Design Optimization for a Student-Built Sub-Orbital Rocket

Ondrej Fercak
Erin S. Schmidt
Ian Zabel

words words words

Nomenclature

$M_0^{+\circlearrowleft}$ Moment in the x-axis (N m)
T^0 Torque (N m)
θ Angular position (°)

I. Introduction

There is an emerging demand by both governments and private industry for small 'venture class' launch vehicles to deliver nano-satellites into Low Earth Orbit (LEO). While there are yet few operational examples of such dedicated small satellite launch vehicles, they would likely share cladistic similarities, at the extrema of their size and mass envelope range, with both large orbital launch vehicles and the comparatively small high-powered rockets that have been operated by hundreds of amateur and university groups for decades. Typically high-powered rockets fly ballistic trajectories with apogees generally less than 10 km above sea-level. However, several amateur and university groups harbor aspirations of sub-orbital flight above the von Karman line 100 km above the surface of the Earth. As of the Spring of 2016 the current record holder for altitude at apogee by a student organization is TU Delft's Delft Aerospace Rocket Engineering (DARE) team which reached 21.457 km with their Stratos II+ rocket on October 16, 2015 (cite).

The Portland State Aerospace Society (PSAS) is an engineering student organization and citizen science project located at Portland State University dedicated to developing low-cost, open-source, and open-hardware high-powered rockets and avionics systems with special interests in small launch vehicle technology and nanosatellites.[1] In 2015 PSAS initiated a project to build and fly it's own entry in this rapidly intensifying 'university space race'.

Herein, we apply design optimization methodology to the problem of design and trajectory optimization of small sounding rockets, and particularly to the PSAS's LV4 'space rocket'. This rocket will leverage powerful liquid-fuel propulsion, an extremely light-weight carbon-composite airframe, full 6-DoF attitude control and active stabilization. LV4 is intended to fly with a design apogee of over 100 km, and is currently planned for launch by 2021. A SolidWorks CAD render of a concept for LV4 is shown in Figure **??**.(LV4 figure)

II. Methods

A. Problem Definition

Clean-sheet conceptual design and trajectory optimization of launxh vehicles is a classically difficult problem. This problem arises for two reasons: that the trajectory equation a 2nd order non-linear partial differential equation having no close form solution, and detailed design choices in propulsion, structures/weights, aerodynamics, and guidance and control which ultimately all appear as variables in this equation are both highly coupled and non-hierarchical. In the past the traditional approach to dealing with this problem has been

to evaluate vehicle performance by examining the value of each parameter by fixing the values of the remaining parameters e.g. the "one variable-at-a-time" trade-off analysis approach. However there are several important limitations to this approach:

- Conceptual design is usually carried out with low-fidelity models
- Some relationships among the design variables are poorly understood
- Optimizing individual design variables does not guarantee optimality at the overall system level

In practice this results in a highly iterative design process and concomitant requirement mismatches, developmental dead-ends and sub-optimal final design. The problem with suboptimal design is magnified by the ramifications of Konstantin Tsiolkovsy's equation, with the severe implication of exponential growth in design requirements for linear increases in rocket dry mass. Since ultimately all design decisions impact the rocket dry mass in some way it is imperative to understand these trade-offs and compromises as ealy in the design process as possible to reduce the potential for technical, schedule and cost risks. This is especially the case for time, technical expertise and funding constrained student organizations. Therefore there is a strong motivation to treat the conceptual design parameters "all-at-once" using applied optimization techniques. A numerical design optimization approach allows us o systematically explore a vast trade space under a realistic timeframe.

Some commercial and/or governmental tools exist for launch vehicle design optimization. These include codes such as FASTPASS,[2] and SWORD.[3] There are also open-source design and optimization tools that can be applied to high-powered rocketry such as Open Rocket(cite), or JSBSim (cite), however these tools either cannot be run in a batch mode or lack I/O tools to support direct numerical optimization. In the context of this developing interest in clean-sheet small launch vehicle designs we can identify a need for a design tool high-level enough for simplicity, speed, and ease of use, but which captures enough of the dimensions of the optimization problem to still be useful as a guide in the early conceptual design phase. This tool will use fairly low-fidelity models, and will be used for trajectory optimization, propulsion design, airframe sizing, and mass estimation. Per PSASs open-source mandate all (non-ITAR) project development deliverables are being made publicly available under a GNU GPL v2 license.[7] This code was written in Python because it is free and open-source, and for its inherent object-oriented modularity, numerical efficiency, and wide use in the community of scientific computing. It is hoped by the authors, and the members of PSAS, that the discourse around educational launch vehicles and their design will be elevated by making the information for this project publicly available.

B. Objectives and Constraints

The objective of the optimization is to minimize the total system complexity and cost of the 100 km launch vehicle. While these objectives can be abstract numerical values in the early conceptual design phase, we expect them to scale closely with the initial mass of the launch vehicle (including loaded propellants), which is usually defined by the Gross Lift-Off Weight (GLOW) figure of merit.

Thus we wish to minimize GLOW subject to practical model linearity and structural constraints. In the most practically usefully sense the design variables include thrust, airframe diameter and propellant tank lengths. However for numerical simplicity thrust will be expanded into mass flow rate and expansion ratio design variables, and then backsolved within the simulation. There are also a large number of design constants present in the models, which include specific impulse and combustion chamber pressure, propellant mixture ratio, and others, but which for the sake of brevity will be largely ignored in the following discussion. The model currently include 5 inequality constraints:

- Apogee: 100 km
- TWR: Trade-off between gravity loss and aerodynamic stability
- L/D Ratio: Trade-off between aerodynamic stability and mechanical (non-rigid body) resonance modes
- Maximum acceleration: Set by the material limits of various launch vehicle subsystems
- Nozzle over-expansion, checks that the assumption made to help linearize thrust model is valid

Mathematically the problem can be stated as

Design variables: $m_{wet} = f(r, L)$, $m_{dry} = f(F)$, $mdot$, $radius$, p_e, $p_c h$ Constraints: $L/D < 15$, $TWR > 1$, Sommerfield criterion $(p_e/p_a) > 0.35$, $3 < radius < 7$, $x >= 100000$, $a/g_0 <= 15$

C. Trajectory Simulation

Forward Euler integration of the rocket trajectory, using finite difference discretizations of the governing equation of motion, which is essentially Newton's 2nd law.

$$F = \frac{d(mv)}{dt}$$

$$\frac{d(mv)}{dt} = Thrust(x) - Drag(x, \frac{dx}{dt}, \frac{dx}{dt}^2) - mg_0$$

discretizations for the governing diff eq.

Instantaneous mass is determined from the mass flow rate, and the initial propellant tankage volume (determined from the length and diameter design variables). The model include the isentropic equations for the calculation of rocket thrust as a function of the expansion ratio, the chamber pressure, the mass flow rate, and the altitude variant ambient pressure. Thrust ceases when propellant is exhausted. The other rocket engine design parameters are based on those of the 1 kN LOX/IPA static test engine presently being developed at PSU as part of a senior capstone project. Drag is calculated using the quadratic drag model $F_d = \frac{1}{2}\rho V^2 C_d A$. The frontal area A is determined from the diameter design variable input. The local speed of sound, density and pressure are determined from the U.S. 1976 Standard Atmosphere. Drag coefficients are interpolated from aerodynamic data from the 1950's era NACA/USN/NASA Aerobee-150 sounding rocket class, which is expected to be dimensionally similar to the LV4 sounding rocket. Given a vector of the four design variables the trajectory function returns numerical values of the objective function and constraint vector. Pseudo-objective function.

D. Optimization Approach

Simplex search algorithm. (Choice of initial vertices, Additional contraction cases in 4D-space), issues with convergence, non-dimensionalization, multi-modality of response surface, parameter sensitivity (table of minima vectors). Benchmark with Scipy. Tables of minima vectors.

develop the pseudo-pbjective function.

1. Reasons for Nelder-Mead Method

There are several general approaches to MDO for LV design found in literature:

1. DOE methods (Taguchi,[4] RSM[5])
2. Gradient methods
3. Stochastic methods (genetic algorithm,[6] simulated annealing)

We selected a gradient free non-stochastic approach: the Nelder-Mead method. This method applies to nonlinear and derivative-free problems. With four design variables, using a simplex is beneficial to keep track of and continually improve on various optimums. Regarding this problem being derivative-free, a zeroth-order method is much easier to implement than one that requires taking the gradient of the pseudo-objective function mass summation. This is due to the fact that the mass summation is discrete, meaning any method requiring a gradient cannot be applied. If certain properties were made continuous functions, such as the atmospheric density with respect to altitude as well as drag coefficient with respect to mach number, a first-order method could potentially be applied. Steepest descent is a good first-order method choice with the benefit of arriving at a solution potentially much faster than the Nelder-Mead method.

2. Benchmarking

To ensure that the code written is providing accurate and sensible results, the output and code must be confirmed through various methods. One method of confirming the results is to apply a provided homework problem and respective solution, and check to see if the output correlates with the known answer. Since a previous homework problem for ME 596 required a simplex search, that problem was applied to the code written for the final project. The results correlated with the known answer, with no necessary changes to the code. Another method of confirming the results is to use an existing code that solves using the simplex search method. Applying the mass summation pseudo-objective function to the preexisting code provided in the ScyPy library resulted in a better optimum for the gross liftoff weight optimization with only a percent difference. With the results and method confirmed through various methods, the behavior of the algorithm can be discussed.

3. Qualitative Discussion of Algorithm behavior

The simplex search method developed has a few behaviors that do not show up in other packages using the same method. Over the course of a number of incrementations, the minimum values will change drastically over a single time step then change gradually over multiple timesteps. This will repeat as the analysis continues multiple times before a minimum is achieved. The simplex search method is also very capable of optimizing the design parameters, but has the potential to get stuck on what seem to be local minima for the objective function. This is potentially due to the fact that the code is choosing one method (reflection, expansion, contraction) and changing the method after the first iteration. This can, for example, lead to a large jump towards and optimum after the first iteration and small changes each iteration afterwards. An optimum is still achieved, with the code optimizing the design variables and constraints extremely well, hugging the limits of the constraints as close as possible. An example of this is how the optimum result may have an improved mass value, but the Sommerfeld criteria will be as close to 0.3 as possible without violating the constraint. The reasoning for this is unclear, but is potentially due to a large penalty function coefficient rp, where the function is drastically penalized any time the constraints are not met. Reducing the penalty function coefficient slightly reducing the tendency to hug the constraints, but harms the code in other ways such as extending analysis time and less optimal gross liftoff weights

III. Results

Discussion of converged results, implications for design of LV4

IV. Future Work

Model improvements: tanks mass/volume, other dry mass contributions, experiment with global optimization schemes, drag improvements, gravity model(WGS84 model harmonic expansion of the gravitational field potential), post-hoc analysis.

A. Drag Model Improvements

Since the model used for drag in the original trajectory function is based on recorded data for the Aerobee-150 sounding rocket, there is room for improvement. To improve the drag model, data that correlates better to the LV4 rocket design is needed. While the aerobee-150 ( ) drag data does correlate to a reasonable extent, data specific to the LV4 airframe is the best option. The desired values are drag coefficients and their respective mach numbers, which either requires flight data for the LV4 rocket or simulation results. To keep costs low while still acquiring good information, a CFD simulation of the LV4 rocket design will be performed. This simulation will be developed using both Solidworks and STAR CCM+, both of which are capable of supersonic CFD. The general method will be to develop the model based the optimization code in Solidworks, and compare those simulation results with recorded flight data. The method for implementing this information into the trajectory function will not change for the time being, meaning the results from the simulation will be tabulated and imported into the code the same way the aerobee-150 ( ) data is.

B. Feed System Improvements

The feed system for the simplex search model is simply a given mass value, and is not subject to optimization. This feed system is crucial to the function of the engine, and if not carefully designed can end up weighing a significant portion of the overall rocket weight. The feed system weight can lead to a potential issue with defining the center of pressure location and stability characteristics as well.

C. Structural Model Improvements

Besides the fact that the carbon-composite airframe will double as the fuel tank liner in certain sections, the overall strength of the structure is of great consideration. The greatest structural loading will occur at the point of highest dynamic pressure, typically occurring 25 percent of the way up to the apogee depending on the speed and size of the rocket. To determine the loading at this point, CFD or a simplified analytical model is required. The aerodynamic shear, pressure, and acceleration are all considerations that lead to determining the stresses on the airframe at this point. In the optimization code, the structural loading is limited by constraining acceleration to below 15 gs. This must be expanded on to include constraints on the maximum velocity of the rocket to ensure the maximum dynamic pressure and structural stresses are not too high.

D. Stability Model Addition

When considering the stability of a rocket, the center of pressure location with respect to the center of mass is very important. If the center of pressure is closer to the rocket nose than the center of mass, the rocket will have a tendency to want to flip, leading to a new design constraint that restricts the center of pressure from moving past the center of mass. The center of pressure also must not be too far behind the center of mass, otherwise the rocket will be impossible to maneuver in the thicker portions of the atmosphere.

As the fuel is drained from the rocket, the center of mass will move. Due to the fact that the feed system is expected to be nearly a quarter of the overall wet mass of the rocket, the center of mass will stay near the end of the rocket throughout the flight and move towards the rear as the fuel is drained. There is potential for the center of mass to move behind the center of pressure because of this, coupled with the fact that the center of pressure will remain fairly consistent throughout flight. The considerations for this were omitted from the optimization model, but merits addition to the model.

1-DoF simulation optimization of system parameters, but also potentially the trajectory itself (control vector, gain scheduling, etc.)

V. Conclusion

Focus on novelty of the problem. This will guide PSAS's technology development pathways, and inform requirements for future senior capstone projects sponsored by the organization.

References

[1] *Portland State Aerospace*. PSAS. Accessed February 25, 2016. http://psas.pdx.edu.

[2] Szedula, J.A., FASTPASS: A Tool For Launch Vehicle Synthesis, AIAA-96-4051-CP, 1996.

[3] Hempel, P. R., Moeller C. P., and Stuntz L. M., Missile Design Optimization Experience And Developments, AIAA-94-4344,1994-CP.

[4] Stanley, D. O., Unal, R., and Joyner, C. R., "Application of Taguchi Methods to Dual Mixture Ratio Propulsion System Optimization for SSTO Vehicles," Journal of Spacecraft and Rockets, Vol. 29, No. 4, 1992, pp. 453-459.

[5] Stanley, D. 0., Engelund. W. C., Lepsch. R. A., McMillin, M. L.Wt K. E.. Powell. R. W., Guinta. A. A., and Unal, R. "Rocket-Powered Single Stage Vehicle Configuration Selection and Design," Journal of Spacecraft and Rockets,Vol. 31, No. 5, 1994. pp. 792-798; also AIAA Paper93-Feb. 1993.

[6] Anderson, m., Burkhalter J., and Jenkins R Multidisciplinary Intelligence Systems Approach To Solid Rocket Motor Design, Part I: Single And Dual Goal Optimization. AIAA 2001-3599, July, 2001.

[7] *Oregon Small Satellite Project.* GitHub. Accessed February 25, 2016. https://github.com/oresat.

Appendix

Note that these codes can also be found on Github at ...

A. Trajectory Simulation Python Script

```
from math import sqrt, pi, exp, log, cos
import numpy as np
import csv

# A simple forward Euler integration for rocket trajectories
def dry_mass(L, dia):
    m_avionics = 3.3                          # Avionics mass          [kg]
    m_recovery = 4                            # Recovery system mass   [kg]
    m_payload = 2                             # Payload mass           [kg]
    m_tankage = 20.88683068354522*L*dia*pi # Tank mass Estimation   [kg]
    m_engine = 2                              # Engine mass            [kg]
    m_feedsys = 20                            # Feed system mass       [kg]
    m_airframe  = 6                           # Airframe mass          [kg]
    return (m_avionics + m_recovery + m_payload + m_tankage
    + m_engine + m_feedsys + m_airframe)   # Dry mass               [kg]

def propellant_mass(A, L, OF=1.3):
    rho_alc = 852.3             # Density, ethanol fuel [kg/m^3]
    rho_lox = 1141.0            # Density, lox          [kg/m^3]
    L_lox = L/(rho_lox/(rho_alc*OF) + 1)
    m_lox = rho_lox*L_lox*A     # Oxidizer mass         [kg]
    m_alc = rho_alc*(L-L_lox)*A # Fuel mass             [kg]
    return m_alc + m_lox        # Propellant Mass       [kg]

def std_at(h):                  # U.S. 1976 Standard Atmosphere
    if h < 11000:
        T = 15.04 - 0.00649*h
        p = 101.29*((T + 273.1)/288.08)**5.256

    elif 11000 <= h and h <25000:
        T = -56.46
        p = 22.65*exp(1.73 - 0.000157*h)

    else:
        T = -131.21 + 0.00299*h
        p = 2.488 * ((T + 273.1)/216.6)**(-11.388)

    rho = p/(0.2869*(T + 273.1)) # Ambient air density [kg/m^3]
    p_a = p*1000                 # Ambient air pressure [Pa]
    T_a = T + 273.1              # Ambient air temperature [K]
    return p_a, rho, T_a

def thrust(x, p_ch, T_ch, p_e, ke, Re, mdot):
    p_a = std_at(x)[0]                          # Ambient air pressure [Pa]
    p_t = p_ch*(1 + (ke - 1)/2)**(-ke/(ke - 1)) # Throat pressure      [Pa]
    T_t = T_ch*(1/(1 + (ke - 1)/2))             # Throat temperature   [K]
    A_t = (mdot / p_t)*sqrt(Re*T_t/ke)          # Throat area          [m^2]
    A_e = A_t*(2/(ke + 1))**(1/(ke - 1))*(p_ch/p_e)**(1/ke) * 1/sqrt((ke + 1)/(ke - 1)*(1 -
    (p_e/p_ch)**((ke - 1)/ke))) # Exit area [m^2]
    ex = A_e/A_t              # Expansion ratio
    alpha_t = [14, 11, 10, 9] # Lookup table of divergence angles, assuming 80% bell length
    ex_t = [5, 10, 15, 20]    # Lookup table of expansion ratios from alpha_t
    alpha= np.interp(ex, ex_t, alpha_t)
    lam = 0.5*(1 + cos(alpha *pi/180)) # Thrust cosine loss correction, even in extreme
    cases this is definitely not an O(1) effect
    Ve = lam*sqrt(2*ke/(ke - 1)*Re*T_ch*(1 - (p_e/p_ch)**((ke - 1)/ke))) # Exhaust velocity
                                [m/s]
    F = mdot*Ve + (p_e - p_a)*A_e                                        # Thrust force,
    ignoring that isp increases w/ p_ch [N]
    return F, A_t, A_e, Ve

def drag(x, v, A, Ma, C_d_t, Ma_t):
    # Check Knudsen number and switch drag models (e.g. rarified gas dyn vs. quadratic drag)
```

```
    (p_a, rho, T_a) = std_at(x)

    #C_d_t = [0.15, 0.15, 0.3, 0.45, 0.25, 0.2, 0.175, .15, .15] # V2 rocket drag coefficient lookup table
    #Ma_t = [0, 0.6, 1.0, 1.1, 2, 3, 4, 5, 5.6]                  # V2 rocket Mach number lookup table
    C_d = np.interp(Ma, Ma_t, C_d_t)                             # Drag coefficient function
    q = 0.5 * rho * v**2                                         # Dyanmic pressure [Pa]
    D = q * C_d * A                                              # Drag force       [N]
    return D, q

def trajectory(L, mdot, dia, p_e, p_ch=350, T_ch=3500, ke=1.3, Re=349, x_init=0):
    # Note combustion gas properties ke, Re, T_ch, etc, determined from CEA
    # Physical constants
    g_0 = 9.81 # Gravitational acceleration [m/s^2]
    dt = 1     # Time step                 [s]
    ka = 1.4   # Ratio of specific heats, air
    Ra = 287.1 # Avg. specific gas constant (dry air)

    # LV4 design variables
    dia = dia*0.0254        # Convert in. to m
    A = pi*(dia/2)**2       # Airframe frontal area projected onto a circle of diameter variable dia
    m_dry = dry_mass(L, A) # Dry mass, call from function dry_mass()
    mdot = mdot             # Mass flow rate [kg/s]
    p_ch = p_ch*6894.76     # Chamber pressure, convert psi to Pa
    p_e = p_e*1000          # Exit pressure, convert kPa to Pa

    # Initial conditions
    x = [x_init]
    v = [0]
    a = [0]
    t = [0]
    rho = [std_at(x[-1])[1]]
    p_a = [std_at(x[-1])[0]]
    T_a = [std_at(x[-1])[2]]
    m_prop = [propellant_mass(A, L)]
    m = [m_dry + m_prop[-1]]
    (F, A_t, A_e, Ve) = thrust(x[-1], p_ch, T_ch, p_e, ke, Re, mdot)
    F = [F]
    D = [0]
    Ma = [0]
    q = [0]
    r = (m_prop[0] + m_dry)/m_dry # Mass ratio
    dV1 = Ve*log(r)/1000          # Tsiolkovsky's bane (delta-V)

    # Drag coefficient look up
    C_d_t = []
    Ma_t = []
    f = open('CD_sustainer_poweron.csv') # Use aerobee 150 drag data
    aerobee_cd_data = csv.reader(f, delimiter=',')
    for row in aerobee_cd_data:
        C_d_t.append(row[1])
        Ma_t.append(row[0])

    while True:
        p_a.append(std_at(x[-1])[0])
        rho.append(std_at(x[-1])[1])
        T_a.append(std_at(x[-1])[2])
        # Check of the propellant tanks are empty
        if m_prop[-1] > 0:
            (Fr, A_t, A_e, Ve) = thrust(x[-1], p_ch, T_ch, p_e, ke, Re, mdot)
            F.append(Fr)
            m_prop.append(m_prop[-1] - mdot*dt)
            mdot_old = mdot
        else:
            Ve = thrust(x[-1], p_ch, T_ch, p_e, ke, Re, mdot_old)[3]
            F.append(0)
            mdot = 0
            m_prop[-1] = 0
```

```
        q.append(drag(x[-1], v[-1], A, Ma[-1], C_d_t, Ma_t)[1])
        D.append(drag(x[-1], v[-1], A, Ma[-1], C_d_t, Ma_t)[0])
        a.append((F[-1] - D[-1])/m[-1] - g_0)
        v.append(a[-1]*dt + v[-1])
        x.append(v[-1]*dt + x[-1])
        Ma.append(v[-1]/sqrt(ka*Ra*T_a[-1]))
        t.append(t[-1] + dt)
        m.append(m_dry + m_prop[-1])
        TWR = a[1]/g_0       # Thrust-to-weight ratio constraint
        ex = A_e/A_t
        S_crit = p_e/p_a[0] # Sommerfield criterion constraint
        if v[-1] <= 0:
            x = np.array(x)
            a = np.array(a)
            F = np.array(F)
            D = np.array(D)
            q = np.array(q)
            return x, v, a, t, F, D, Ma, rho, p_a, T_a, TWR, ex, Ve, A_t, dV1, m, S_crit, q,
    m_prop
```

B. Simplex Search Python Script

```
# Class simplex:
# Nelder-Mead simplex search
import numpy as np
from math import sqrt, pi, exp, log, cos
import math as m

def search(f, x_start, max_iter = 100, gamma = 5, beta = 0.5, rp=100, a=10, epsilon = 1E-6):
    """
    parameters of the function:
    f is the function to be optimized
    x_start (numpy array) is the initial simplex vertices
    epsilon is the termination criteria
    gamma is the contraction coefficient
    beta is the expansion coefficient
    """
    # Init Arrays
    N = len(x_start)     # Amount of design variables
    fb = []              # Empty function matrix
    xnew = []            # Empty re-write for design variables
    x    = []            # Empty x matrix
    C    = [[0]*N]*(N+1) # Empty center point matrix #####CHANGED

    # Generate vertices of initial simplex
    x0 = (x_start)   # x0 Value for x Matrix
    x1 = [x0 + [((N + 1)**0.5 + N - 1.)/(N + 1.)*a, 0., 0., 0.]]
    x2 = [x0 + [0., ((N + 1)**0.5 - 1.)/(N + 1.)*a, 0., 0.]]
    x3 = [x0 + [0., 0., ((N + 1)**0.5 - 1.)/(N + 1.)*a, 0.]]
    x4 = [x0 + [0., 0., 0., ((N + 1)**0.5 - 1.)/(N + 1.)*a]]
    x = np.vstack((x0, x1, x2, x3, x4))

    # Simplex iteration
    while True:
        # Find best, worst, 2nd worst, and new center point
        f_run = np.array([f(x[0], rp), f(x[1], rp), f(x[2], rp), f(x[3], rp), f(x[4], rp)]).
    tolist() # Func. values at vertices
        xw = x[f_run.index(sorted(f_run)[-1])] # Worst point
        xb = x[f_run.index(sorted(f_run)[0])]  # Best point
        xs = x[f_run.index(sorted(f_run)[-2])] # 2nd worst point
        for i in range(0, N+1):
            if i == f_run.index(sorted(f_run)[-1]):
                C[i] = [0,0,0,0]
            else:
                C[i] = x[i].tolist()
        xc = sum(np.array(C))/(N) # Center point
        xr = 2*xc - xw            # Reflection point
        fxr = f(xr, rp)
        fxc = f(xc, rp)

```

```
        # Check cases
        # f(xr, rp) < f(xb, rp): # Expansion
        if fxr < f_run[f_run.index(sorted(f_run)[0])]:
            xnew = (1 + gamma)*xc - gamma*xr
        # f(xr, rp) > f(xw, rp): # Contraction 1
        elif fxr > f_run[f_run.index(sorted(f_run)[-1])]:
            xnew = (1 - beta)*xc + beta*xw
        # f(xs, rp) < f(xr, rp) and f(xr, rp) < f(xw, rp): # Contraction 2
        elif f_run[f_run.index(sorted(f_run)[-2])] < fxr and fxr < f_run[f_run.index(sorted(f_run)[-1])]:
            xnew = (1 + beta)*xc - beta*xw
        else:
            xnew = xr

        # Replace Vertices
        x[f_run.index(sorted(f_run)[-1])] = xnew
        #x[f_run.index(sorted(f_run)[1])] = xb # Replace best
        #x[f_run.index(sorted(f_run)[2])] = xs # Replace second best
        fb.append(f(xb, rp))
        print('Current optimum = ', fb[-1])

        # Break if any termination critera is satisfied
        if len(fb) == max_iter: #or term_check(x, xc, xw, N, rp, f_run) <= epsilon:
            (alt, v, a, t, F, D, Ma, rho, p_a, T_a, TWR, ex, Ve, A_t, dV1, m, S_crit, q, m_prop, p_ch) = trajectory(xb[0], xb[1], xb[2], xb[3])
            return f(x[f_run.index(sorted(f_run)[0])], rp), x[f_run.index(sorted(f_run)[0])], len(fb)

def term_check(N, rp, f_run, fxc): # Termination critera
    M = [0]*(N + 1)
    for i in range(0, N + 1):
        if i == f_run.index(sorted(f_run)[-1]): # Avoid worst point
            M[i] = 0
        else:
            M[i] = (f_run[i] - fxc)**2
    return m.sqrt(sum(M)/N)

# Pseudo-objective function
def f(x, p_ch=350, rp=50): ##CHANGE CHAMBER PRESSURE HERE
    L = x[0]    # Rocket length (m)
    mdot = x[1] # Propellant mass flow rate (kg/s)
    dia = x[2]  # Rocket diameter (in)
    p_e = x[3]  # Pressure (kPa)
    (alt, v, a, t, F, D, Ma, rho, p_a, T_a, TWR, ex, Ve, A_t, dV1, m, S_crit, q, m_prop) = trajectory(L, mdot, dia, p_e, p_ch)
    #CHANGE CONSTRAINTS HERE
    obj_func = m[0] + rp*(max(0, (L+2)/(dia*0.0254) - 15)**2 + max(0, -TWR + 2)**2 + max(0, -S_crit + 0.35)**2 + max(0, -alt[-1] + 100000)**2 + max(0, max(abs(a))/9.81 - 15)**2)
    return obj_func

if __name__ == '__main__': # Testing
    ##CHANGE INITIAL DESIGN GUESS HERE
    X0 = np.array([1, 0.453592 * 0.9 * 4, 12, 50])
    #X0 = np.array([2, 0.453592 * 0.9 * 6, 8, 50])
    """max_iter = 200
    rp = 50
    gamma = 6
    beta = .5
    a = 5
    (f, x, it) = search(f, np.array(X0), max_iter, gamma, beta, rp, a)
    """
    from scipy.optimize import minimize
    res = minimize(f, X0, method='nelder-mead')

    p_ch = 350 # Chamber pressure [kPa] **DONT FORGET TO CHANGE THE VALUE IN THE OBJECTIVE FUNCTION IN def f()**
    (alt, v, a, t, F, D, Ma, rho, p_a, T_a, TWR, ex, Ve, A_t, dV1, m, S_crit, q, m_prop) = trajectory(res.x[0], res.x[1], res.x[2], res.x[3], p_ch)
    print('\n')

```

```
    # Plot the results
    import matplotlib
    import matplotlib.pyplot as plt
    import pylab
    %config InlineBackend.figure_formats=['svg']
    %matplotlib inline
    # Redefine the optimized output
    L = res.x[0]
    mdot = res.x[1]
    dia = res.x[2]
    p_e = res.x[3]

    pylab.rcParams['figure.figsize'] = (10.0, 10.0)
    f, (ax1, ax2, ax3, ax4, ax6, ax7) = plt.subplots(6, sharex=True)
    ax1.plot(t, alt/1000)
    ax1.set_ylabel("Altitude (km)")
    ax1.yaxis.major.locator.set_params(nbins=6)
    ax1.set_title('LV4 Trajectory')
    ax2.plot(t, v)
    ax2.yaxis.major.locator.set_params(nbins=6)
    ax2.set_ylabel("Velocity (m/s)")
    ax3.plot(t, a/9.81)
    ax3.yaxis.major.locator.set_params(nbins=10)
    ax3.set_ylabel("Acceleration/g0")
    ax4.plot(t, F/1000)
    ax4.yaxis.major.locator.set_params(nbins=6)
    ax4.set_ylabel("Thrust (kN)")
    ax6.plot(t, q/1000)
    ax6.yaxis.major.locator.set_params(nbins=6)
    ax6.set_ylabel("Dynamic Pressure (kPa)")
    ax7.plot(t, Ma)
    ax7.yaxis.major.locator.set_params(nbins=6)
    ax7.set_ylabel("Mach number")
    ax7.set_xlabel("t (s)")
    plt.show()

    np.set_printoptions(precision=3)
    print('\n')
    print('OPTIMIZED DESIGN VECTOR')
    print('-------------------------------')
    print('x_optimized                                = ', res.x)
    print('x_initial_guess                            = ', X0)
    print('design tankage length                      = {0:.2f} m'.format(res.x[0]))
    print('design mass flow rate                      = {0:.2f} kg/s'.format(res.x[1]))
    print('design airframe diameter                   = {0:.2f} in.'.format(res.x[2]))
    print('design nozzle exit pressure                = {0:.2f} kPa'.format(res.x[3]))
    print('iterations                                 =', res.nit)
    print('design GLOW                                = {0:.1f} kg'.format(m[0]))
    print('x0 GLOW                                    = {0:.1f} kg'.format(trajectory(X0[0],
     X0[1], X0[2], X0[3], p_ch)[-4][0]))

    print('\n')
    print('CONSTRAINTS')
    print('-------------------------------')
    print('L/D ratio (check < 15)                     = {0:.2f}'.format((L+2)/(dia*0.0254)))
    print('Sommerfield criterion (check pe/pa >= 0.3) = {0:.1f}'.format(S_crit))
    print("Max acceleration (check < 15)              = {0:.2f} g's".format(max(abs(a))
    /9.81))
    print('TWR at lift off (check TWR > 2)            = {0:.2f}'.format(TWR))
    print('altitude at apogee                         = {0:.1f} km'.format(alt[-1]/1000))

    print('\n')
    print('ADDITIONAL INFORMATION')
    print('-------------------------------')
    print('mission time at apogee                     = {0:.1f} s'.format(t[-1]))
    print('design total propellant mass               = {0:.3f} kg'.format(m_prop[0]))
    print('design thrust (sea level)                  = {0:.1f} kN'.format(F[0]/1000))
    j = 0
    for thing in F:
        if thing == 0:
```

```
179             fdex = j
180             break
181         j += 1
182     print('design thrust (vacuum)                    = {0:.1f} kN'.format(F[fdex - 1]/1000)
        )
183     print('design burn time                          = {} s'.format(fdex))
184     print('design expansion ratio                    = {0:.1f}'.format(ex))
185     print('design throat area                        = {0:.1f} in.^2'.format(A_t/0.0254**2)
        )
186     print('design isp                                = {0:.1f} s'.format(Ve/9.81))
187     print('design chamber pressure                   = {0:.1f} psi'.format(p_ch))
188     print('design dV                                 = {0:.1f} km/s'.format(dV1))
189     print('estimated minimum required dV             = {0:.1f} km/s'.format( sqrt(2*9.81*
        alt[-1])/1000))
```