPM**")
    if cyl == 12:
        rpm = 10000
        print(f"**Maximum engine speed determined to be {rpm} RPM**")

    enginemap = []
    gasinj = []
    #Open enginemap file and append to 2D array
    with open("/Users/haydensupple/Downloads/EngineMapping.txt", "r") as file:
        for line in file:
            line_values = line.strip().split(',')
            enginemap.append(line_values)

        for i in range(0, len(enginemap)-1):
            enginemap[i] = enginemap[i][:int(rpm/100)-8]
    #Open Gasinjector file and append to 2D array
    with open("/Users/haydensupple/Downloads/GasINJ2.txt", "r") as gasfile:
        for line in gasfile:
```

```
            line_values = line.strip().split(',')
            gasinj.append(line_values)
    #Ensure RPM matches maximum RPM
    while int(gasinj[1][-1]) < rpm:
        gasinj[1].append(int(gasinj[1][-1])+100)
    #Enact functions
    ratio2gram(enginemap, gasinj, vol, cyl, int(gasinj[1][-1]))
    exsheet(gasinj)

    print("**Simulation will Report Base Horsepower (Hp) and Torque (Ft-Lbs)**")
##########################################################################################
#######
##Simulation Below##
    #Estimate volume in cubic inches
    vol = vol*cyl*61.0237
    #Enact simulator
    enginemapsim(vol, cr, gasinj)
if __name__ == "__main__":
    main()
```

DynamometerSim.py

```
def enginemapsim(displacement, CR, enginemap):
    #Import all functions
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.interpolate import CubicSpline
    import random

    # Initialize variables
    topTorque = 0
    topInj = 0
    topHP = (displacement/10)*CR*1.1
    topRPM = 0
    RM = 0
    # Find topInjection value
    for val in enginemap[11]:
        try:
            inj = float(val)
            if inj < 10 and inj > topInj:
```

```
            topInj = inj
            topRPM = RM
        except ValueError:
            continue
        RM += 1
    scale = topHP/topInj

    # Prepare figure and axis
    fig, ax1 = plt.subplots()

    # Generate Xvals and Yvals with two extra to end function
    Xvals = np.array([float(x) for x in enginemap[1][1:]])
    Xvals = np.append(Xvals, Xvals[-1]+100)
    Xvals = np.append(Xvals, Xvals[-1]+100)
    #Append all Ys to Yvals with scale
    Yvals = np.array([float(y)*scale for y in enginemap[11][1:len(Xvals)-1]])
    #Add realism scaling to program
    for i in range(0, len(Xvals)-2):
        Yvals[i] = Yvals[i]*(0.5+0.009803*i)
    #Append half the end value and 0 for realism
    Yvals = np.append(Yvals, Yvals[-1]/2)
    Yvals = np.append(Yvals, 0)

    addition = 0
    #Iterate throughout Yvals to add realism scaling
    for i in range(0, len(Yvals)):
        addition += random.randint(-2, 2)
        Yvals[i] = Yvals[i] + addition

    torques = []
    #CubicSpline to calculate a line
    spline = CubicSpline(Xvals, Yvals)
    #Linespace to calculate X Line
    Xinterp = np.linspace(min(Xvals), max(Xvals), 1000)
    Yinterp = spline(Xinterp)
    differential = 0.2
    n = 0
    for num in range(0, len(Xvals)-1):
        try:
            #Iterate throughout num in range of Xvals to find Torque values
```

```
            torque = (spline(float(Xvals[num]))*525.2)/float(Xvals[num])*differential
            #Append to torques
            torques = np.append(torques, torque)
            if torque > topTorque:
                topTorque = torque
                gRPM = Xvals[num]
            if n<14:
                differential += 0.96**(n+1)
            n += 1
        except ValueError:
            continue

    torques = np.append(torques, 0)
    #Output legends
    maxt = f"Max Torque={round(topTorque,2)}(lb•ft)@{gRPM}(RPM)"
    maxh = f"Max Power={round(topHP, 2)}(HP)@{enginemap[1][topRPM]}(RPM)"
    ax1.plot(Xinterp, Yinterp, 'r-', label=maxh)
    ax1.set_xlabel('Engine RPMs')
    ax1.set_ylabel('Horsepower')
    ax1.tick_params(axis='y')

    #Format Labels
    ax2 = ax1.twinx()
    ax2.plot(Xvals, torques, '--', label=maxt, color='#eb4d8f')
    ax2.set_ylabel('Torques (ft•lb)')
    ax2.tick_params(axis='y')

    handles, labels = ax1.get_legend_handles_labels()
    handles2, labels2 = ax2.get_legend_handles_labels()
    ax1.legend(handles + handles2, labels + labels2, loc='upper left')

    #Title Plot
    plt.title('Simulated Dynamometer')

    plt.show()
```