ME 499 Propulsion Final Project

By Austin Vaden-Kiernan

Introduction: The goal of this project is to develop engine designs and performance parameters based on different propellant combinations and different types of engines (Solid, Liquid, Hybrid). Using the given information from the prompt each design configuration must find the optimal O/F ratio, specific impulse, engine and nozzle dimensions, mass of propellant, and propellant tank size. After these parameters have been determined an outline sketch of each concept can then be generated. Based on values calculated and propellant environmental concerns a best design for each engine type will be determined. Finally a final design recommendation for the best engine type and design will be discussed based on environmental impacts and performance of the various propellant combinations.

Overall Problem Givens: Δv = 2300 m/s, mo = 100,000 kg, $\lambda = 0.9$, Time Burnout = 100 s, T = 1962000 N, Dmax = 3 m, $Pc_{initial} = 1000\ psi$

Solid Givens: $\dot{r_b} = 0.04Pc^{0.3}$

Hybrid Givens: $\dot{r} = 0.1Go_{OX}^{.68}$

Assumptions: Engine at sea level, ideal nozzle, ideal rocket analysis, optimal expansion, neglect drag and gravity losses, complete combustion, one dimensional flow.

Problem Approach: The analysis for optimal oxidizer fuel ratio was determined based on CEA analysis using chamber temperature, molecular weight, and gamma. A coefficient of thrust was determined based on gamma at optimal O/F ratio and pressure ratio. Specific impulse could then be calculated based on maximum cstar value from CEA. Using the pressure ratio and gamma an area ratio was determined for each design and used to calculate area of exit and throat. Length of fuel and chamber were calculated based on the diameter constraints of the booster. Mass of propellant was calculated based on the total mass flow rate over the time burnout constraint period given. Finally tank size dimensions were determined based on mass flow rate of oxidizer and fuel providing different volumes for each tank as both propellants require more or less volume based on density. Once volume for oxidizer and fuel was calculated the area and radius for each tank was determined based on area of the chamber.

Equations Used: $At = T/(CF * Pc), Ae = Ae/At * At, Lc = (Lstar * At)/Ac,$

$R = 1/(e^{\Delta v/(Isp*c)}),\ c = Isp * g0, mdot = T/c,\ mp = mdot * t_{burnout}$

$r = O/F_{optimal}, mdotox = (r * mdot)/(1 + r),\ mdotfuel = mdot/(1 + r)$

$Vdotox = mdotox/rhoox,\ Vdotfuel = mdotfuel/rhofuel,\ Vox = Vdotox * t_{burnout}$

$Vfuel = Vdotfuek * t_{burnout},\ Ap = \pi * Rpi^2, Abi = (\pi * 2 * Rpi * L) + (\pi * (Rpf^2 - Rpi^2))$

$Abf = \pi * 2 * Rpf * Lc,\ \dot{r_b} = 0.04Pc^{0.3},\ \dot{r} = 0.1Go_{OX}^{.68},\ Pressure\ ratio = Pc/Pa$

Liquid Rocket Engine Design and Analysis:

Propellant Combination 1: LOX/LCH4(Methane)

(Best Performing Liquid Design):

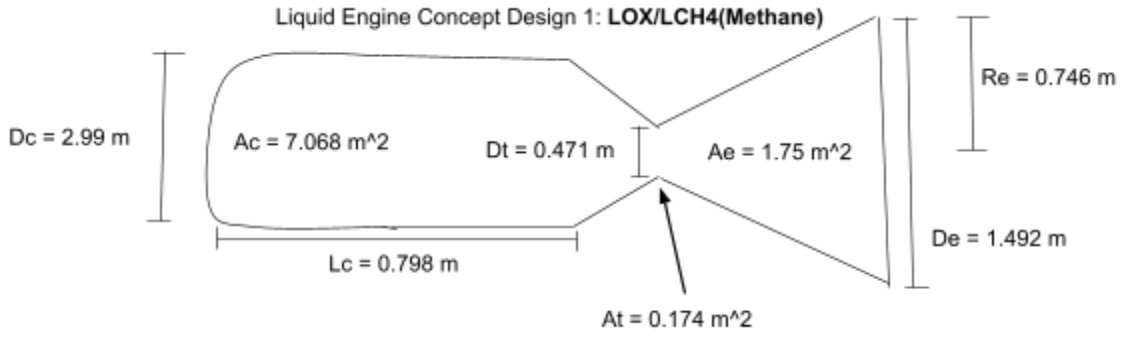


Figure 1: Concept sketch of liquid engine propellant combination 1.

- The optimal O/F ratio for this propellant combination was determined to be 3. This value was taken at the maximum characteristic velocity for the given propellant combination.
- The specific impulse at sea level for this propellant combination was determined to be 310.58 seconds (Isp = 310.58 sec). This value was based on the thrust coefficient and characteristic velocity at optimal O/F point.
- Engine dimensions: Chamber Area = 7.068 m^2, Chamber Diameter = 2.99 m, and Length of Chamber = 0.798 m.
- Nozzle dimensions: Throat Dimensions:

  Area of Throat = 0.174 m^2, Radius of Throat = 0.235 m, Diameter of Throat = 0.471 m

  Area Ratio = 10.048

  Exit Dimensions:

  Area of Exit = 1.75 m^2, Radius of Exit = 0.746 m, Diameter of Exit = 1.492 m

- Mass of propellant needed for the booster using the specific propellant combination of liquid oxygen and liquid Methane was determined to be 64,394.643 kg

- Propellant tank size: Volume of Oxidizer = 42.328 kg/m^3, Volume of Fuel = 38.094 kg/m^3 Length of Oxidizer Tank = 5.988 m, Length of Fuel Tank = 5.389 m

Liquid Engine Concept 1 Discussion: Based on the values calculated for the propellant combination the booster has a reasonable specific impulse at 310 seconds. Liquid oxygen should not come in contact with skin but is otherwise considered a clean fuel for the environment. Methane is bad for the environment as it contributes to greenhouse gas production and is more potent than carbon dioxide for

global warming. The dimensions of the nozzle and engine are within the diameter constraint required of the booster. The mass of propellant for the required delta v and burnout is reasonable considering other propellant combinations require much higher propellant mass. The mass of propellant is low for this rocket design at 64,394.643 kg giving this design a leading edge over other design concepts presented. Ultimately this rocket design provides a good specific impulse, requires less mass to achieve the same desired performance, and allows for a smaller booster engine/nozzle dimensions compared to other concept designs and engine types. Due to the overall higher performance, propellant mass, environment impacts, and propellant chemistry, concept design 1 is the best choice design for liquid engines.

Propellant Combination 2: LOX/LH2

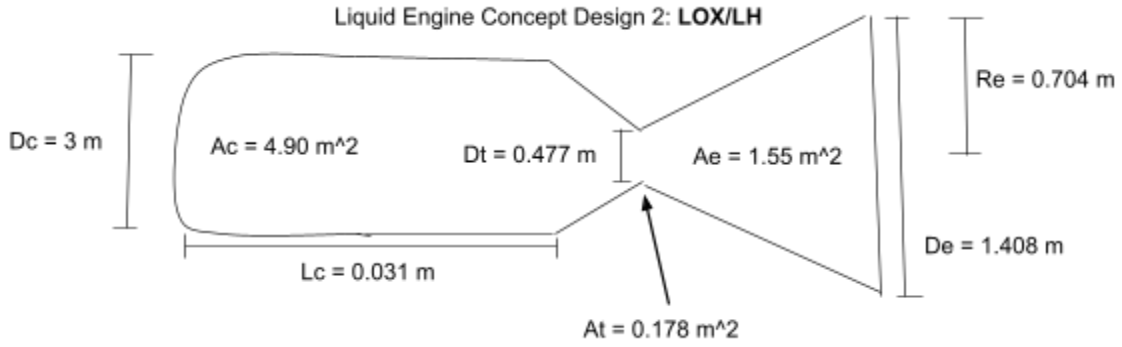


Figure 2: Concept sketch of liquid engine propellant combination 2.

- The optimal O/F ratio for this propellant combination was determined to be 3.5. This value was taken at the maximum characteristic velocity for the given propellant combination.
- The specific impulse at sea level for this propellant combination was determined to be 394.639 seconds (Isp = 394.639 sec). This value was based on the thrust coefficient and characteristic velocity at optimal O/F point.
- Engine dimensions: Chamber Area = 4.90 m^2, Chamber Diameter = 2.99 m, and Length of Chamber = 0.031 m.
- Nozzle dimensions: Throat Dimensions

  Area of Throat = 0.178 m^2, Radius of Throat = 0.235 m, Diameter of Throat = 0.477 m

  Area Ratio = 8.72

  Exit Dimensions:

  Area of Exit = 1.55 m^2, Radius of Exit = 0.704 m, Diameter of Exit = 1.408 m

- Mass of propellant needed for the booster using the specific propellant combination of liquid oxygen and liquid hydrogen was determined to be 50,679.133 kg.
- Propellant tank size: Volume of Oxidizer = 34.546 kg/m^3, Volume of Fuel = 158.62 kg/m^3 Length of Oxidizer Tank = 4.887 m, Length of Fuel Tank = 22.44 m.

Liquid Engine Concept 2 Discussion: The specific impulse value calculated for this propellant combination is very good at 394 seconds. Liquid hydrogen can be dangerous due to extreme freezing on human skin but otherwise these propellants are safe to use environmentally and are very abundant. The cryogenic nature of these propellants does add more complexity to the system but yields good performance using smaller booster dimensions. The mass of propellant is low for this rocket design at 50,679.133 kg giving this design a leading edge over other design concepts presented. Overall this design has high performance characterized by its specific impulse value but the lengths and dimensions leave room to be desired as the tanks are too large.

Solid Rocket Engine Design and Analysis:

Propellant Combination 1: Ammonium Perchlorate / Zirconium

(Best Performing Solid Design):

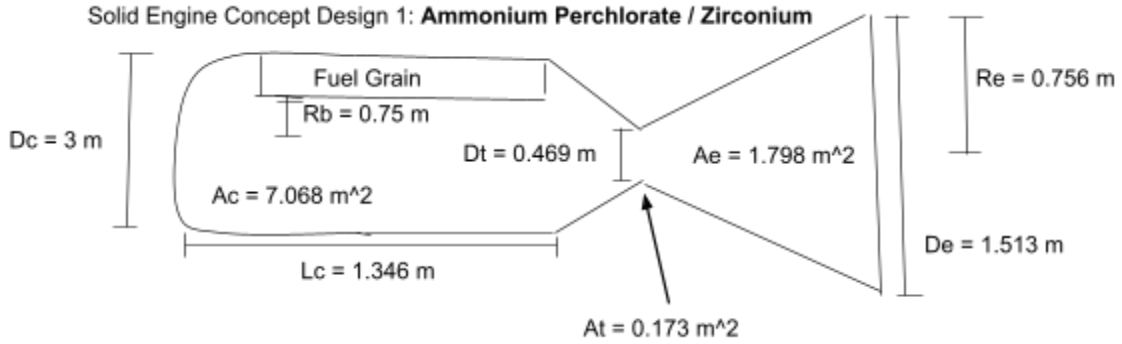


Figure 3: Concept sketch of solid engine propellant combination 1.

- The specific impulse at sea level for this propellant combination was determined to be 251.194 seconds (Isp = 251.194 sec). This value was based on the thrust coefficient and characteristic velocity functions.
- Engine dimensions: Chamber Area = 7.068 m^2 and Length of Chamber = 1.346 m.
- Nozzle dimensions: Throat Dimensions

  Area of Throat = 0.173 m^2, Radius of Throat = 0.234 m, Diameter of Throat = 0.469 m

  Area Ratio = 10.39

  Burn Area:

  Initial Surface Area of Burn = 1.044 m^2, Final Surface Area of Burn = 12.69 m^2

  Exit Dimensions:

  Area of Exit = 1.798 m^2, Radius of Exit = 0.756 m, Diameter of Exit = 1.513 m

- Mass of propellant needed for the booster using the specific propellant combination of Ammonium Perchlorate and Zirconium was determined to be 79,619.58 kg

Solid Engine Concept 1 Discussion: This rocket design has an Isp (251.194 s) within the lower bound of good specific impulse performance for solid engine rockets. The dimensions of the motor are very good as they are not too big and within the diameter constraint of the booster. The propellant combination of Ammonium Perchlorate and Zirconium works well in this design for giving high energetics allowing a smaller motor design while also meeting the performance requirements of the booster. Ammonium Perchlorate is a highly reactive chemical and is considered dangerous for humans and the environment often, common uses are in fireworks and rocket fuel. Zirconium is not considered to be bad for the environment as many plants and animals on earth can absorb the element without concern. The mass of propellant is quite high for this rocket design at 79,619.58 kg leaving room for improvement in other concept designs. Overall this is a very functional design concept for a solid engine rocket considering the relatively high specific impulse, smaller size, and mixed effects on the environment. Due to the overall higher performance, propellant mass, environment impacts, and propellant chemistry, concept design 1 is the best choice design for solid engines.

Propellant Combination 2: Ammonium Perchlorate / Aluminum

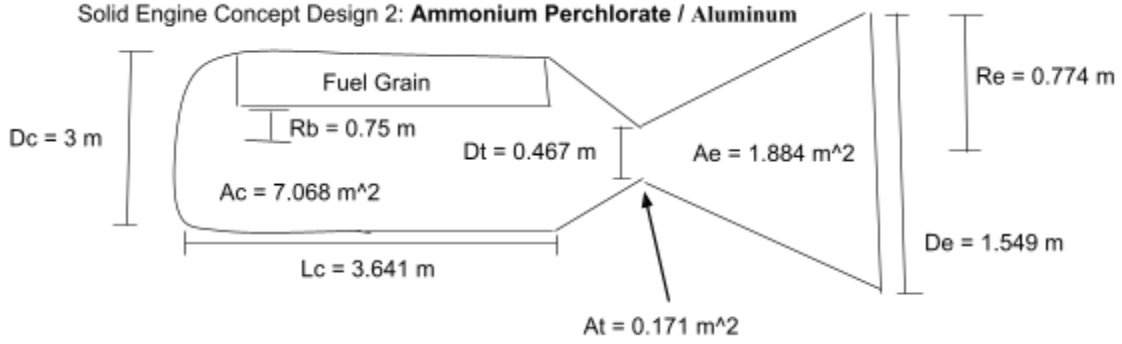


Figure 4: Concept sketch of solid engine propellant combination 2.

- The specific impulse at sea level for this propellant combination was determined to be 237.286 seconds (Isp = 237.286 sec). This value was based on the thrust coefficient and characteristic velocity functions.
- Engine dimensions: Chamber Area = 7.068 m^2 and Length of Chamber = 3.641 m.
- Nozzle dimensions: Throat Dimensions

  Area of Throat = 0.171 m^2, Radius of Throat = 0.232 m, Diameter of Throat = 0.467 m

  Area Ratio = 11.024

  Burn Area:

  Initial Surface Area of Burn = 11.857 m^2, Final Surface Area of Burn = 34.317 m^2

  Exit Dimensions:

  Area of Exit = 1.884 m^2, Radius of Exit = 0.774 m, Diameter of Exit = 1.549 m

- Mass of propellant needed for the booster using the specific propellant combination of Ammonium Perchlorate and Aluminum was determined to be 84,286.206 kg

Solid Engine Concept 2 Discussion: The specific impulse calculated for the second design is quite low at Isp = 237.286 seconds. The dimensions of the nozzle and engine are a little high but still not unusable. As discussed before Ammonium Perchlorate is a highly reactive chemical and is considered dangerous for humans and the environment, common uses are in fireworks and rocket fuel. The actual process of making and manufacturing aluminum can be toxic for the environment but as a rocket fuel it is not considered dangerous. The mass of propellant is quite high for this rocket design at 84,286.206 kg leaving room for improvement in other concept designs.Overall the rocket has average to poor performance and the fuels are not a healthy combination for the environment.

Hybrid Rocket Engine Design and Analysis:

Propellant Combination 1 : LOX/Polybutadiene (C2H6)

(Best Performing Hybrid Design):

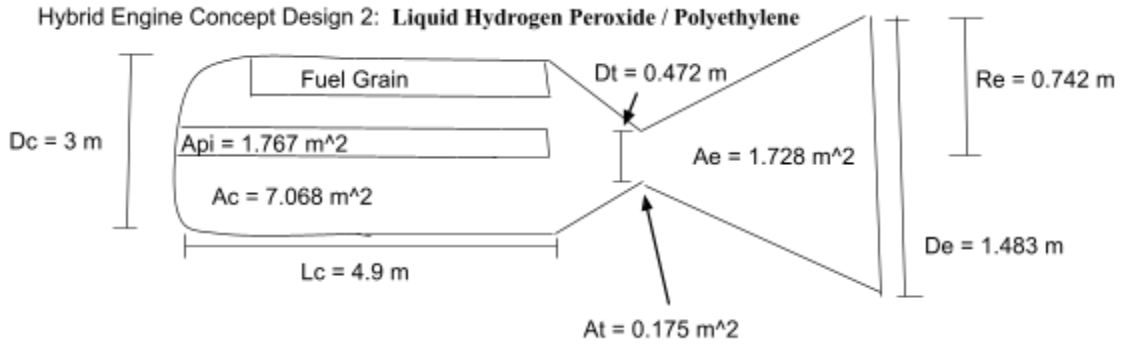


Figure 5: Concept sketch of hybrid engine propellant combination 1.

- The optimal O/F ratio for this propellant combination was determined to be 2.615. This value was taken at the maximum characteristic velocity for the given propellant combination.
- The specific impulse at sea level for this propellant combination was determined to be 319.68 seconds (Isp = 319.68 sec). This value was based on the thrust coefficient and characteristic velocity at optimal O/F point.
- Engine dimensions: Chamber Area = 7.068 m^2 and Length of Chamber = 4.9 m.
- Nozzle dimensions: Throat Dimensions

Area of Throat = 0.175 m^2, Radius of Throat = 0.236 m, Diameter of Throat = 0.472 m

Area Ratio = 9.896

Exit Dimensions:

Area of Exit = 1.728 m^2, Radius of Exit = 0.742 m, Diameter of Exit = 1.483 m

Burn Area:

Initial Surface Area of Burn = 28.413 m^2, Final Surface Area of Burn = 46.22 m^2

Area of Port:

Initial Area of Port = 1.767 m^2, Final Area of Port = 7.068 m^2

- Mass of propellant needed for the booster using the specific propellant combination of liquid oxygen and Polybutadiene was determined to be 86,482.195 kg

- Propellant tank size: Volume of Oxidizer = 52.931 kg/m^3, Volume of Fuel = 26 kg/m^3
  Radius of Oxidizer Tank = 1.88 m, Radius of Fuel Tank = 1.29 m

Hybrid Engine Concept 1 Discussion: The analysis for optimal oxidizer fuel ratio was determined based on CEA analysis using chamber temperature, molecular weight, and gamma. Concept 1 using liquid oxygen and Polybutadiene gave a high specific impulse value at Isp = 319.68 seconds. The length dimensions are very high at 4.9 meters making the booster engine quite bulky. The mass of propellant is quite heavy at 86,482.195 kg creating a tradeoff with liftoff mass and rocket performance. Liquid oxygen is eco friendly propellant and only hazardous when in contact with skin due to low temperatures. Polybutadiene is a polymer and therefore acts as a permanent pollutant for the environment as degradation will take hundreds of years [1]. Overall the rocket design provides a high specific impulse, high motor and size dimensions for oxidizer and fuel tanks, and weighs a lot based on a large mass of propellant required to achieve deltav required. Due to the overall higher performance, propellant mass, environment impacts, and propellant chemistry, concept design 1 is the best choice design for hybrid engines.

Propellant Combination 2: Liquid Hydrogen Peroxide (H2O2)/Polyethylene (C2H4)

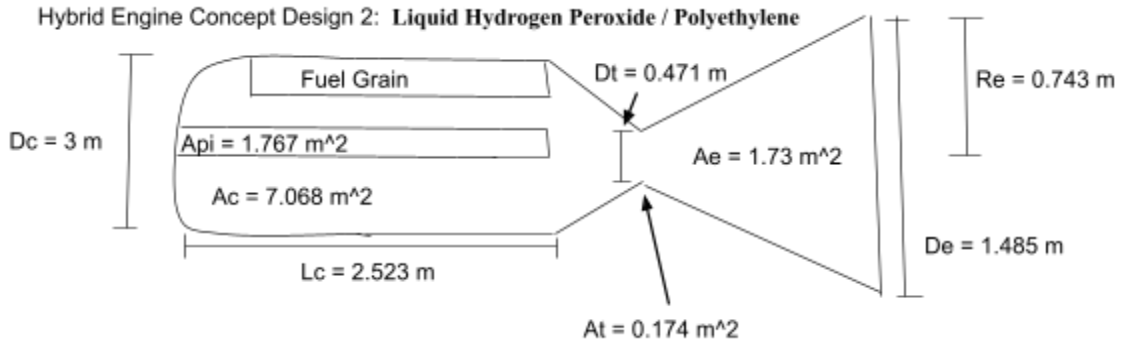


Figure 6: Concept sketch of hybrid engine propellant combination 2.

- The optimal O/F ratio for this propellant combination was determined to be 5.5. This value was taken at the maximum characteristic velocity for the given propellant combination.
- The specific impulse at sea level for this propellant combination was determined to be 283.113 seconds (Isp = 283.113 sec). This value was based on the thrust coefficient and characteristic velocity at optimal O/F point.
- Engine dimensions: Chamber Area = 7.068 m^2 and Length of Chamber = 2.523 m.
- Nozzle dimensions: Throat Dimensions

  Area of Throat = 0.174 m^2, Radius of Throat = 0.235 m, Diameter of Throat = 0.471 m

  Area Ratio = 9.926

Exit Dimensions:

Area of Exit = 1.73 m^2, Radius of Exit = 0.743 m, Diameter of Exit = 1.485 m

Burn Area:

Initial Surface Area of Burn = 17.194 m^2, Final Surface Area of Burn = 23.785 m^2

Area of Port:

Initial Area of Port = 1.767 m^2, Final Area of Port = 7.068 m^2

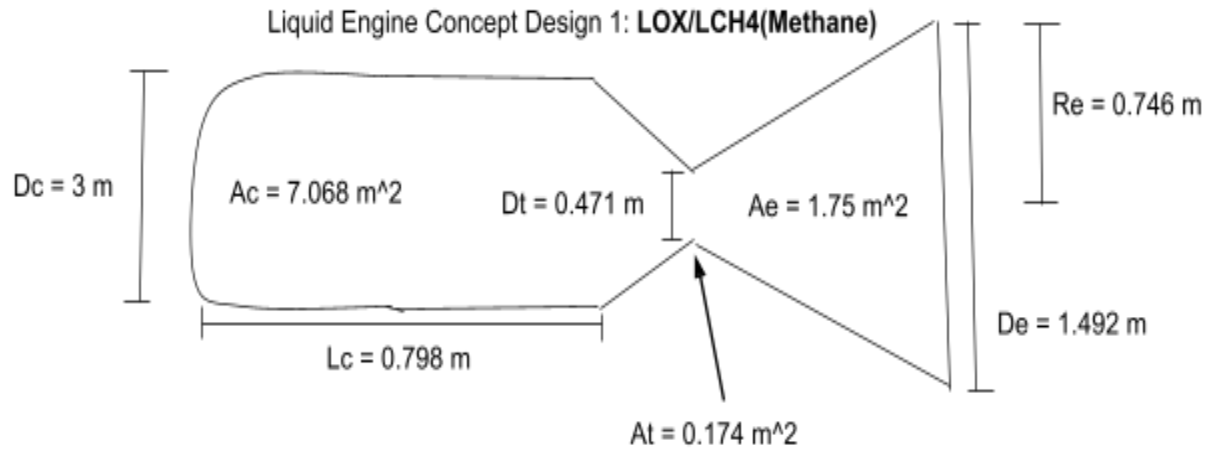
- Mass of propellant needed for the booster using the specific propellant combination of liquid Hydrogen Peroxide and Polyethylene was determined to be 83,487.175 kg
- Propellant tank size: Volume of Oxidizer = 48.71 kg/m^3, Volume of Fuel = 13.379 kg/m^3 Radius of Oxidizer Tank = 2.478 m, Radius of Fuel Tank = 1.299 m

Hybrid Engine Concept 2 Discussion: Concept 2 using Liquid Hydrogen Peroxide and Polyethylene gave a good specific impulse value at Isp = 283.113 seconds. The length dimensions are reasonable at 2.523 meters making the booster engine small enough for constraints. The mass of propellant is quite heavy at 83,487.175 kg creating a tradeoff with liftoff mass and rocket performance. Liquid Hydrogen Peroxide is a safe propellant that in small concentrations does not hurt ecosystems or environment. Liquid Hydrogen Peroxide is also a storable propellant giving an added benefit to design of concept 2. Polybutadiene is also a polymer and therefore acts as a permanent pollutant for the environment as degradation will take hundreds of years. Overall the rocket design provides lower specific impulse, lower motor and size dimensions for oxidizer and fuel tanks, and weighs a lot based on a large mass of propellant required to achieve deltav required.

Design Discussion: When comparing the different engine types and propellant combinations there are four main areas of interest to determine a suitable solution. Specific impulse, mass of propellant, environmental impacts of propellant combination, and engine/nozzle dimensions are key to determining an adequate recommendation. The liquid engine combinations provided the highest specific impulse and also had the lowest weight associated with mass of propellant. The propellants used are also safer for humans and the global ecosystem compared to propellants used in solid engines. Solid engine design concepts provided lower specific impulse values and higher mass of propellant constraints. The dimensions were also increased to accommodate for these increases in required mass to reach deltav with the given time burnout constraint. The propellants analyzed were also more toxic for humans and the environment overall compared to liquid and hybrid engines. Hybrid engines provided specific impulse values very close to liquid engines. However, hybrid engines are more complex and have propellant combinations that are more harmful to the environment through the use of polymer fuels that take hundreds of years to decompose. The structure for hybrid engines is very bulky with large dimensions and the mass of propellant in both analyses were very high compared to liquid engines.

Design Recommendation: After discussing the various tradeoffs of each engine type and their respective benefits and issues a design recommendation can now be made considering the four main areas of analysis. Propellant combination 1 is the best booster engine design considering the propellants used, the environmental impacts, mass of propellant, and size of the structure. The design gives high specific impulse at (Isp = 310.58 sec) compared to each design combination examined. Liquid oxygen should not come in contact with skin but is otherwise a safe oxidizer to use as its environmental impacts are slight and is very accessible for production and supply. Liquid methane causes increased global warming and is not a safe propellant for the environment. Overall the propellant combination could be safer; however, in the purpose of this analysis is a cleaner propellant combination compared to solid and hybrid engine fuels using metals and polymers respectively. The propellant mass for this concept is also one of the lowest evaluated at 64,394.643 kg. This lower mass of propellant is a great benefit for the mission as the rocket will weigh less overall and increase performance characteristics of the overall launch vehicle. The cost of liquid oxygen and liquid methane is also considerably lower compared to other propellant designs with liquid oxygen about 17.223 cents per kilogram and liquid methane at 1.35 dollars per kilogram. The structure is also very small with a chamber length of just 0.798 meters allowing for less weight and greater design options for possibly more than one engine per booster with the launch vehicle space savings. This concept should move forward based on the low propellant mass, high propellant energetics, smaller size of structure compared to alternative designs, and reasonable environmental impacts compared to propellant combinations of solid and hybrid engines.

Final Recommended Concept Design:



References:

[1] Okninski, A., Kopacz, W., Kaniewski, D., & Sobczak, K. (2021). Hybrid rocket propulsion technology for space transportation revisited - propellant solutions and challenges. *FirePhysChem*, *Volume 1*(Issue 4), 260–271. https://doi.org/https://doi.org/10.1016/j.fpc.2021.11.015. (https://www.sciencedirect.com/science/article/pii/S2667134421000614)

Appendix/Jupyter Notebook Code:

Liquid Engine Analysis ME 499 Final Project

March 17, 2022

[34]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

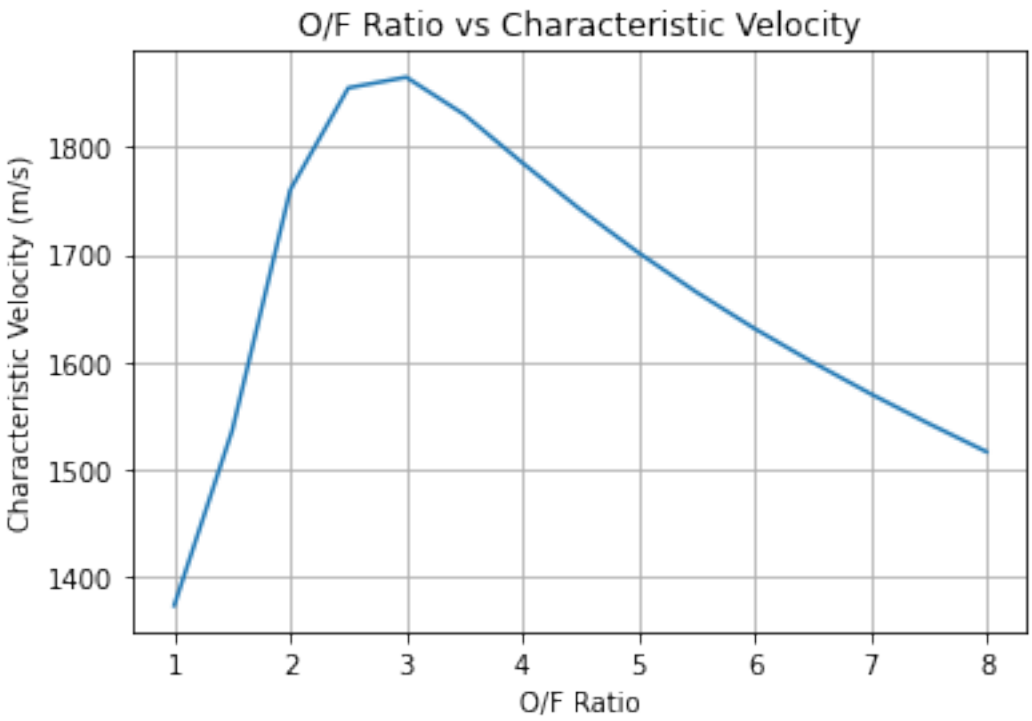
[19]:
```
Deltav = 2300
mo = 100000
MF = .9
tb = 100
g0 = 9.81
# Thrust calc 100000 kg * g0 *2
T = 1962000
Dmax = 3
# psi to Pa
Pc = 6.895e6
Pa = 101325
Ru = 8314.5
```

[20]:
```
# Part 1 Propellent and Design Concept 1 using LOX/LCH4(Methane)
# 1) Optimal O/F ratio determined for specific propellant combination, plots␣
 ↪generated of relationship.
# values from CEA rocket problem analysis
Tc = [1277.45,1673.77,2552.20,3162.18,3466.95,3557.08,3557.69,3525.11,3478.
 ↪53,3425.20,3368.25,3309.20,3248.85,3187.67,3125.96]
MW = [13.559,13.395, 16.021,18.466,20.447,21.931,23.089,24.041,24.847,25.539,26.
 ↪139,26.661,27.118,27.518,27.869]
gamma = [1.1802,1.2799, 1.2280,1.1731, 1.1402,1.1297,1.1272,1.1268,1.1274,1.
 ↪1286,1.1303,1.1324,1.1351,1.1381,1.1416]

#OF values from 1 to 8 in increments of .5
OF = np.linspace(1,8,15)


def get_cstar(Tc, MW, gamma):
#     Calculates cstar using chamber properties

            return (np.sqrt(Ru * Tc / (gamma * MW)) * (2 / (gamma + 1))**(-0.
 ↪5*(gamma + 1)/(gamma - 1)))
```

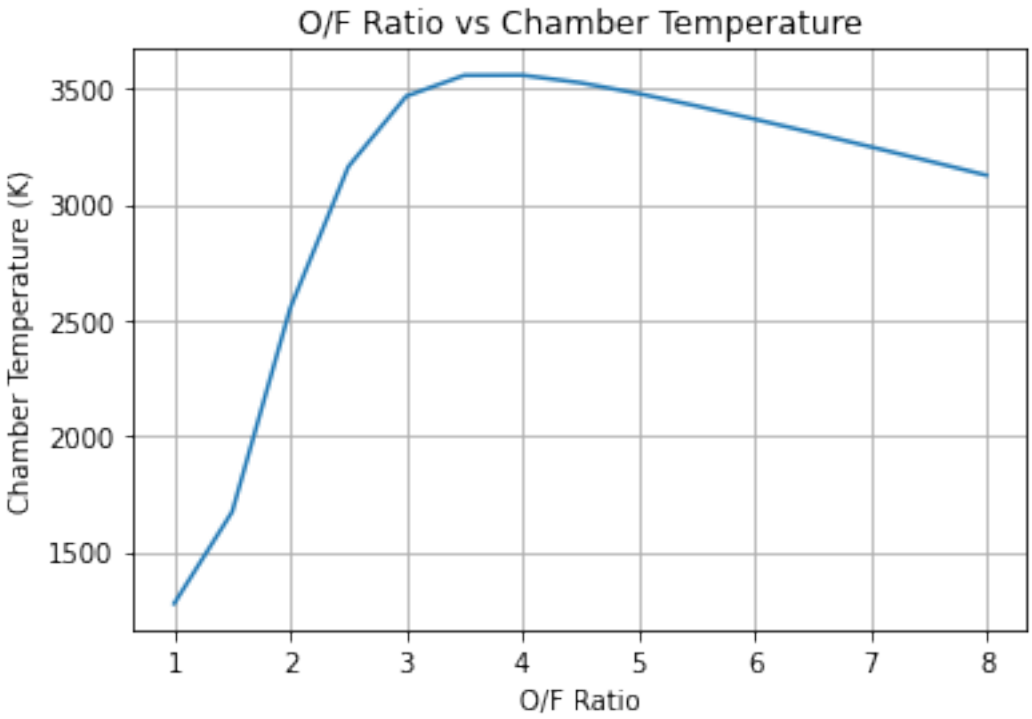
```
Cstar0 = get_cstar(Tc[0], MW[0], gamma[0])
Cstar1 = get_cstar(Tc[1], MW[1], gamma[1])
Cstar2 = get_cstar(Tc[2], MW[2], gamma[2])
Cstar3 = get_cstar(Tc[3], MW[3], gamma[3])
Cstar4 = get_cstar(Tc[4], MW[4], gamma[4])
Cstar5 = get_cstar(Tc[5], MW[5], gamma[5])
Cstar6 = get_cstar(Tc[6], MW[6], gamma[6])
Cstar7 = get_cstar(Tc[7], MW[7], gamma[7])
Cstar8 = get_cstar(Tc[8], MW[8], gamma[8])
Cstar9 = get_cstar(Tc[9], MW[9], gamma[9])
Cstar10 = get_cstar(Tc[10], MW[10], gamma[10])
Cstar11 = get_cstar(Tc[11], MW[11], gamma[11])
Cstar12 = get_cstar(Tc[12], MW[12], gamma[12])
Cstar13 = get_cstar(Tc[13], MW[13], gamma[13])
Cstar14 = get_cstar(Tc[14], MW[14], gamma[14])

Cstar =␣
 ↪[Cstar0,Cstar1,Cstar2,Cstar3,Cstar4,Cstar5,Cstar6,Cstar7,Cstar8,Cstar9,Cstar10,Cstar11,Csta

Cstarmax =  np.max(Cstar)
OFMax = OF[Cstar.index(Cstarmax)]
GammaMAX = gamma[Cstar.index(Cstarmax)]

# these are the values at the optimized maximum Characteristic Velocity
print(OFMax, 'Optimal Oxidizer to Fuel Ratio at Max Characteristic Velocity')
print(Cstarmax, ' (m/s) Max Characteristic Velocity at Optimal Oxidizer to Fuel␣
 ↪Ratio')
print(GammaMAX, 'Max Gamma at Optimal Oxidizer to Fuel Ratio')
```

```
3.0 Optimal Oxidizer to Fuel Ratio at Max Characteristic Velocity
1864.9746453984762  (m/s) Max Characteristic Velocity at Optimal Oxidizer to
Fuel Ratio
1.1402 Max Gamma at Optimal Oxidizer to Fuel Ratio
```

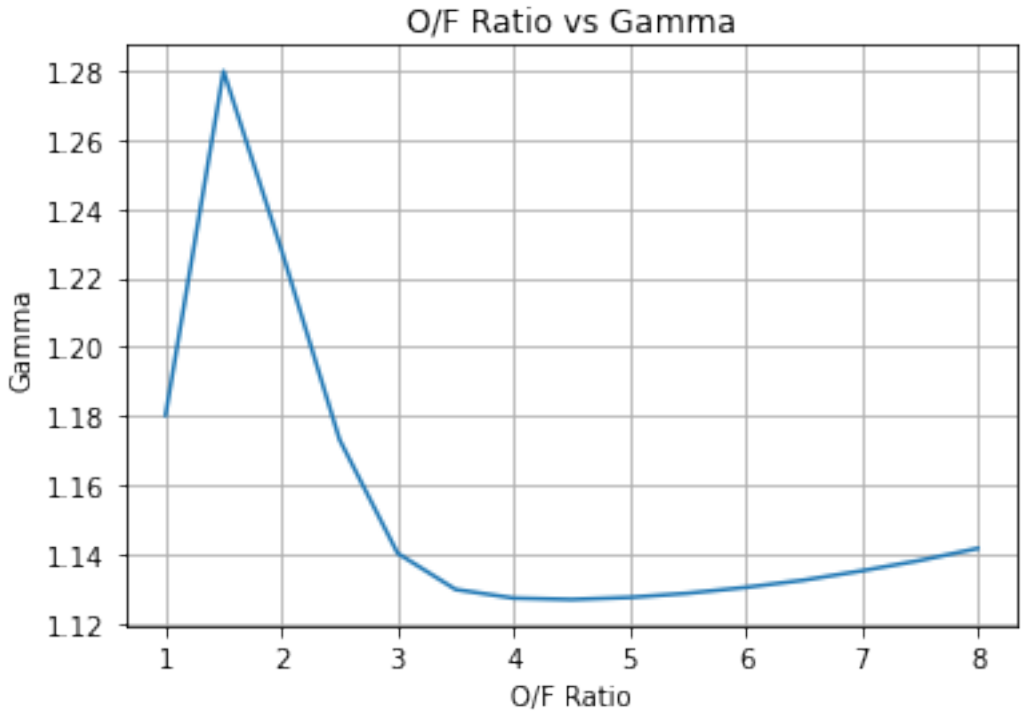
[21]:
```
plt.plot(OF, Cstar)
plt.title('O/F Ratio vs Characteristic Velocity')
plt.xlabel('O/F Ratio')
plt.ylabel('Characteristic Velocity (m/s)')
plt.grid()
plt.show()
```



[22]:
```
plt.plot(OF, Tc)
plt.title('O/F Ratio vs Chamber Temperature')
plt.xlabel('O/F Ratio')
plt.ylabel('Chamber Temperature (K)')
plt.grid()
plt.show()
```



[23]:
```
plt.plot(OF, gamma)
plt.title('O/F Ratio vs Gamma')
plt.xlabel('O/F Ratio')
plt.ylabel('Gamma')
plt.grid()
plt.show()
```



[24]:
```
# 2) Specific Impulse of Propellant combination using LO2/LCH4
# cstarmax and gammamax used becuase of optimal O/F ratio
gamma = GammaMAX
pressure_ratio = Pc/Pa
def calc_thrust_coeff(gamma , pressure_ratio):
    ''' Calculates thrust coefficient for optimum expansion.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return np.sqrt(2 * np.power(gamma, 2) / (gamma - 1) *
        np.power(2 / (gamma + 1), (gamma + 1)/(gamma - 1)) * (1 - np.power(1.0 /
 ↪ pressure_ratio, (gamma - 1)/gamma)))

CF = calc_thrust_coeff(gamma, pressure_ratio)
Isp = (Cstarmax*CF)/g0
print(Isp, 'Specific Impulse, s')
```

```
310.5848383952276 Specific Impulse, s
```

[33]:
```
# 3) Engine and nozzle dimension calculations
At = T/(Pc*CF)
print(At, 'Area of Throat, m^2')
Rt = np.sqrt(At/(np.pi))
```

```
print(Rt, 'Radius of Throat, m')
Dt = Rt*2
print(Dt, 'Diameter of Throat, m')

gamma = GammaMAX
pressure_ratio = Pc/Pa
# Area Exit over area throat ratio function calculation
def get_area_ratio(pressure_ratio, gamma):
    '''Calculates area ratio based on specific heat ratio and pressure ratio.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return (np.power(2 / (gamma + 1), 1/(gamma-1)) * np.power(pressure_ratio, 1␣
 ↪/ gamma) *
        np.sqrt((gamma - 1) / (gamma + 1) /(1 - np.power(pressure_ratio, (1 -␣
 ↪gamma)/gamma))))

AeAt = get_area_ratio(pressure_ratio, gamma)

print(AeAt, 'Area Ratio')

Ae = AeAt*At
print(Ae, 'Area of Exit, m^2')
Re = np.sqrt(Ae/(np.pi))
print(Re, 'Radius of Exit, m')
De = Re*2
print(De, 'Diameter of Exit, m')

# L star for LO2 and LCH4
Lstar = .9
Dl = .5
Ac = np.pi*(Dl/2)**2
print(Ac, 'Chamber Area, m^2')
Lc = (At*Lstar)/Ac
print(Lc, 'Length of Chamber, m')
```

```
0.17417603735669138 Area of Throat, m^2
0.23546115311650054 Radius of Throat, m
0.4709223062330011 Diameter of Throat, m
10.048117548245294 Area Ratio
1.7501412974475985 Area of Exit, m^2
0.7463827953510831 Radius of Exit, m
1.4927655907021662 Diameter of Exit, m
0.19634954084936207 Chamber Area, m^2
0.7983641466281103 Length of Chamber, m
```

[26]:
```
# 4) mass of propellent needed using LO2/LCH4
c = Isp*g0
# F = T
mdot = T/c
mp = tb*mdot
print(mp, 'Mass of Propellent, kg')
```

```
64394.6436771954 Mass of Propellent, kg
```

[27]:
```
# 5) Propellant tank size using LO2/LCH4
R = 1/(np.exp(Deltav/(Isp*g0)))
r = OFMax
mdotox = (r*mdot)/(1+r)
# mdotox = T/c
print(mdotox, 'Mass Flow Rate of LO2')
mdotf = mdot/(1+r)
# mdotf = mdotox/OF
print(mdotf, 'Mass Flow Rate of LCH4')

# density of oxidizer and methane
rhoox = 1141
rhoCH4 = 422.6
Vdotox = mdotox/rhoox
Vdotf = mdotf/rhoCH4
print(Vdotox, 'Volumetric Flow Rate of LO2, kg/m^3')
print(Vdotf, 'Volumetric Flow Rate of LCH4, kg/m^3')

Vox = Vdotox * tb
Vf = Vdotf * tb
print(Vox, 'Volume of Oxidizer, kg/m^3')
print(Vf, 'Volume of Fuel, kg/m^3')

Lcox = Vox/Ac
print(Lcox, 'Length of Oxidizer Tank')
Lcf = Vf/Ac
print(Lcf,'Length of Fuel Tank')
```

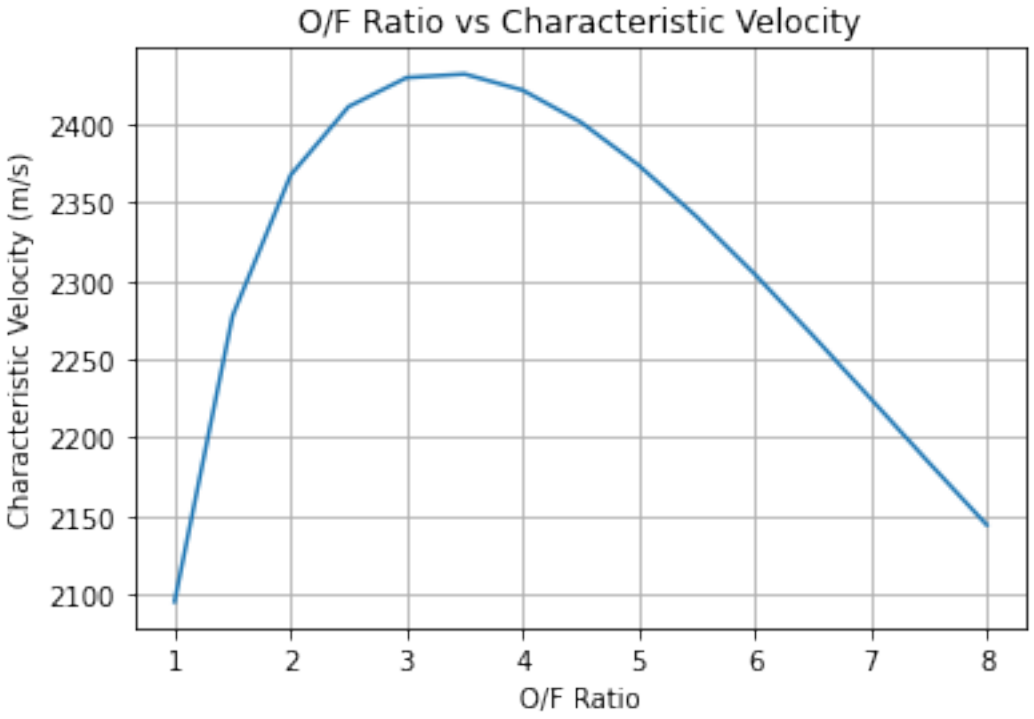
```
482.95982757896553 Mass Flow Rate of LO2
160.9866091929885 Mass Flow Rate of LCH4
0.42327767535404515 Volumetric Flow Rate of LO2, kg/m^3
0.38094323046140205 Volumetric Flow Rate of LCH4, kg/m^3
42.32776753540451 Volume of Oxidizer, kg/m^3
38.094323046140204 Volume of Fuel, kg/m^3
5.988154162937138 Length of Oxidizer Tank
5.389244281362419 Length of Fuel Tank
```

[72]:
```
# Part 2 Propellent and Design Concept 2 using LOX/LH2
# 1) Optimal O/F ratio determined for specific propellant combination, plots␣
 ↪generated of relationship.
# values from CEA rocket problem analysis
Tc1 = [977.49,1411.90,1797.68,2143.07,2450.82,2719.07,2946.10,3132.96,3282.
 ↪62,3398.37,3483.35,3540.88,3574.89,3589.89,3590.47]
MW1 = [4.032,5.040,6.048,7.054,8.055,9.041,10.002,10.930,11.818,12.662,13.
 ↪458,14.203,14.894,15.535,16.127]
gamma1 = [1.3587,1.3157,1.2823,1.2558,1.2317,1.2087,1.1881,1.1712,1.1578,1.
 ↪1475,1.1401,1.1350,1.1320,1.1304,1.1299]

#OF values from 1 to 8 in increments of .5
OF1 = np.linspace(1,8,15)


def get_cstar1(Tc1, MW1, gamma1):
#     Calculates cstar using chamber properties

            return (np.sqrt(Ru * Tc1 / (gamma1 * MW1)) * (2 / (gamma1 +␣
 ↪1))**(-0.5*(gamma1 + 1)/(gamma1 - 1)))

Cstar10 = get_cstar1(Tc1[0], MW1[0], gamma1[0])
Cstar11 = get_cstar1(Tc1[1], MW1[1], gamma1[1])
Cstar12 = get_cstar1(Tc1[2], MW1[2], gamma1[2])
Cstar13 = get_cstar1(Tc1[3], MW1[3], gamma1[3])
Cstar14 = get_cstar1(Tc1[4], MW1[4], gamma1[4])
Cstar15 = get_cstar1(Tc1[5], MW1[5], gamma1[5])
Cstar16 = get_cstar1(Tc1[6], MW1[6], gamma1[6])
Cstar17 = get_cstar1(Tc1[7], MW1[7], gamma1[7])
Cstar18 = get_cstar1(Tc1[8], MW1[8], gamma1[8])
Cstar19 = get_cstar1(Tc1[9], MW1[9], gamma1[9])
Cstar110 = get_cstar1(Tc1[10], MW1[10], gamma1[10])
Cstar111 = get_cstar1(Tc1[11], MW1[11], gamma1[11])
Cstar112 = get_cstar1(Tc1[12], MW1[12], gamma1[12])
Cstar113 = get_cstar1(Tc1[13], MW1[13], gamma1[13])
Cstar114 = get_cstar1(Tc1[14], MW1[14], gamma1[14])

Cstar1 =␣
 ↪[Cstar10,Cstar11,Cstar12,Cstar13,Cstar14,Cstar15,Cstar16,Cstar17,Cstar18,Cstar19,Cstar110,C

Cstarmax1 =  np.max(Cstar1)
OFMax1 = OF1[Cstar1.index(Cstarmax1)]
GammaMAX1 = gamma1[Cstar1.index(Cstarmax1)]

# these are the values at the optimized maximum Characteristic Velocity
print(OFMax1, 'Optimal Oxidizer to Fuel Ratio at Max Characteristic Velocity')
```

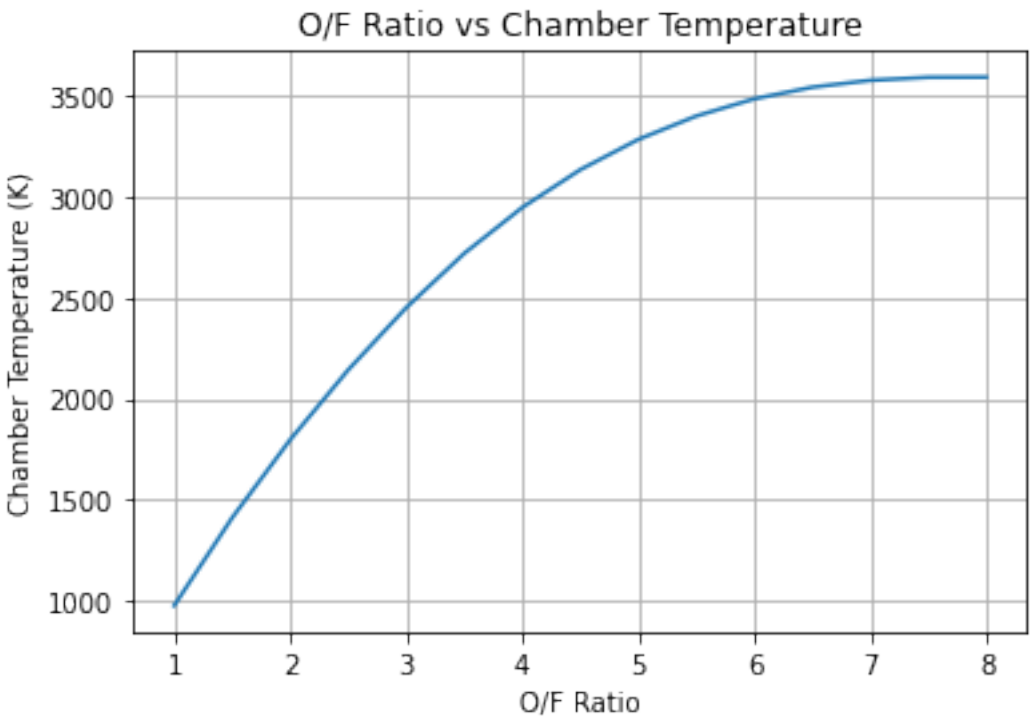
```
print(Cstarmax1, ' (m/s) Max Characteristic Velocity at Optimal Oxidizer to␣
↪Fuel Ratio')
print(GammaMAX1, 'Max Gamma at Optimal Oxidizer to Fuel Ratio')
```

```
3.5 Optimal Oxidizer to Fuel Ratio at Max Characteristic Velocity
2431.996466417375  (m/s) Max Characteristic Velocity at Optimal Oxidizer to Fuel
Ratio
1.2087 Max Gamma at Optimal Oxidizer to Fuel Ratio
```

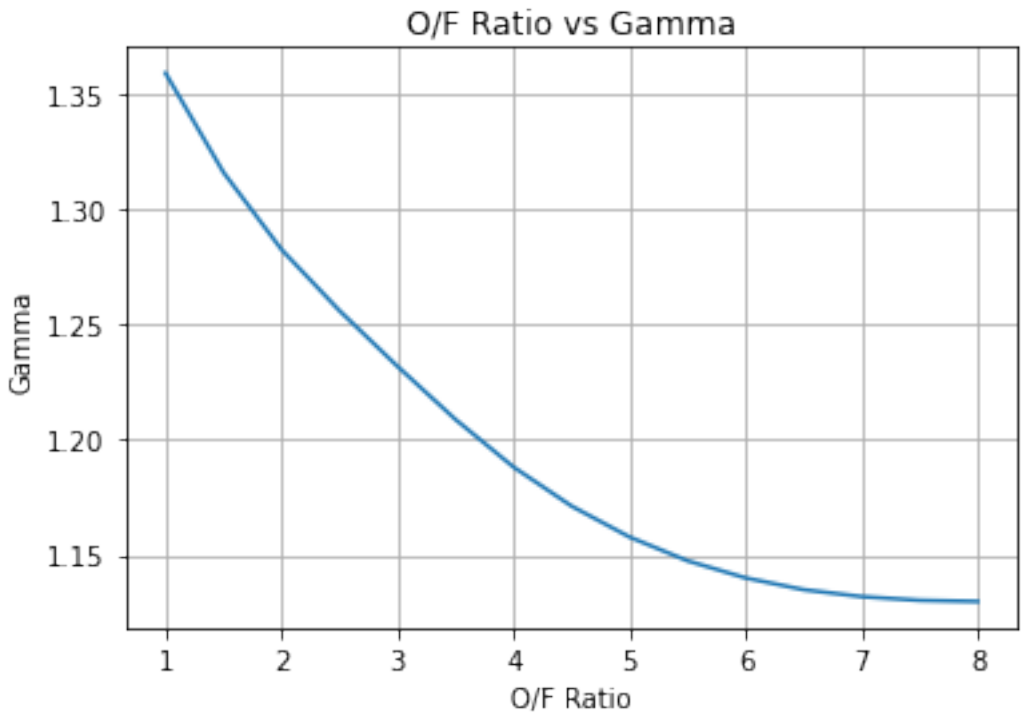
[73]:
```
plt.plot(OF1, Cstar1)
plt.title('O/F Ratio vs Characteristic Velocity')
plt.xlabel('O/F Ratio')
plt.ylabel('Characteristic Velocity (m/s)')
plt.grid()
plt.show()
```



[74]:
```
plt.plot(OF1, Tc1)
plt.title('O/F Ratio vs Chamber Temperature')
plt.xlabel('O/F Ratio')
plt.ylabel('Chamber Temperature (K)')
plt.grid()
plt.show()
```



[75]:
```
plt.plot(OF1, gamma1)
plt.title('O/F Ratio vs Gamma')
plt.xlabel('O/F Ratio')
plt.ylabel('Gamma')
plt.grid()
plt.show()
```



[76]:
```
# 2) Specific Impulse of Propellant combination using LO2/LH2
# cstarmax and gammamax used becuase of optimal O/F ratio
gamma1 = GammaMAX1
pressure_ratio1 = Pc/Pa
def calc_thrust_coeff1(gamma1 , pressure_ratio1):
    ''' Calculates thrust coefficient for optimum expansion.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return np.sqrt(2 * np.power(gamma1, 2) / (gamma1 - 1) *
        np.power(2 / (gamma1 + 1), (gamma1 + 1)/(gamma1 - 1)) * (1 - np.power(1.
 ↪0 / pressure_ratio1, (gamma1 - 1)/gamma1)))

CF1 = calc_thrust_coeff1(gamma1, pressure_ratio1)
Isp1 = (Cstarmax1*CF1)/g0
print(Isp1, 'Specific Impulse, s')
```

```
394.6397390214911 Specific Impulse, s
```

[83]:
```
# 3) Engine and nozzle dimension calculations
At1 = T/(Pc*CF1)
print(At1, 'Area of Throat, m^2')
Rt1 = np.sqrt(At1/(np.pi))
```

```
print(Rt, 'Radius of Throat, m')
Dt1 = Rt1*2
print(Dt1, 'Diameter of Throat, m')

gamma1 = GammaMAX1
pressure_ratio1 = Pc/Pa
# Area Exit over area throat ratio function calculation
def get_area_ratio1(pressure_ratio1, gamma1):
    '''Calculates area ratio based on specific heat ratio and pressure ratio.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return (np.power(2 / (gamma1 + 1), 1/(gamma1-1)) * np.
 ↪power(pressure_ratio1, 1 / gamma1) *
        np.sqrt((gamma1 - 1) / (gamma1 + 1) /(1 - np.power(pressure_ratio1, (1␣
 ↪- gamma1)/gamma1))))

AeAt1 = get_area_ratio1(pressure_ratio1, gamma1)

print(AeAt1, 'Area Ratio')

Ae1 = AeAt1*At1
print(Ae1, 'Area of Exit, m^2')
Re1 = np.sqrt(Ae1/(np.pi))
print(Re1, 'Radius of Exit, m')
De1 = Re1*2
print(De1, 'Diameter of Exit, m')


# range of Lstar for LO2 and LH2 between .7 and 1 choose average = .85
Lstar1 = .85
Dc1 = 2.5
Ac1 = np.pi*(Dc1/2)**2
print(Ac1, 'Chamber Area, m^2')
Lc1 = (At1*Lstar1)/Ac1
print(Lc1, 'Length of Chamber, m')
```

```
0.17875485638937125 Area of Throat, m^2
0.23546115311650054 Radius of Throat, m
0.47707206160956694 Diameter of Throat, m
8.72090936335439 Area Ratio
1.558904900831137 Area of Exit, m^2
0.7044251852077074 Radius of Exit, m
1.4088503704154147 Diameter of Exit, m
4.908738521234052 Chamber Area, m^2
0.030953294267702727 Length of Chamber, m
```

[84]:
```
# 4) mass of propellent needed using LO2/LH2
c1 = Isp1*g0
# F = T
mdot1 = T/c1
mp1 = tb*mdot1
print(mp1, 'Mass of Propellent, kg')
```

```
50679.13345369116 Mass of Propellent, kg
```

[88]:
```
# 5) Propellant tank size using LO2/LH2
R1 = 1/(np.exp(Deltav/(Isp1*g0)))
OF1 = 3.5
r1 = OF1
mdotox1 = (r1*mdot1)/(1+r1)
# mdotox = T/c
print(mdotox1, 'Mass Flow Rate of LO2')
mdotf1 = mdot1/(1+r1)
# mdotf = mdotox/OF
print(mdotf1, 'Mass Flow Rate of LCH4')

# density of oxidizer and methane
rhoox = 1141
rhoH2 = 71
Vdotox1 = mdotox1/rhoox
Vdotf1 = mdotf1/rhoH2
print(Vdotox1, 'Volumetric Flow Rate of LO2, kg/m^3')
print(Vdotf1, 'Volumetric Flow Rate of LCH4, kg/m^3')
Vox1 = Vdotox1 * tb
Vf1 = Vdotf1 * tb
print(Vox1, 'Volume of Oxidizer, kg/m^3')
print(Vf1, 'Volume of Fuel, kg/m^3')

Lcox1 = Vox1/Ac
print(Lcox1, 'Length of Oxidizer Tank, m')
Lcf1 = Vf1/Ac
print(Lcf1,'Length of Fuel Tank, m')
```

```
394.1710379731535 Mass Flow Rate of LO2
112.62029656375813 Mass Flow Rate of LCH4
0.3454610324041661 Volumetric Flow Rate of LO2, kg/m^3
1.5862013600529314 Volumetric Flow Rate of LCH4, kg/m^3
34.546103240416606 Volume of Oxidizer, kg/m^3
158.62013600529315 Volume of Fuel, kg/m^3
4.887273862466886 Length of Oxidizer Tank, m
22.440158861467673 Length of Fuel Tank, m
```

Solid Engine Analysis ME 499 Final Project

March 17, 2022

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

[11]:
```
Deltav = 2300
mo = 100000
MF = .9
tb = 100
# Thrust calc 100000 kg * g0 *2
T = 1962000
Dc = 3
# psi to Pa
Pc = 6.895e6
Pcpsi = 1000
a = 0.04
n = .3
g0 = 9.81
Ru = 8314.5
# in/s to m/s
rdot = (a*Pcpsi**n)*0.0254
print(rdot, 'Burn Time, m/s')
```

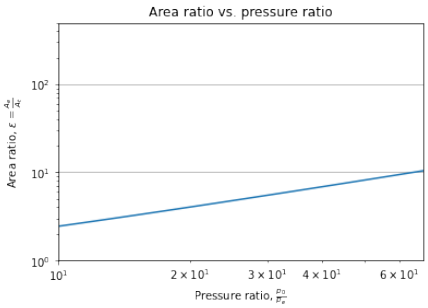
```
0.008070374864798698 Burn Time, m/s
```

[12]:
```
# 1) Specific Impulse calculation using Ammonium Perchlorate/Zirconium
# gamma for Ammonium Perchlorate/Zirconium
gammaAPZ = 1.125
Pa = 101325
pressure_ratio = Pc/Pa
print(pressure_ratio, 'Pressure Ratio')
def calc_thrust_coeff(gammaAPZ, pressure_ratio):
    ''' Calculates thrust coefficient for optimum expansion.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return np.sqrt(2 * np.power(gammaAPZ, 2) / (gammaAPZ - 1) *
        np.power(2 / (gammaAPZ + 1), (gammaAPZ + 1)/(gammaAPZ - 1)) *
```

```
                    (1 - np.power(1.0 / pressure_ratio, (gammaAPZ - 1)/
↪gammaAPZ)))

CF = calc_thrust_coeff(gammaAPZ, pressure_ratio)
print(CF, 'Thrust Coefficient')

gammas = [1.125]
pressure_ratios = np.logspace(1, 4, num=50)
labels = [r'$\gamma = 1.125$']
# let's define a function to calculate area ratio based on gamma and the␣
↪pressure ratio:
def calc_area_ratio(gammaAPZ, pressure_ratio):
    '''Calculates area ratio based on specific heat ratio and pressure ratio.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return (np.power(2 / (gammaAPZ + 1), 1/(gammaAPZ-1)) *
        np.power(pressure_ratio, 1 / gammaAPZ) *
        np.sqrt((gammaAPZ - 1) / (gammaAPZ + 1) /
                (1 - np.power(pressure_ratio, (1 - gammaAPZ)/gammaAPZ))))

for gamma, label in zip(gammas, labels):
    area_ratios = calc_area_ratio(gamma, pressure_ratios)
    plt.plot(pressure_ratios, area_ratios)
    plt.text(0.9*pressure_ratios[-10], 1.01*area_ratios[-10],
        label,
        horizontalalignment='right', fontsize=8)

plt.xlim([10, pressure_ratio])
plt.ylim([1, 500])
plt.xlabel(r'Pressure ratio, $\frac{p_0}{p_e}$')
plt.ylabel(r'Area ratio, $\epsilon = \frac{A_e}{A_t}$')
plt.grid(True)
plt.xscale('log')
plt.yscale('log')
plt.title('Area ratio vs. pressure ratio')
plt.show()

# Values From CEA
Tc = 4603.53
MW = 42.470
gamma = 1.125


def get_cstar(Tc, MW, gamma):
#     Calculates cstar using chamber properties
```

```
            return (np.sqrt(Ru * Tc / (gamma * MW)) * (2 / (gamma + 1))**(-0.
 ↪5*(gamma + 1)/(gamma - 1)))

Cstar = get_cstar(Tc, MW, gamma)
print(Cstar, 'Characteristic Velocity, m/s')

Isp = (Cstar*CF)/g0
print(Isp, 'Specific Impulse, s')
```

```
68.04835924006909 Pressure Ratio
1.6445116216936828 Thrust Coefficient
```




```
1498.4496170810166 Characteristic Velocity, m/s
251.19447602570648 Specific Impulse, s
```

[13]:
```
# 2) Engine and nozzle dimension calculations
# F = T
At = T/(Pc*CF)
print(At, 'Area of Throat, m^2')
Rt = np.sqrt(At/(np.pi))
print(Rt, 'Radius of Throat, m')
Dt = Rt*2
print(Dt, 'Diameter of Throat, m')

area_ratio = (np.power(2 / (gammaAPZ + 1), 1/(gammaAPZ-1)) * np.
 ↪power(pressure_ratio, 1 / gammaAPZ) *
    np.sqrt((gammaAPZ - 1) / (gammaAPZ + 1) /(1 - np.power(pressure_ratio, (1 -␣
 ↪gammaAPZ)/gammaAPZ))))
print(area_ratio, 'Area Ratio')
AeAt = area_ratio


Ae = AeAt*At
print(Ae, 'Area of Exit, m^2')
Re = np.sqrt(Ae/(np.pi))
```

```
print(Re, 'Radius of Exit, m')
De = Re*2
print(De, 'Diameter of Exit, m')
```

```
0.17303254103031834 Area of Throat, m^2
0.23468695839661122 Radius of Throat, m
0.46937391679322243 Diameter of Throat, m
10.39189784450689 Area Ratio
1.7981364901625152 Area of Exit, m^2
0.7565478316184319 Radius of Exit, m
1.5130956632368637 Diameter of Exit, m
```

[14]:
```
# 3) mass of propellent needed using Ammonium Perchlorate/Zirconium
c = Isp*g0
# F = T
mdot = T/c
mp = tb*mdot
print(mp, 'Mass of Propellent, kg')
```

```
79619.58525693558 Mass of Propellent, kg
```

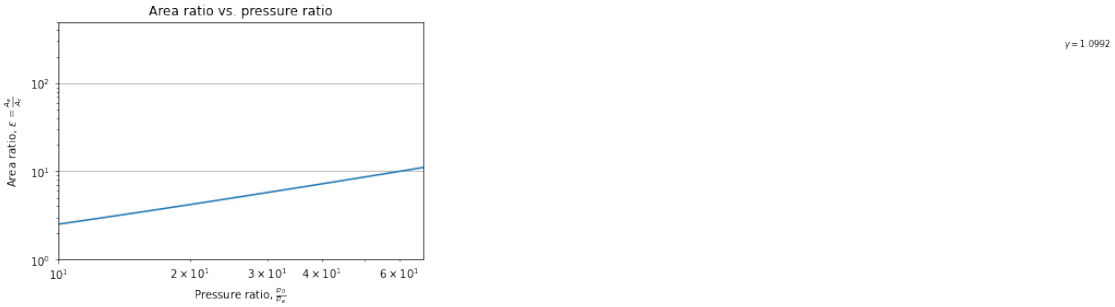
[29]:
```
# length and area of chamber calculation using Ammonium Perchlorate/Zirconium
Rc = Dc/2
rhoAmmoniumPerchlorate = 1950
rhoZirconium = 6520
rhotot = rhoAmmoniumPerchlorate + rhoZirconium
Vtot = mp/rhotot

Dweb = Rc/2
Rb = Rc-Dweb
Lc = (Ab-np.pi*(Rc**2-Rb**2))/(np.pi*Rb*2)
print(Lc, 'Length of Chamber, m')

# inital and final boundry area calculations
Abi = (np.pi*2*Rb*Lc)+((np.pi*(Rb**2-Rc**2)))
print(Abi, 'Surface Area of Burn Inital, m^2')
Abf = (np.pi*2*Rc*Lc)
print(Abf, 'Surface Area of Burn at Burnout, m^2')

Ac = np.pi*Rc**2
print(Ac, 'Area of Chamber, m^2')
```

```
1.3467335981489532 Length of Chamber, m
1.044894964498174 Surface Area of Burn Inital, m^2
12.6926651348619 Surface Area of Burn at Burnout, m^2
7.0685834705770345 Area of Chamber, m^2
```

[16]:
```
# 1) Specific Impulse calculation using Ammonium Perchlorate/Aluminum
# gamma for Ammonium Perchlorate/Aluminum
gammaAPA = 1.0992
Pa = 101325
pressure_ratio = Pc/Pa
print(pressure_ratio, 'Pressure Ratio')
def calc_thrust_coeff1(gammaAPA, pressure_ratio):
    ''' Calculates thrust coefficient for optimum expansion.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return np.sqrt(2 * np.power(gammaAPA, 2) / (gammaAPA - 1) *
        np.power(2 / (gammaAPA + 1), (gammaAPA + 1)/(gammaAPA - 1)) *
                   (1 - np.power(1.0 / pressure_ratio, (gammaAPA - 1)/
 ↪gammaAPA)))

CF1 = calc_thrust_coeff1(gammaAPA, pressure_ratio)
print(CF1, 'Thrust Coefficient')

gammas = [1.0992]
pressure_ratios = np.logspace(1, 4, num=50)
labels = [r'$\gamma = 1.0992$']
# let's define a function to calculate area ratio based on gamma and the␣
 ↪pressure ratio:
def calc_area_ratio1(gammaAPA, pressure_ratio):
    '''Calculates area ratio based on specific heat ratio and pressure ratio.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return (np.power(2 / (gammaAPA + 1), 1/(gammaAPA-1)) * np.
 ↪power(pressure_ratio, 1 / gammaAPA) *
        np.sqrt((gammaAPA - 1) / (gammaAPA + 1) /(1 - np.power(pressure_ratio,␣
 ↪(1 - gammaAPA)/gammaAPA))))

for gamma, label in zip(gammas, labels):
    area_ratios = calc_area_ratio1(gamma, pressure_ratios)
    plt.plot(pressure_ratios, area_ratios)
    plt.text(0.9*pressure_ratios[-10], 1.01*area_ratios[-10],
        label,
        horizontalalignment='right', fontsize=8)

plt.xlim([10, pressure_ratio])
plt.ylim([1, 500])
plt.xlabel(r'Pressure ratio, $\frac{p_0}{p_e}$')
plt.ylabel(r'Area ratio, $\epsilon = \frac{A_e}{A_t}$')
plt.grid(True)
plt.xscale('log')
```

```
plt.yscale('log')
plt.title('Area ratio vs. pressure ratio')
plt.show()

# Values From CEA
Tc1 = 4304.18
MW1 = 46.358
gamma1 = 1.0992

def get_cstar1(Tc1, MW1, gamma1):
#     Calculates cstar using chamber properties

            return (np.sqrt(Ru * Tc1 / (gamma1 * MW1)) * (2 / (gamma1 +␣
 ↪1))**(-0.5*(gamma1 + 1)/(gamma1 - 1)))

Cstar1 = get_cstar1(Tc1, MW1, gamma1)
print(Cstar1, 'Characteristic Velocity, m/s')

Isp1 = (Cstar1*CF1)/g0
print(Isp1, 'Specific Impulse, s')
```

```
68.04835924006909 Pressure Ratio
1.664310363958975 Thrust Coefficient
```




```
1398.6471889786378 Characteristic Velocity, m/s
237.28674945354066 Specific Impulse, s
```

[17]:
```
# 2) Engine and nozzle dimension calculations
# F = T
At1 = T/(Pc*CF1)
print(At1, 'Area of Throat, m^2')
Rt1 = np.sqrt(At1/(np.pi))
print(Rt1, 'Radius of Throat, m')
Dt1 = Rt1*2
print(Dt1, 'Diameter of Throat, m')
```

```
area_ratio1 = (np.power(2 / (gammaAPA + 1), 1/(gammaAPA-1)) * np.
 ↪power(pressure_ratio, 1 / gammaAPA) *
    np.sqrt((gammaAPA - 1) / (gammaAPA + 1) /(1 - np.power(pressure_ratio, (1 -␣
 ↪gammaAPA)/gammaAPA))))
print(area_ratio1, 'Area Ratio')
AeAt1 = area_ratio1

Ae1 = AeAt1*At1
print(Ae1, 'Area of Exit, m^2')
Re1 = np.sqrt(Ae1/(np.pi))
print(Re1, 'Radius of Exit, m')
De1 = Re1*2
print(De1, 'Diameter of Exit, m')
```

```
0.17097413488351126 Area of Throat, m^2
0.23328685649890898 Radius of Throat, m
0.46657371299781797 Diameter of Throat, m
11.024571885084223 Area Ratio
1.884916640513356 Area of Exit, m^2
0.7745886658786968 Radius of Exit, m
1.5491773317573936 Diameter of Exit, m
```

[18]:
```
# 3) mass of propellent needed using Ammonium Perchlorate/Aluminum
c1 = Isp1*g0
# F = T
mdot1 = T/c1
mp1 = tb*mdot1
print(mp1, 'Mass of Propellent, kg')
```

```
84286.20665106241 Mass of Propellent, kg
```

[26]:
```
# length and area of chamber calculation using Ammonium Perchlorate/Aluminum
Rc = Dc/2
rhoAmmoniumPerchlorate = 1950
rhoAluminum = 2700
rhotot1 = rhoAmmoniumPerchlorate + rhoAluminum
Vtot1 = mp1/rhotot1

Dweb = Rc/2
Rb = Rc-Dweb
Lc1 = (Ab1-np.pi*(Rc**2-Rb**2))/(np.pi*Rb*2)
print(Lc1, 'Length of Chamber, m')

# inital and final boundry area calculations
Abi1 = (np.pi*2*Rb*Lc1)+((np.pi*(Rb**2-Rc**2)))
print(Abi1, 'Surface Area of Burn Inital, m^2')
```

```
Abf1 = (np.pi*2*Rc*Lc1)
print(Abf1, 'Surface Area of Burn at Burnout, m^2')

Ac = np.pi*Rc**2
print(Ac, 'Area of Chamber, m^2')
```

```
3.641161080939318 Length of Chamber, m
11.857129750691271 Surface Area of Burn Inital, m^2
34.317134707248094 Surface Area of Burn at Burnout, m^2
7.0685834705770345 Area of Chamber, m^2
```

[ ]:

Hybrid Engine Analysis ME 499 Final Project

March 17, 2022

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

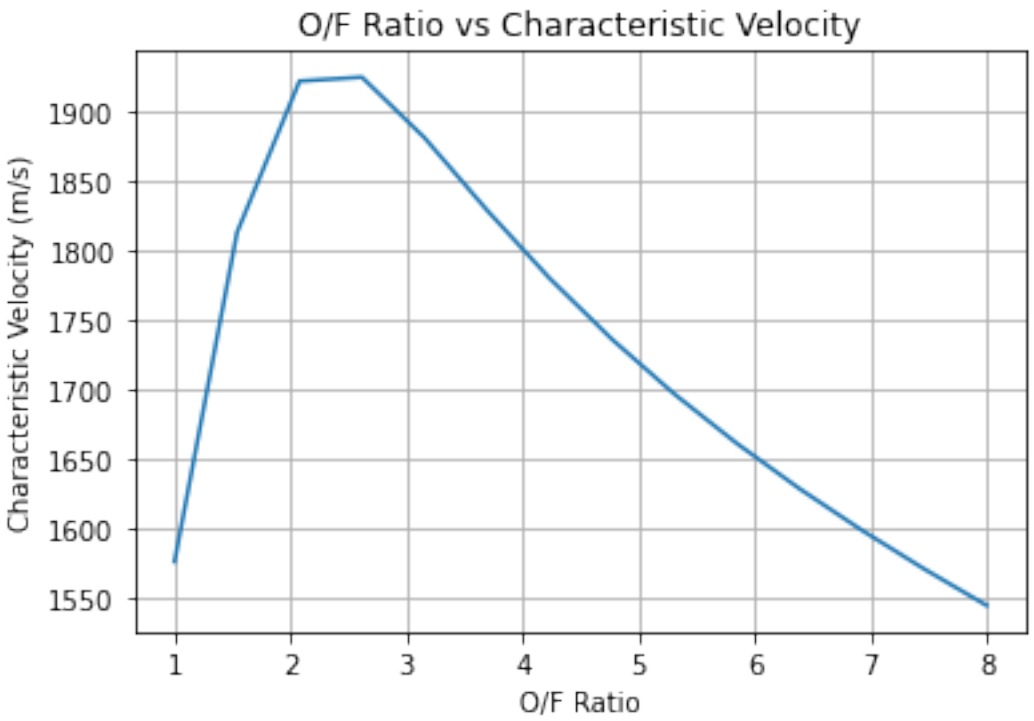
[3]:
```
Deltav = 2300
mo = 100000
MF = .9
tb = 100
# Thrust calc 100000 kg * g0 *2
T = 1962000
Dmax = 3
# psi to Pa
Pc = 6.895e6
# density in kg/m^3
rhoHTPB = 915
rhoN2O = 1226
Pa = 101325
Ru = 8314.5
g0 = 9.81
# convert units
a = 1.1e-5
n = .68
```

[4]:
```
# Part 1 Propellent and Design Concept 1 using LOX/Polybutadiene
# 1) Optimal O/F ratio determined for specific propellant combination, plots␣
 ↪generated of relationship.
# values from CEA rocket problem analysis
Tc = [1674.66,2571.52,3281.57,3622.18,3725.01,3728.35,3695.36,3647.78,3593.
 ↪64,3536.26,3477.15,3417.03,3356.27,3295.08]
MW = [12.796,15.012,17.739,19.957,21.652,22.983,24.076,24.997,25.785,26.464,27.
 ↪054,27.568,28.017,28.409]
gamma = [1.2714,1.2507,1.1826,1.1472,1.1352,1.1314,1.1299,1.1294,1.1296,1.
 ↪1302,1.1313,1.1328,1.1348,1.1372]

#OF values from 1 to 8 in increments of .5
OF = np.linspace(1,8,14)
```

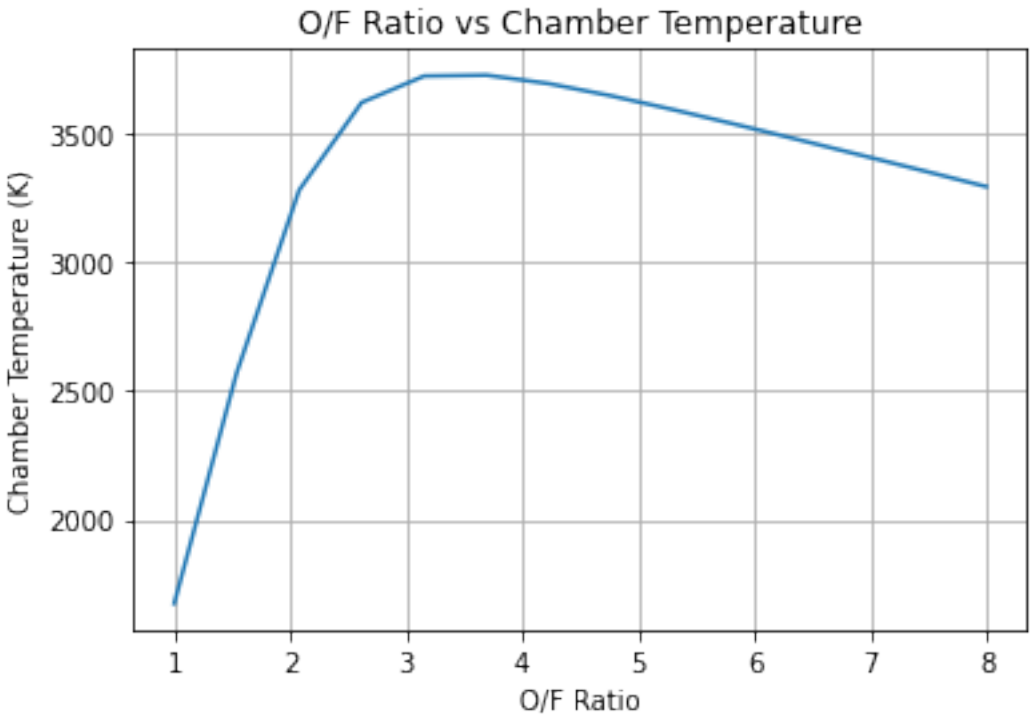
```
def get_cstar(Tc, MW, gamma):
#     Calculates cstar using chamber properties

            return (np.sqrt(Ru * Tc / (gamma * MW)) * (2 / (gamma + 1))**(-0.
 ↪5*(gamma + 1)/(gamma - 1)))

Cstar0 = get_cstar(Tc[0], MW[0], gamma[0])
Cstar1 = get_cstar(Tc[1], MW[1], gamma[1])
Cstar2 = get_cstar(Tc[2], MW[2], gamma[2])
Cstar3 = get_cstar(Tc[3], MW[3], gamma[3])
Cstar4 = get_cstar(Tc[4], MW[4], gamma[4])
Cstar5 = get_cstar(Tc[5], MW[5], gamma[5])
Cstar6 = get_cstar(Tc[6], MW[6], gamma[6])
Cstar7 = get_cstar(Tc[7], MW[7], gamma[7])
Cstar8 = get_cstar(Tc[8], MW[8], gamma[8])
Cstar9 = get_cstar(Tc[9], MW[9], gamma[9])
Cstar10 = get_cstar(Tc[10], MW[10], gamma[10])
Cstar11 = get_cstar(Tc[11], MW[11], gamma[11])
Cstar12 = get_cstar(Tc[12], MW[12], gamma[12])
Cstar13 = get_cstar(Tc[13], MW[13], gamma[13])

Cstar =␣
 ↪[Cstar0,Cstar1,Cstar2,Cstar3,Cstar4,Cstar5,Cstar6,Cstar7,Cstar8,Cstar9,Cstar10,Cstar11,Csta

Cstarmax =  np.max(Cstar)
OFMax = OF[Cstar.index(Cstarmax)]
GammaMAX = gamma[Cstar.index(Cstarmax)]

# these are the values at the optimized maximum Characteristic Velocity
print(OFMax, 'Optimal Oxidizer to Fuel Ratio at Max Characteristic Velocity')
print(Cstarmax, ' (m/s) Max Characteristic Velocity at Optimal Oxidizer to Fuel␣
 ↪Ratio')
print(GammaMAX, 'Max Gamma at Optimal Oxidizer to Fuel Ratio')
```

```
2.6153846153846154 Optimal Oxidizer to Fuel Ratio at Max Characteristic Velocity
1925.2405883741756  (m/s) Max Characteristic Velocity at Optimal Oxidizer to
Fuel Ratio
1.1472 Max Gamma at Optimal Oxidizer to Fuel Ratio
```

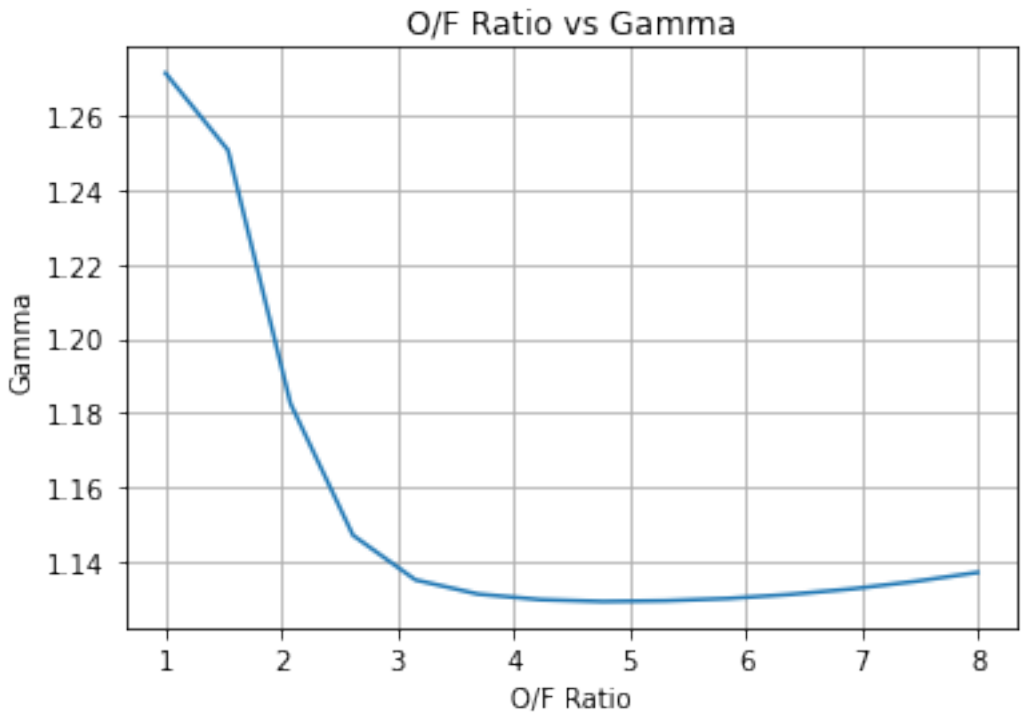
[5]:
```
plt.plot(OF, Cstar)
plt.title('O/F Ratio vs Characteristic Velocity')
plt.xlabel('O/F Ratio')
plt.ylabel('Characteristic Velocity (m/s)')
plt.grid()
plt.show()
```



[6]:
```
plt.plot(OF, Tc)
plt.title('O/F Ratio vs Chamber Temperature')
plt.xlabel('O/F Ratio')
plt.ylabel('Chamber Temperature (K)')
plt.grid()
plt.show()
```



[7]:
```
plt.plot(OF, gamma)
plt.title('O/F Ratio vs Gamma')
plt.xlabel('O/F Ratio')
plt.ylabel('Gamma')
plt.grid()
plt.show()
```



[8]:
```
# 2) Specific Impulse of Propellant combination using LOX/Polybutadiene
# cstarmax and gammamax used becuase of optimal O/F ratio
gamma = GammaMAX
pressure_ratio = Pc/Pa
def calc_thrust_coeff(gamma , pressure_ratio):
    ''' Calculates thrust coefficient for optimum expansion.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return np.sqrt(2 * np.power(gamma, 2) / (gamma - 1) *
        np.power(2 / (gamma + 1), (gamma + 1)/(gamma - 1)) * (1 - np.power(1.0 /
 ↪ pressure_ratio, (gamma - 1)/gamma)))

CF = calc_thrust_coeff(gamma, pressure_ratio)
Isp = (Cstarmax*CF)/g0
print(Isp, 'Specific Impulse, s')
```

```
319.68498010275863 Specific Impulse, s
```

[15]:
```
# 3) Engine and nozzle dimension calculations
At = T/(Pc*CF)
print(At, 'Area of Throat, m^2')
Rt = np.sqrt(At/(np.pi))
```

```
print(Rt, 'Radius of Throat, m')
Dt = Rt*2
print(Dt, 'Diameter of Throat, m')

gamma = GammaMAX
pressure_ratio = Pc/Pa
# Area Exit over area throat ratio function calculation
def get_area_ratio(pressure_ratio, gamma):
    '''Calculates area ratio based on specific heat ratio and pressure ratio.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return (np.power(2 / (gamma + 1), 1/(gamma-1)) * np.power(pressure_ratio, 1␣
↪/ gamma) *
        np.sqrt((gamma - 1) / (gamma + 1) /(1 - np.power(pressure_ratio, (1 -␣
↪gamma)/gamma))))

AeAt = get_area_ratio(pressure_ratio, gamma)
print(AeAt, 'Area Ratio')

Ae = AeAt*At
print(Ae, 'Area of Exit, m^2')
Re = np.sqrt(Ae/(np.pi))
print(Re, 'Radius of Exit, m')
De = Re*2
print(De, 'Diameter of Exit, m')

rhoox = 1141
rhof = 920
T = 100000*2*g0
g0 = 9.81
c = Isp*g0
mdotox = T/c
mdotfuel = mdotox/OFMax


Rweb = .75
Rpi = Dmax/2 - Rweb
Rpf = Dmax/2
Api = np.pi*Rpi**2
Apf = np.pi*Rpf**2
print(Api, 'Inital Area of Port, m^2')
print(Apf, 'Final Area of Port, m^2')

Goi2 = (mdotox/Api)
Goi21 = (mdotox/Apf)
```

```
rdot2 = a*Goi2**n
print(rdot2, 'Inital Burn Rate, m/s')
rdot21 = a*Goi21**n
print(rdot21, ' Final Burn Rate, m/s')
```

```
0.17468616156714545 Area of Throat, m^2
0.23580570859570246 Radius of Throat, m
0.4716114171914049 Diameter of Throat, m
9.896441509578565 Area Ratio
1.7287713804820461 Area of Exit, m^2
0.7418119851815788 Radius of Exit, m
1.4836239703631575 Diameter of Exit, m
1.7671458676442586 Inital Area of Port, m^2
7.0685834705770345 Final Area of Port, m^2
0.0005953011653689245 Inital Burn Rate, m/s
0.00023191879114304188  Final Burn Rate, m/s
```

[14]:
```
# 4) mass of propellent needed using LOX/Polybutadiene
c = Isp*g0
# F = T
mdot = mdotox + mdotfuel
mp = tb*mdot
print(mp, 'Mass of Propellent, kg')

mpf = mdotfuel*tb
# Using the Equation mp = mpox + mpf
mpox = mp - mpf
Vf = mpf/rhof

# length of chamber calculation
Lc = Vf/(np.pi*(Rpf**2-Rpi**2))
print(Lc, 'Length of Fuel Grain, m')

# inital and final boundry area calculations
Abi = (np.pi*2*Rpi*Lc)+((np.pi*(Rpf**2-Rpi**2)))
print(Abi, 'Surface Area of Burn Inital, m^2')
Abf = (np.pi*2*Rpf*Lc)
print(Abf, 'Surface Area of Burn at Burnout, m^2')
```

```
86482.19511170847 Mass of Propellent, kg
4.904454584275464 Length of Fuel Grain, m
28.413135340669648 Surface Area of Burn Inital, m^2
46.223395475473744 Surface Area of Burn at Burnout, m^2
```

[23]:
```
# 5) Propellant tank size using LOX/Polybutadiene
r = OFMax
mdotox = (r*mdot)/(1+r)
```

```
mdotf = mdot/(1+r)

Vdotox = mdotox/rhoox
vdotf = mdotf/rhof
print(Vdotox, 'Volumetric Flow Rate of Oxidizer, kg/m^3')
Vox = Vdotox * tb
Vf = vdotf * tb
print(Vox, 'Volume of Oxidizer, kg/m^3')
print(Vf, 'Volume of Fuel, kg/m^3')

Atankox = Vox/Lc
print(Atankox, 'Area of Oxidizer tank, m')
Rtank = np.sqrt(Atankox/(np.pi))
print(Rtank, 'Radius of oxidizer Tank, m')
Atankf = Vf/Lc
print(Atankf, 'Area of fuel tank, m')
Rtankf = np.sqrt(Atankf/(np.pi))
print(Rtankf, 'Radius of fuel Tank, m')
```

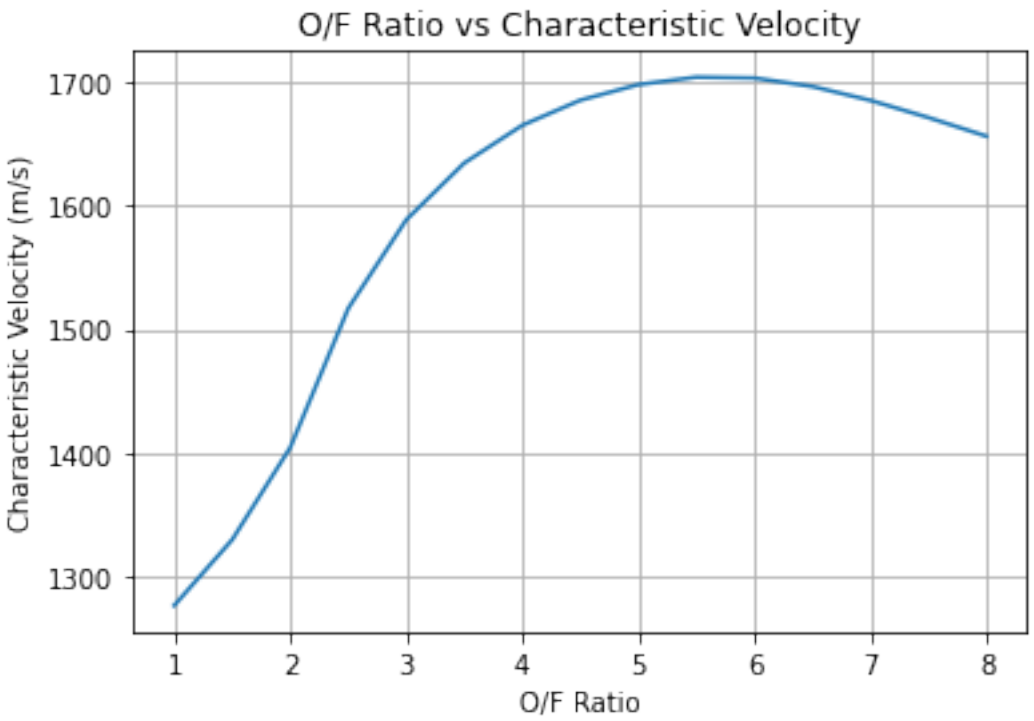
```
0.5483048900363787 Volumetric Flow Rate of Oxidizer, kg/m^3
54.83048900363787 Volume of Oxidizer, kg/m^3
26.00065995495398 Volume of Fuel, kg/m^3
11.179732233515624 Area of Oxidizer tank, m
1.8864303047861626 Radius of oxidizer Tank, m
5.3014376029327765 Area of fuel tank, m
1.299038105676658 Radius of fuel Tank, m
```

[17]:
```
# Part 2 Propellent and Design Concept 2 using Hydrogen Peroxide/Polyethylene
# 1) Optimal O/F ratio determined for specific propellant combination, plots␣
 ↪generated of relationship.
# values from CEA rocket problem analysis
Tc1 = [1276.01,1331.95,1547.41,1937.77,2244.97,2484.37,2671.52,2815.80,2922.
 ↪34,2993.43,3031.23,3040.83,3029.85,3005.49,2973.02]
MW1 = [15.900,14.923,14.954,16.197,17.327,18.313,19.171,19.913,20.547,21.072,21.
 ↪492,21.815,22.060,22.245,22.385]
gamma1 = [1.1556,1.1957,1.2626,1.2466,1.2248,1.2062,1.1893,1.1736,1.1589,1.
 ↪1458,1.1358,1.1300,1.1277,1.1279,1.1294]

#OF values from 1 to 8 in increments of .5
OF1 = np.linspace(1,8,15)


def get_cstar1(Tc1, MW1, gamma1):
#     Calculates cstar using chamber properties

            return (np.sqrt(Ru * Tc1 / (gamma1 * MW1)) * (2 / (gamma1 +␣
 ↪1))**(-0.5*(gamma1 + 1)/(gamma1 - 1)))
```

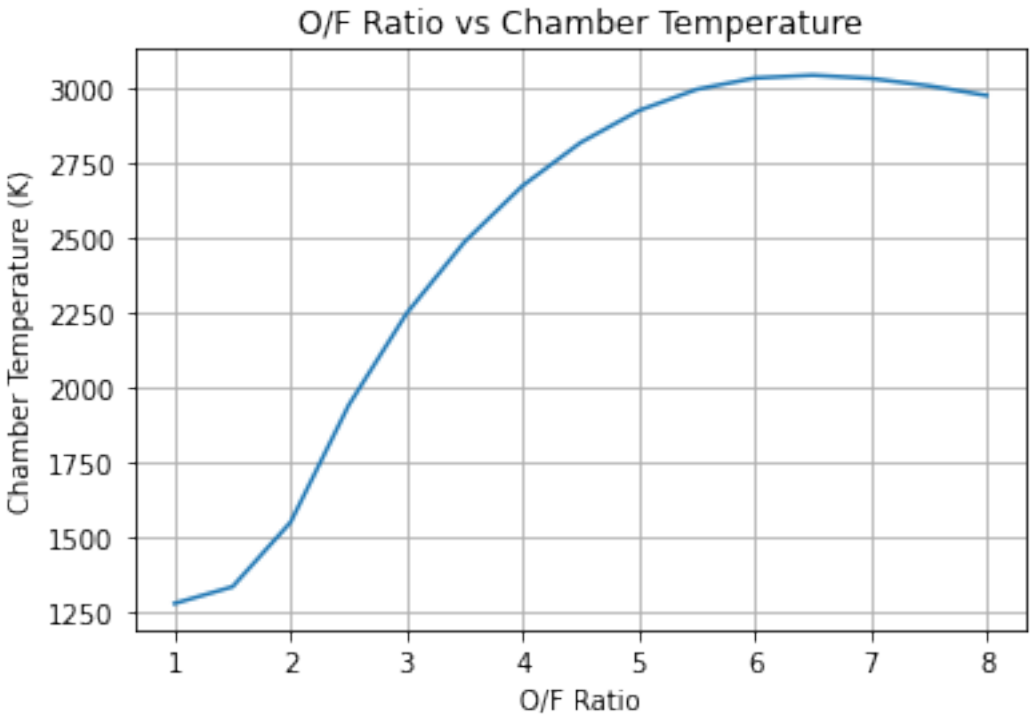
```
Cstar10 = get_cstar1(Tc1[0], MW1[0], gamma1[0])
Cstar11 = get_cstar1(Tc1[1], MW1[1], gamma1[1])
Cstar12 = get_cstar1(Tc1[2], MW1[2], gamma1[2])
Cstar13 = get_cstar1(Tc1[3], MW1[3], gamma1[3])
Cstar14 = get_cstar1(Tc1[4], MW1[4], gamma1[4])
Cstar15 = get_cstar1(Tc1[5], MW1[5], gamma1[5])
Cstar16 = get_cstar1(Tc1[6], MW1[6], gamma1[6])
Cstar17 = get_cstar1(Tc1[7], MW1[7], gamma1[7])
Cstar18 = get_cstar1(Tc1[8], MW1[8], gamma1[8])
Cstar19 = get_cstar1(Tc1[9], MW1[9], gamma1[9])
Cstar110 = get_cstar1(Tc1[10], MW1[10], gamma1[10])
Cstar111 = get_cstar1(Tc1[11], MW1[11], gamma1[11])
Cstar112 = get_cstar1(Tc1[12], MW1[12], gamma1[12])
Cstar113 = get_cstar1(Tc1[13], MW1[13], gamma1[13])
Cstar114 = get_cstar1(Tc1[14], MW1[14], gamma1[14])

Cstar1 =␣
 ↪[Cstar10,Cstar11,Cstar12,Cstar13,Cstar14,Cstar15,Cstar16,Cstar17,Cstar18,Cstar19,Cstar110,C

Cstarmax1 =  np.max(Cstar1)
OFMax1 = OF1[Cstar1.index(Cstarmax1)]
GammaMAX1 = gamma1[Cstar1.index(Cstarmax1)]

# these are the values at the optimized maximum Characteristic Velocity
print(OFMax1, 'Optimal Oxidizer to Fuel Ratio at Max Characteristic Velocity')
print(Cstarmax1, ' (m/s) Max Characteristic Velocity at Optimal Oxidizer to␣
 ↪Fuel Ratio')
print(GammaMAX1, 'Max Gamma at Optimal Oxidizer to Fuel Ratio')
```

```
5.5 Optimal Oxidizer to Fuel Ratio at Max Characteristic Velocity
1704.009358380236  (m/s) Max Characteristic Velocity at Optimal Oxidizer to Fuel
Ratio
1.1458 Max Gamma at Optimal Oxidizer to Fuel Ratio
```

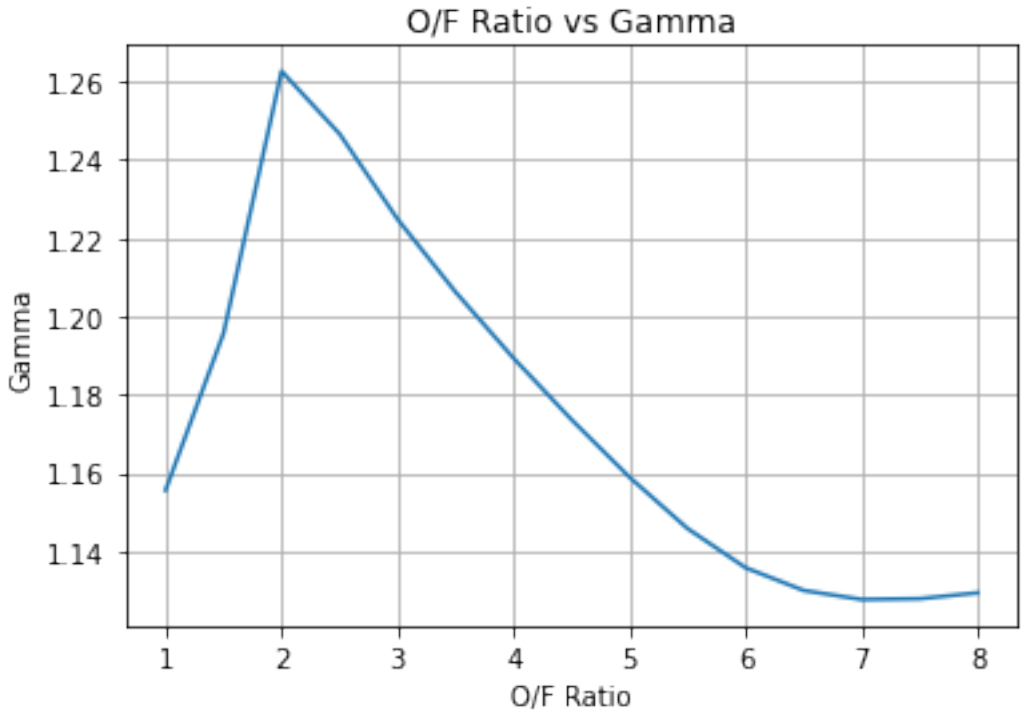
[18]:
```
plt.plot(OF1, Cstar1)
plt.title('O/F Ratio vs Characteristic Velocity')
plt.xlabel('O/F Ratio')
plt.ylabel('Characteristic Velocity (m/s)')
plt.grid()
plt.show()
```



[19]:
```
plt.plot(OF1, Tc1)
plt.title('O/F Ratio vs Chamber Temperature')
plt.xlabel('O/F Ratio')
plt.ylabel('Chamber Temperature (K)')
plt.grid()
plt.show()
```



[20]:
```
plt.plot(OF1, gamma1)
plt.title('O/F Ratio vs Gamma')
plt.xlabel('O/F Ratio')
plt.ylabel('Gamma')
plt.grid()
plt.show()
```



[21]:
```
# 2) Specific Impulse of Propellant combination using Hydrogen Peroxide/
 ↪Polyethylene
# cstarmax and gammamax used becuase of optimal O/F ratio
gamma1 = GammaMAX1
pressure_ratio1 = Pc/Pa
def calc_thrust_coeff1(gamma1 , pressure_ratio1):
    ''' Calculates thrust coefficient for optimum expansion.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return np.sqrt(2 * np.power(gamma1, 2) / (gamma1 - 1) *
        np.power(2 / (gamma1 + 1), (gamma1 + 1)/(gamma1 - 1)) * (1 - np.power(1.
 ↪0 / pressure_ratio1, (gamma1 - 1)/gamma1)))

CF1 = calc_thrust_coeff1(gamma1, pressure_ratio1)
Isp1 = (Cstarmax1*CF1)/g0
print(Isp1, 'Specific Impulse, s')
```

```
283.11370457425147 Specific Impulse, s
```

[22]:
```
# 3) Engine and nozzle dimension calculations
At1 = T/(Pc*CF1)
print(At1, 'Area of Throat, m^2')
```

```
Rt1 = np.sqrt(At1/(np.pi))
print(Rt, 'Radius of Throat, m')
Dt1 = Rt1*2
print(Dt1, 'Diameter of Throat, m')

gamma1 = GammaMAX1
pressure_ratio1 = Pc/Pa
# Area Exit over area throat ratio function calculation
def get_area_ratio1(pressure_ratio1, gamma1):
    '''Calculates area ratio based on specific heat ratio and pressure ratio.
    pressure ratio: chamber / exit
    area ratio: exit / throat
    '''
    return (np.power(2 / (gamma1 + 1), 1/(gamma1-1)) * np.
 ↪power(pressure_ratio1, 1 / gamma1) *
        np.sqrt((gamma1 - 1) / (gamma1 + 1) /(1 - np.power(pressure_ratio1, (1␣
 ↪- gamma1)/gamma1))))

AeAt1 = get_area_ratio1(pressure_ratio1, gamma1)
print(AeAt1, 'Area Ratio')

Ae1 = AeAt1*At1
print(Ae1, 'Area of Exit, m^2')
Re1 = np.sqrt(Ae1/(np.pi))
print(Re1, 'Radius of Exit, m')
De1 = Re1*2
print(De1, 'Diameter of Exit, m')

rhoox1 = 1450
rhof1 = 960
T = 100000*2*g0
g0 = 9.81
c = Isp1*g0
mdotox1 = T/c
print(mdotox1, 'Mass Flow Rate of Oxidizer, kg/s')
mdotfuel1 = mdotox1/OFMax1
print(mdotfuel1,'Mass Flow Rate of Fuel, kg/s')

Rweb = .75
Rpi = Dmax/2 - Rweb
Rpf = Dmax/2
Api1 = np.pi*Rpi**2
Apf1 = np.pi*Rpf**2
print(Api1, 'Inital Area of Port, m^2')
print(Apf1, 'Final Area of Port, m^2')

Goi = (mdotox1/Api1)
```

```
Goi1 = (mdotox1/Apf1)

rdot = a*Goi**n
print(rdot, 'Inital Burn Rate, m/s')
rdot1 = a*Goi1**n
print(rdot1, ' Final Burn Rate, m/s')
```

```
0.1745849521155513 Area of Throat, m^2
0.23580570859570246 Radius of Throat, m
0.47147477657793607 Diameter of Throat, m
9.926453241324698 Area Ratio
1.7330093638139314 Area of Exit, m^2
0.7427206832659611 Radius of Exit, m
1.4854413665319222 Diameter of Exit, m
706.4299494111791 Mass Flow Rate of Oxidizer, kg/s
128.44180898385073 Mass Flow Rate of Fuel, kg/s
1.7671458676442586 Inital Area of Port, m^2
7.0685834705770345 Final Area of Port, m^2
0.0006465683527394893 Inital Burn Rate, m/s
0.000251891579392023  Final Burn Rate, m/s
```

[24]:
```
# 4) mass of propellent needed and length using Hydrogen Peroxide/Polyethylene
c1 = Isp1*g0
# F = T
mdot1 = mdotox1 + mdotfuel1
mp1 = tb*mdot1
print(mp1, 'Mass of Propellent, kg')

mpf1 = mdotfuel1*tb
print(mpf1, 'Mass of Fuel, kg')
# Using the Equation mp = mpox + mpf
mpox1 = mp1 - mpf1
print(mpox1, 'Mass of Oxidizer, kg')
Vf1 = mpf1/rhof1
print(Vf1, 'Volume of Fuel, kg/m^3')

# length of chamber calculation
Lc1 = Vf1/(np.pi*(Rpf**2-Rpi**2))
print(Lc1, 'Length of Fuel Grain, m')

# inital and final boundry area calculations
Abi1 = (np.pi*2*Rpi*Lc1)+((np.pi*(Rpf**2-Rpi**2)))
print(Abi1, 'Surface Area of Burn Inital, m^2')
Abf1 = (np.pi*2*Rpf*Lc1)
print(Abf1, 'Surface Area of Burn at Burnout, m^2')
```

```
83487.17583950298 Mass of Propellent, kg
12844.180898385073 Mass of Fuel, kg
```

```
70642.9949411179 Mass of Oxidizer, kg
13.379355102484451 Volume of Fuel, kg/m^3
2.523722074005538 Length of Fuel Grain, m
17.194197694030066 Surface Area of Burn Inital, m^2
23.78552018219458 Surface Area of Burn at Burnout, m^2
```

[32]:
```
# 5) Propellant tank size using Hydrogen Peroxide/Polyethylene
r1 = OFMax1
mdotox11 = (r1*mdot1)/(1+r1)
mdotf1 = mdot1/(1+r1)

Vdotox1 = mdotox1/rhoox1
vdotf1 = mdotf1/rhof1
print(Vdotox1, 'Volumetric Flow Rate of Oxidizer, kg/m^3')
Vox1 = Vdotox1 * tb
Vf1 = vdotf1 * tb
print(Vox1, 'Volume of Oxidizer, kg/m^3')
print(Vf1, 'Volume of Fuel, kg/m^3')

Atankox1 = Vox1/Lc1
print(Atankox1, 'Area of Oxidizer tank, m')
Rtank1 = np.sqrt(Atankox1/(np.pi))
print(Rtank1, 'Radius of oxidizer Tank, m')
Atankf1 = Vf1/Lc1
print(Atankf1, 'Area of fuel tank, m')
Rtankf1 = np.sqrt(Atankf1/(np.pi))
print(Rtankf1, 'Radius of fuel Tank, m')
```

```
0.4871930685594339 Volumetric Flow Rate of Oxidizer, kg/m^3
48.71930685594339 Volume of Oxidizer, kg/m^3
13.379355102484453 Volume of Fuel, kg/m^3
19.30454520240349 Area of Oxidizer tank, m
2.478876274888865 Radius of oxidizer Tank, m
5.3014376029327765 Area of fuel tank, m
1.299038105676658 Radius of fuel Tank, m
```