Multi-Disciplinary Design Optimization of Small Size Expendable Launch Vehicle

An internship report submitted

in partial fulfillment for the award of the degree of

Bachelor of Technology

in

Aerospace Engineering

by

Dasu Deva Karthik Lakshman, Shashwat Gupta

Department of Aerospace Engineering

Indian Institute of Space Science and Technology

Thiruvananthapuram, India

October 23, 2021

Certificate

This is to certify that the internship report titled ***Multi-Disciplinary Design Optimization of Small Size Expendable Launch Vehicle*** submitted by **Dasu Deva Karthik Lakshman, Shashwat Gupta**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Aerospace Engineering** is a bona fide record of the original work carried out by him/her under my supervision. The contents of this internship report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Aravind Vaidyanathan
Professor and Head of the Department
Department of Aerospace Engineering

Dr. Aravind Vaidyanathan
Professor and Head of the Department
Department of Aerospace Engineering

Place: Thiruvananthapuram
Date: October 23, 2021

Declaration

We declare that this internship report titled ***Multi-Disciplinary Design Optimization of Small Size Expendable Launch Vehicle*** submitted in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Aerospace Engineering** is a record of the original work carried out by me under the supervision of **Dr. Aravind Vaidyanathan**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited.

Place: Thiruvananthapuram
Date: October 23, 2021

Dasu Deva Karthik Lakshman, Shashwat Gupta
(SC18B010, SC18B027)

Acknowledgements

We acknowledge the guidance and support of our advisor Dr. Aravind Vaidyanathan throughout the duration of this project. We acknowledge Shri. Sharvi Ghanekar, alumni of IIST for her contributions with the initial design. We would like to express our most sincere gratitude towards Dr. R V Ramanan, Adjunct Professor (retd.) for his immense help through the discussions we have had with him. Our interactions with them have been a source of inspiration for us. We are grateful to have been given numerous learning opportunities at IIST. We would also like to thank our parents and family for their love and encouragement.

Dasu Deva Karthik Lakshman, Shashwat Gupta

Abstract

The design of a launch vehicle starts from the definition of mission objective(s) and their versatility. Launch Vehicle is not a single entity but a system consisting of numerous subsystems such as aerodynamics, avionics, launch vehicle configuration, propulsion, trajectory, etc., which makes Launch Vehicle Design a complex design process. The subsystems often have conflicting objectives which require the search for multi-disciplinary compromises among the subsystems. This can be done by using special design techniques called Multi-Disciplinary Design Optimization (MDO) Methods. This work illustrates the design of Small Size Expendable Launch Vehicle (SSELV) using Multi-Disciplinary Design Optimization Method with the objective of delivering 500 kg payload to a 500 km circular polar orbit. Launch vehicle configuration, propulsion, aerodynamics and ascent trajectory are identified as major subsystems and are optimized using MDO method. Launch vehicle sizing is optimized as a first step, and that gives us the optimal number of stages, and stage masses and configurations to accomplish the objective. The optimized configuration of the vehicle comprises of three stages and the dimensions of the Launch Vehicle are taken as 2m diameter with 18m of height. Solid Rocket Motors (SRM) are chosen for propelling the first stage of the launch vehicle due to their compact design and large thrust. Star-grain configuration is selected for SRM and the grain geometry is optimized to get the desired thrust. Since the configuration of the launch vehicle is fixed, aerodynamics analysis is done separately on ANSYS Fluent and the results are integrated with the optimization process. Two-dimensional ascent trajectory is optimized for the current work and the launch site is chosen to be **Kulasekharapattinam**. The MDO method used in the current work is Multi-Disciplinary Feasible (MDF) method and the results obtained using this method are compared against traditional Sequential or Fixed Point Iteration (FPI) method. A hybrid optimization solver consisting of standard Particle Swarm Optimization (PSO) and Sequential Quadratic Programming (SQP) is employed to implement both the design methods.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

A launch vehicle consists of various subsystems (or disciplines) like aerodynamics, avionics, propulsion, trajectory, configuration, etc. The design process of a launch vehicle is complex and multifaceted. The analyses are not only limited to how the complex disciplines interact with each other but also have to incorporate how the disciplines interact with and influence each other. The interactions of these disciplines of the launch vehicle play a major role in meeting mission requirements and improving the efficiency and performance of the vehicle. Numerous approaches are used for designing the launch vehicle, like systems engineering approach, traditional sequential or fixed point iteration (FPI) approach, Multi-Disciplinary Design Optimization (MDO) approach, etc. Traditional or FPI method involves discipline experts of various disciplines who optimize the each subsystem iteratively until obtaining a design point where all disciplines agree on the common variables and satisfactory values of the global objectives are reached, thus requiring huge amount of time. Moreover, the resulting solution is not necessarily the global optimum because of the sequential approach. MDO approach doesn't present this problem as MDO methods are a type of optimization methods which handle various disciplines simultaneously and make use of interactions among the subsystems and concurrently vary all the design variables at the same time to obtain global optimal solution in the multi-disciplinary search space.

The world is moving towards miniaturization of spacecrafts and making use of spacecraft constellations at Low-Earth Orbits (LEO), instead of single heavy spacecraft in a higher orbit for performing a specific task. As a consequence, the frequency of launches will increase and use of heavy launch vehicles for injecting the spacecrafts into LEO may not be economical in terms of cost and effort. Small-Size Expendable Launch Vehicles (SSELV) fill this gap, SSELV's are effective in terms of cost, effort, technology and require less time for launches. The current work focuses on the design optimization of Small-Size Expendable Launch Vehicle using MDO method with the goal of injecting a 500 kg payload

into a 500 km circular orbit.

Launch Vehicle Configuration/Sizing, Aerodynamics, Propulsion and Ascent Trajectory are considered as major disciplines in the SSELV and the whole system is optimized using both traditional and MDO methods and the results are compared.

1.1 Launch Vehicle Configuration/Sizing

Sizing/Configuration of the launch vehicle refers to determining the masses of each of the stages of the vehicle and the configuration of the stages, i.e. the mass of structures and propellant in each of the stages. The total initial mass of the vehicle, also called gross liftoff mass, is the sum of the masses of all stages and that of the payload. Launch Vehicle Staging Optimization is done using semi-Analytical approach derived from Lagrange Multiplier method. The gross liftoff mass is observed to decrease with increase in the number of stages, but increasing the number of stages also means an increase in complexity and inter-stage mass, so an optimum number of 3 stages has been chosen for the current design.

1.2 Propulsion

Since the launch vehicle in the current work is of expendable type, Solid Rocket Motors (SRM) are the ideal choice due to their simplicity in manufacturing, operating and storage. Out of the three stages, the first is a solid rocket motor with star grain configuration and the geometry of star grain has been optimised as for mission requirements subject to the specified constraints. Two-dimensional star grain with analytical burnback analysis has been considered for analysis and Zero-Dimensional Internal Ballistic Analysis is chosen as physical model for the combustion. The second and third stages are assumed to deliver constant thrust.

An optimal SRM as a first stage should have an "M" shaped thrust profile because it needs to accelerate initially to clear the launch pad and tower and then it should decrease as the launch vehicle approaches trans-sonic region, the drag on the vehicle increases dramatically and also to satisfy the dynamic pressure constraint. After the completion of the atmospheric phase, the density reduces drastically and the aerodynamic drag on the vehicle would be much lesser than what it would be in the atmospheric phase. So the thrust can be increased to achieve larger $Deltav$ without large losses. This "M" shaped thrust profile has been optimised for the first stage in our design.

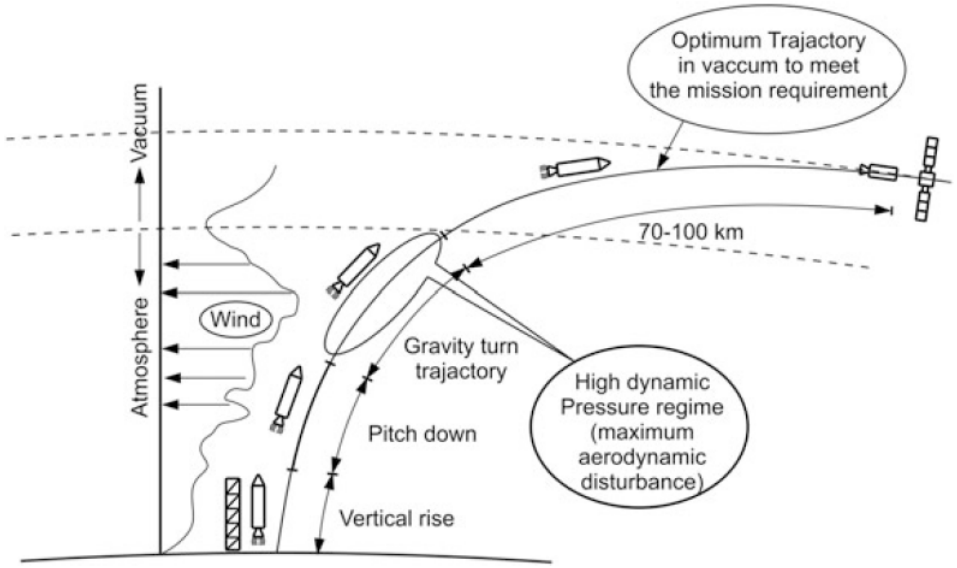

Figure 1.1: Trajectory Phases of a Launch Vehicle[1]

1.3 Aerodynamic Analysis

The drag coefficient of the launch vehicle was plotted against mach number using ANSYS Fluent for the flow simulation. The drag coefficient is observed to be low in subsonic, supersonic and hypersonic regions and highest in the trans-sonic region. It peaks sharply as the mach number approaches a value a little over unity from the subsonic regime, the drag coefficient shoots up sharply and it slowly decreases as the mach number increases beyond the peak drag value. This variation has been plotted and shown in Fig. 8.3.

1.4 Ascent Trajectory

The ascent trajectory lies at the heart of launch vehicle design as it plays a major role in the optimization of the launch vehicle. The ascent of the launch vehicle can be divided into two phases: endo-atmospheric phase and exo-atmospheric phase. The endo-atmospheric phase consists of atmosphere and the ascent is performed in open-loop mode due to the presence of huge aerodynamic loads, which complicates the optimal control. The exo-atmospheric phase is mostly vacuum flight and ascent is performed in closed-loop mode as there are no aerodynamic loads. The ascent trajectory is formulated mathematically by numerically solving the governing equations of motion and in body centered frame and converting them to earth centered inertial frame which is used as the inertial frame. The

ascent trajectory consists of several phases starting from vertical lift-off phase, followed by constant pitching (and yawing if necessary) after which it undergoes gravity turn. The end of the gravity turn phase implies the end of the atmospheric phase. Thereafter, the optimal trajectory is followed. The trajectory is optimized with the goal of either maximizing the altitude or maximizing the final payload at injection. The various trajectory phases (not to scale) are shown in Fig. 1.1.

1.5 Work Flow

The current work deals primarily with the design optimization of Small-Size Expendable Launch Vehicle comprising of four major disciplines: Launch Vehicle Configuration/Sizing, Propulsion, Aerodynamics and Ascent Trajectory.

Chapter 2 provides the brief survey of literature in the fields of launch vehicle design, Solid Rocket Motor (SRM) design, Ascent Trajectory and Optimization techniques,

Chapter 3 provides the problem formulation and optimization technique for the Launch Vehicle Configuration/Sizing, and specifies the Gross Lift Off Mass and stage masses of the vehicle.

Chapter 4 details the design methodology, problem formulation and optimization techniques for Solid Rocket Motors (SRM).

Chapter 5 provides the aerodynamic analysis of the Launch Vehicle using ANSYS Fluent and specifies the variation of drag coefficient with the mach number.

Chapter 6 details the problem formulation, methodology and optimization of the ascent trajectory. This includes description of various co-ordinate frames and transformations, and modelling of various phases and other certain topics relevant to trajectory design.

Chapter 7 describes the optimization methods, mathematical formulation, terminology and other significant aspects used in designing the launch vehicle.

The results, analyses and discussions pertaining to the optimization methods and methodology are presented in Chapter 8. Conclusions and future work related to the study in Chapter 9 conclude the report.

Chapter 2

Review of Related Literature

This chapter reviews the major works related to Launch vehicle Design and different approaches to do it. The concepts of the current work draw from and are loosely based on ideas from these works. The design of a launch vehicle can be done in many ways; the Sequential or Fixed Point Iteration method is the traditional method for launch vehicle design in which each discipline is optimized separately and sequentially, and the Multi-Disciplinary optimization (MDO) method is the emerging method in which all the disciplines interact with each other and the unified objective function converges to global optimal solution. MDO methods in Launch Vehicle Design gained popularity with improvements in computation power in the 21^{st} Century. The disciplines involved are Launch Vehicle Sizing, Propulsion, Aerodynamics and Ascent Trajectory. From the basic vehicle definition and geometry, the aerodynamic analysis is done independent of other disciplines, and the sizing, propulsion and trajectory disciplines are optimized.

Most of the researchers optimize three to four disciplines for the Launch Vehicle Design, and a few used cost as an additional discipline. *Castellini et. al*[11] pointed out the main advantages of MDO methods over traditional methods in Launch Vehicle Design. Castellini used two approaches for propulsion discipline, one by using Off-The-Shelf components data (Thrust-time curves, etc) and another by completely starting from new design. *Lorenzo et. al*[12], along with the trajectory, optimized the normalized parametric thrust-time profile of a solid rocket motor for the first stage instead of grain and internal ballistic design. Most of the researchers used available softwares as black boxes for the propulsion and Ascent trajectory analyses in launch vehicle Design.

Realising the potential of the MDO methods, many researchers from the past three decades applied them in the launch vehicle design using many formulations (or) architectures. Nearly 80% of the researchers used Multi-Disciplinary Feasible (MDF) method, a classic single-level MDO architecture in Launch Vehicle Design. A few of the researchers

also used Bi-Level MDO architectures like Collaborative Optimization (CO), etc.

Balesdent et. al[10] proposed a new MDO architecture for Launch vehicle design known as SWORD (Stage-Wise decomposition for optimal Rocket Design). This method splits up the design process into the different flight phases and transforms the multi-stage launch vehicle optimization problem into the coordination of the optimizations of each of the stages. For the case of the global optimization of a three-stage-to-orbit launch vehicle, SWORD outperforms the classical MDF method in terms of efficiency and speed.

Geethakrishnan et. al[13] proposed a new Bi-level hybrid MDO architecture known as Sequenced Collaborative Optimization with Nested Sequential Individual Discipline Feasible (SCO-NSIDF), and is developed for the implementation of the MDO problem of Two-Stage-to-Launch-Vehicle design. Hybrid optimization algorithm, Genetic Algorithm Guided Gradient Search (GAGGS), is chosen as optimizer at subsystems level and Genetic Algorithm (GA) is chosen as system level optimizer.

Solid Rocket Motors (or Solid Propellant Rocket Engines) are widely used in the Multi-Disciplinary Optimization of Launch Vehicle due to their simplicity and analytical tractability. NASA's Solid Propellant Grain Design and Internal ballistics[14] monograph details the Solid Rocket Motor design from the starting point. The solid rocket motor design can be divided into 3 major portions, the initial grain geometry, internal ballistics and grain burnback analysis. Grain burnback analysis is nothing but finding the regression of burning surface and is most important thing in Solid Rocket Motor design and there are many techniques to do it. *Hartfield et. al*[5] pioneered the analytical techniques for burnback analysis for different grain configurations, where he represented the star grain using five independent design variables and three burning phases. *Riccardi et. al*[15] represented the star grain with seven independent parameters and sixteen different burning phases. Kamran[16] used CAD software integrated with the solver for finding regression of the burning surface of the grain.

Ascent trajectory optimization lies at the heart of the launch vehicle design and it can be divided into two major portions, one is formulation or simulation of the trajectory, and other is optimization. Most of the available Trajectory Optimization softwares uses direct method for optimization in which the trajectory optimization problem is formulated as a Non-Linear Programming Problem which includes many constraints and variables. NASA's Program to Optimize Simulated Trajectories (POST) monograph[17] gives a comprehensive review of the various co-ordinate frames and transformations central to the simulation of the ascent trajectory. Gravitational and atmospheric models, as well as the oblate

spheroid model of the Earth, and their application to the problem are discussed. POST is a generalised point mass (or 3 DOF) FORTRAN program for simulating and optimizing trajectories aerospace vehicles. POST uses Recursive Quadratic Programming (RQP) for optimizing the trajectories.

Adimurthy et. al[18] proposed a new optimization methodology for three dimensional trajectory optimization problem which uses a Diagonalized Multiplier Method (DMM) in which the multiplier update formula of Tapia and Han in conjunction with standard BFGS inverse-Hessian update formula. A software package PYOPT (Pitch and Yaw rates Optimization) was developed by ISRO based on the above DMM method, and it has been proved to be faster than POST those based on recursive quadratic programming (RQP).

Many researchers tried optimizing the trajectory using indirect methods like Euler-Lagrange Equations, Pontryagin minimization principle to construct the necessary and sufficient conditions which are then solved numerically. *Pontani et. al*[19] used indirect methods for problem formulation and solved using a heuristic method Particle Swarm Optimization, which he termed as indirect heuristic approach. The author pointed out the advantages of using Particle Swarm Optimization (PSO) for optimizing trajectories.

Range safety or impact point constraints are very important as the spent stages must not fall in the populated areas. Land mass restrictions can be given as additional constraints to the optimization problem. Yoon and Ahn[20] proposed a novel technique for handling range safety constraints by calculating the impact points analytically. The authors elaborated different kinds of impact point constraints.

Meta-Heuristic Optimization Methods like Particle Swarm Optimization, Differential Evolution, etc. are proven to be very efficient for solving real valued global unconstrained optimization problems but they are not capable of solving constrained problems. The constrained optimization problem is usually converted to unconstrained problem by using penalty function methods. *Paraspoulos and Vrahatis*[21] proposed a new type of penalty function known as Non-Stationary Exterior Penalty Function Method in which the penalty values are dynamically modified based on how much the constraint is violated.

2.1 Scope of the Current Work

The current work aims to conceptually design the Small Size Expendable Launch Vehicle for a mission to a Low-Earth Orbit. In the current work, Multi-Disciplinary Design Optimization method is used to obtain the global optimal solution for the entire vehicle and its subsystems, and is compared with the traditional Sequential/Fixed Point Iteration Design

method.

Launch Vehicle Sizing, Propulsion, Aerodynamics and Ascent Trajectory are identified for the Launch Vehicle Design. The Sizing optimization problem to obtain the Gross Liftoff Mass has been solved using the serial staging formulation[**?**].

The first stage of the Small Size Launch Vehicle is the most crucial and occupies major portion of the launch vehicle. In the current work, the propulsion module for the first stage is designed and optimized, thrust for remaining stages are considered as constant; solid propellant is considered as the fuel and grain design is optimized by formulating it as a Non-Linear Programming Problem (NLP) subjected to maximum pressure and time constraints with grain design parameters as design variables.

The configuration of the launch vehicle is fixed and the aerodynamic analysis is done separately using ANSYS Fluent and then the results are integrated with the design.

Ascent trajectory is optimized by formulating it as a Non-Linear Programming Problem (NLP) subjected to path, terminal constraints with pitch, yaw rates and coasting time after second stage as design variables.

The above disciplines are optimized in a unified manner by using the Multi-Disciplinary Feasible Method and the entire problem is formulated as Non-Linear Programming Problem (NLP).

A Hybrid Optimization Solver is used to solve the NLP problems. It comprises of standard Particle Swarm Optimization (PSO) and Sequential Quadratic Programming (SQP).

The implementation is done in MATLAB.

Chapter 3

Sizing Module

The determination of the masses and configurations (distribution between propellant and structural mass) of the stages of the launch vehicle, and hence the gross liftoff mass, comprises the staging or sizing optimisation. The following parameters serve as the inputs for the staging module:

1. Final payload mass
2. Total Δv requirement
3. Number of stages of the launch vehicle
4. Thrust-to-Weight ratio for each stage
5. Structural ratio (ϵ) of each stage
6. Specific impulse (I_{sp}) of each stage given in terms of c, $c = g_0 \times I_{sp}$

The outputs that need to be solved for are:

1. The structural mass of each stage (m_s)
2. The propellant mass of each stage (m_p)
3. The initial mass of each stage (m_i)

These outputs also allow us to compute the mass ratio and the maximum deliverable thrust of each stage.

3.1 $\triangle v$ Requirement

The Δv requirement is the cornerstone of the staging module, and it depends on the final orbit that the payload has to be inserted in. It ideally equals the orbital velocity ($\sqrt{\frac{\mu}{r}}$) of the payload, but in actual flight, there will be various losses due to gravity, atmospheric drag, steering and gain or loss due to Earth's rotation depending on direction of launch. But the problem with this line of thinking is that as the orbital radius increases, the orbital velocity decreases and so does the Δv requirement. This is not physically true and to overcome this inconsistency, the Energy method is used.
A specified orbit has a total specific energy associated to it and is described by Eqn. 3.1.

$$\begin{aligned} TE &= PE + KE \\ &= \tfrac{-\mu}{a} + \tfrac{\mu}{2a} \\ &= \tfrac{-\mu}{2a} \end{aligned} \tag{3.1}$$

, where TE, PE, KE are the specific total, potential, and kinetic energies associated with the orbit having semi-major axis a.
Moreover, the specific total energy at the surface of the Earth is given by Eqn. 3.2, according to Newton's Law of Gravitation under the reasonable assumption that the distance between the center of the Earth and that of the spacecraft on the Launch Pad is equal to the Radius of the Earth.

$$TE_s = \frac{-GM_E}{R_E} \tag{3.2}$$

, where G is the Universal Gravitational Constant, M_E is the mass of the Earth and R_E is the radius of the Earth.
The ideal Δv requirement is then obtained using Eqn. 3.3.

$$TE - TE_s = \frac{v_{ideal}^2}{2} \tag{3.3}$$

The losses and gains are finally adjusted into this ideal Δv requirement as per Eqn. 3.4.

$$\Delta v_{mission} = \Delta v_{ideal} + \Delta v_{loss} - \Delta v_{gain} \tag{3.4}$$

3.2 Optimisation Problem

According to Tsiolkovsky's Rocket Equation, we have Eqn. 3.5 for a single stage.

$$\Delta v_{mission} = c \times \ln(\Lambda) \tag{3.5}$$

For a multi-stage launch vehicle, we can extend the equation to get Eqn. 3.6.

$$\Delta v_{mission} = \sum_{k=1}^{N} c_k \ln(\Lambda_k) \tag{3.6}$$

, where c_k is the exhaust velocity of the k^{th} stage and is equal to $g_0 \times I_{sp_k}$, Λ_k is the Mass Ratio of the k^{th} stage, and N is the number of stages in the Launch Vehicle. Figure 3.1 shows the mass definitions for the simple case of serial staging.

In serial staging, a single stage burns until its propellant is exhausted and the burnt stage

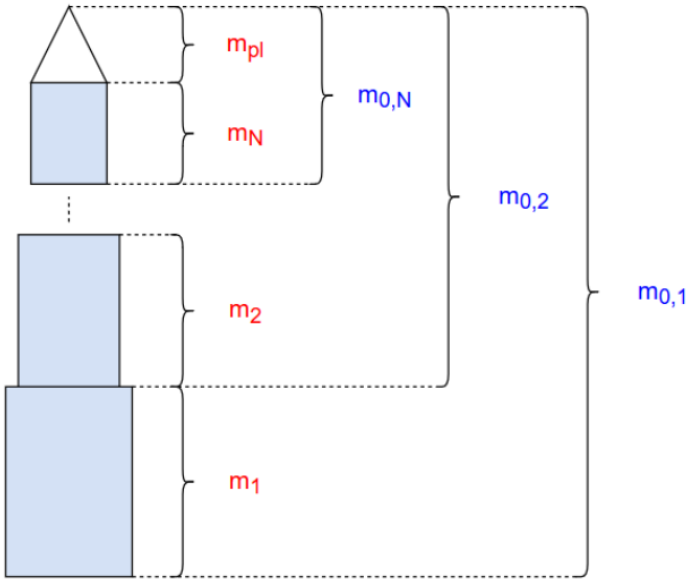

Figure 3.1: Serial Staging[2]

is then discarded.

The mass of the stage is the sum of those of the structure and propellant:

$$m_k = m_{s,k} + m_{p,k}$$

The initial stage mass is the sum of the stage mass as well as the payload mass for that particular stage:

$$m_{0,k} = m_k + m_{0,k+1}$$

The mass ratio and structural ratio for the k^{th} stage have been defined in Eqns. 3.7 and 3.8.

$$\Lambda_k = \frac{m_{0,k}}{m_{s,k} + m_{0,k+1}} \tag{3.7}$$

$$\epsilon_k = \frac{m_{s,k}}{m_k} \tag{3.8}$$

Using Eqns. 3.7 and 3.8, the gross liftoff mass (m_0) and the payload mass (m_{pl}) can be related as in Eqn. 3.9.

$$\frac{m_0}{m_{pl}} = \prod_{k=1}^{N} \frac{(1-\epsilon_k)\Lambda_k}{1-\epsilon_k\Lambda_k} \tag{3.9}$$

The optimisation problem is to minimize the logarithm of this ratio $\ln(\frac{m_0}{m_{pl}})$ under the constraint of Eqn. 3.6. Hence the optimization problem can be formulated as:

$$\begin{aligned} Minimize \qquad & \sum_{k=1}^{N} \ln\left(\frac{(1-\epsilon_k)\Lambda_k}{1-\epsilon_k\Lambda_k}\right) \\ Subject\ to\ the\ constraint \qquad & \Delta v_{mission} = \sum_{k=1}^{N} c_k \ln(\Lambda_k) \end{aligned}$$

3.3 Optimization Methodology

Lagrange multiplier method is used to solve the stated optimisation problem. The technique is described as follows for an optimisation problem defined as:

$$\begin{aligned} Minimize \qquad & f(x) \\ Subject\ to \qquad & g(x) = k \end{aligned}$$

The system of equations given below is solved to find the local extrema of the given problem:

$$\nabla f(x) - p\nabla g(x) = 0$$

$$g(x) = k$$

, where p is the Lagrange multiplier. This converts the constrained optimization problem to the equivalent of an unconstrained optimization problem, where the objective function is now called a Lagrangian function, to which the derivative test can be applied.

3.4 Method of Determining Solution

For our optimisation problem, the Lagrangian function will be Eqn. 3.10.

$$f^* = \sum_{k=1}^{N} \ln\left(\frac{(1-\epsilon_k)\Lambda_k}{1-\epsilon_k\Lambda_k}\right) + p\left(\sum_{k=1}^{N} c_k \ln(\Lambda_k) - \Delta v_{mission}\right) \tag{3.10}$$

Simplifying the logarithm term,

$$f^* = \sum_{k=1}^{N} (\ln(1-\epsilon_k) + \ln(\Lambda_k) - \ln(1-\epsilon_k\Lambda_k)) + p\left(\sum_{k=1}^{N} c_k \ln(\Lambda_k) - \Delta v_{mission}\right)$$

Upon differentiating this equation with respect to Λ_k and equating it to zero, we will obtain the values of the mass ratios Λ_ks in terms of the unknown Lagrangian parameter p and the known parameters c_k and ϵ_k as in Eqn. 3.11.

$$\Lambda_k = \frac{1 + pc_k}{pc_k\epsilon_k} \tag{3.11}$$

The constraint function of Eqn. 3.6, which is to be solved by the Netwon-Raphson method, can be written using the substitution from Eqn. 3.11 as Eqn. 3.12.

$$\sum_{k=1}^{N} c_k \ln\left(\frac{1+pc_k}{pc_k\epsilon_k}\right) = \Delta v_{mission} \tag{3.12}$$

Equation 3.12 is then solved using iterative root-finding methods like the Newton-Raphson method, for which an appropriate initial guess for the Lagrange multiplier p is required. Here, a suitable initial guess is taken as:

$$p_0 = \frac{-1}{\min(c_k(1-\epsilon_k))}$$

Once the solution to p is obtained, it can be substituted in Eqn. 3.11 to obtain the mass ratios for each of the stages and subsequently, the the structural and propellant masses for each stage can be obtained using Eqns. 3.7 through 3.9.

3.5 Results

The final orbit is a 500 km circular orbit and the Δv requirement is 10387.92 m/s. Our launch vehicle is taken to have 3 stages as the maximum decrement in gross liftoff mass is achieved between 2 and 3 stages, and further decrements in gross liftoff mass are not worth the complexity involved in increasing the number of stages. Further results have been discussed in Chapter 8.

Chapter 4

Propulsion Module

Propulsion is an important discipline in the Launch Vehicle Design. More than 80% of the mass of the launch vehicle comprises of Propellant mass, Optimizing the Launch Vehicle Propulsion helps in finding optimal gross lift off mass by reducing the proportion of propellant without altering the mission objective. Solid Propellant Rocket Engines, also known as **Solid Rocket Motors (SRMs)** are chosen for the current work. SRMs have no moving parts (contradicting the term "motor") and are the simplest design as compared to their counterpart, the Liquid Rocket Propellant Engine. An SRM consists of a motor case, insulator, igniter and propellant grain, and is shown in Fig. 4.1. SRMs are proven to be reliable and cost-effective propulsion systems for a wide range of aerospace applications starting from short range tactical missiles to large launch vehicles from their inception. This is due to their simplicity, manufacturing tractability, long-lifetime storage and short time needed for launching, and containing high chemical energy in a relatively small volume[22].

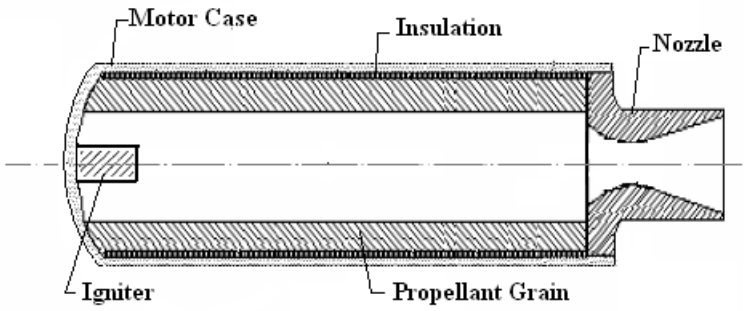

Figure 4.1: Main Parts of a Solid Propellant Rocket Motor[3]

4.1 Design of Solid Rocket Motors (SRMs)

The design of SRMs can be done in two ways, one is by taking them to be a part of the system such as a launch vehicle or a ballistic missile, and the other is by considering them as a separate system. In the former approach, SRM is considered as a subsystem and integrated with the other subsystems such as aerodynamics and trajectory to interact and produce highly optimized system model for the launch vehicle or ballistic missile. In the latter approach, propulsion is considered as a separate system, and is optimized such that it satisfies the overall mission requirements under specified constraints. In other words, the later approach has its own objective function but the former has an overall system objective function. In the current work, the two approaches are explored and the results are compared. propulsion subsystem as a part of system design is discussed in Chapter 7. The design methodology of SRM as a separate system is described below.

4.2 Methodology

The design process should be carried out in a logical sequence as shown in Fig. 4.2, the design parameters are to be identified as a first step. The parameters fall into the two categories: independent parameters and dependent parameters. Recognition of the distinction between these two prevents conflicting requirements and provides the designer the maximum degrees of freedom permitted by the design problem[14]. Propellant properties, mission/vehicle related requirements are usually considered as independent parameters. Nozzle configuration and propellant grain configuration and geometry are usually considered as the dependent parameters. In the current work, only propellant grain geometry is considered as dependent parameter to reduce the complexity and dimensionality of the design problem.

In the current work, the fixed constants, design constraints, requirements (total impulse, burning time, maximum chamber pressure, propellant type, material and grain configuration) are taken as independent parameters and once their values are fixed, the iterations can be done using optimization methods for dependent parameters (grain geometry) till it satisfies the requirements and constraints to get the optimal detailed design as shown in Fig. 4.2.

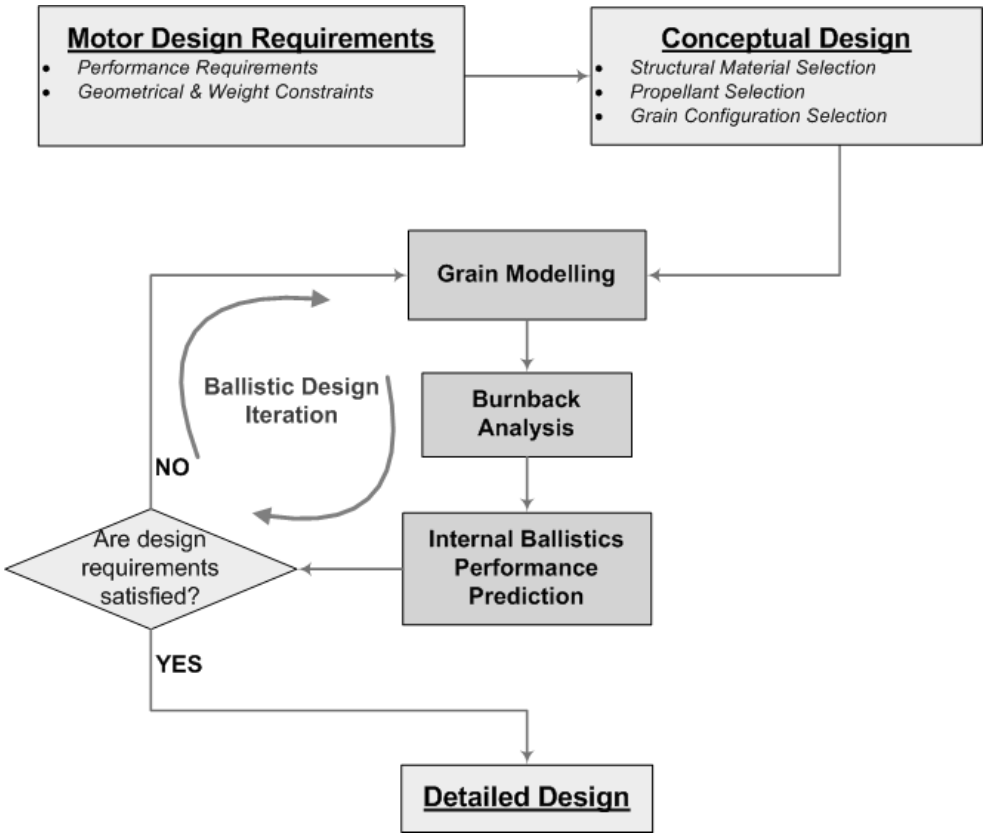

Figure 4.2: Design Process of a Solid Propellant Rocket Motor[4]

4.2.1 Independent Parameters

Two-dimensional single-segmented star grain configuration is selected for the current work because of the availability of the analytical design techniques, and the propellant type selected is the solid Hydroxyl Terminated Poly-Butadiene/Ammonium Perchlorate/Aluminium Power (HTPB/AP/Al) in the proportions 14:68:18. The motor case, nozzle, maximum chamber pressure and other configurations are taken from VEGA small launcher's 1st stage P80 Solid Rocket Motor[11]. The independent parameters are tabulated in Table. 4.1.

4.2.2 Grain Geometry and Burnback Analysis

The rate of propellant consumed at each instant is required to calculate ballistic parameters like chamber pressure, etc and it is entirely dependent on the burning surface at each instant. Knowing the evolution of the burning surface helps to find the variation of ballistic parameters throughout the burn time. This evolution of the burning surface is known as **Burn**

Table 4.1: Independent Parameters[11]

Propellant Properties	Propellant	HTPB/AP/Al: 14:68:18
	Density of solid propellant(ρ_{SP})	1770$\frac{kg}{m^3}$
	Chamber Temperature(T_{cc})	3328.9K
	Specific Heat Ratio(γ)	1.142
	Molar Mass(M_s)	27.84$\frac{kg}{kmol}$
	Gas Constant(R_{gas})	303.4479$\frac{J}{kg-K}$
	Burn Rate Coefficient(a)	3.5958×10^{-5}
	Burn Rate Exponent(n)	0.3
SRM Configuration	Maximum Chamber Pressure(P_{MEOP})	95 bar
	Nozzle Area Ratio($\frac{A_e}{A_t}$)	16
	Nozzle Deflection Angle(α)	6.5°

Back and it is purely a geometric analysis. There are various numerical and analytical techniques available for the Burn Back analysis. Analytical techniques provides simplicity and requires less computational power which is extremely helpful when the design problem has more variables. Hartfield et al. pioneered the analytical burn back analyses for different grain configurations[5]. Burn Back Analysis of the star grain configuration is described below.

The star grain configuration considered is defined by the five independent geometric parameters: the number of star points N, angular fraction ϵ, web thickness w, star curvature or fillet radius f, Inlet Radius R_i and y is the distance burned. A sample burn back diagram (left) and the geometric definition diagram for the star grain configuration is shown in Fig. 4.3. Only the summary of burn back equations are presented here, the detailed development of equations can be found in [23].

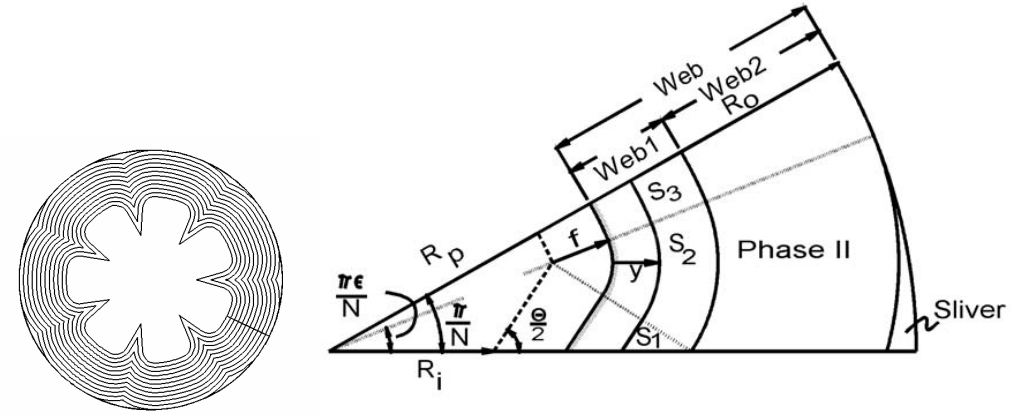

Figure 4.3: Burn Back of Star Grain (left) and Star Grain Geometry (right)[5]

Let us define $R_p = R_o - w - f$. Now the geometric relationship between the two primary angles θ and $\frac{\pi\epsilon}{N}$ in the geometry definition can be written as given by Eqn. 4.1.

$$\frac{\theta}{2} = tan^{-1}\left(\frac{R_p sin\left(\frac{\pi\epsilon}{N}\right) tan\left(\frac{\pi\epsilon}{N}\right)}{R_p sin\left(\frac{\pi\epsilon}{N}\right) - R_i tan\left(\frac{\pi\epsilon}{N}\right)}\right) \tag{4.1}$$

The burning of the cross-section is divided into three Burning Phases. In Phase I of burning, the burn perimeter can be written as the sum of two arcs and a straight line.

$$S = 2N\Bigg\{\frac{R_p sin\left(\frac{\pi\epsilon}{N}\right)}{sin\frac{\theta}{2}} - (y+f)cot\left(\frac{\theta}{2}\right) + (y+f)\left(\frac{\pi}{2} - \frac{\theta}{2} + \frac{\pi\epsilon}{N}\right) + (R_p + y + f)\left(\frac{\pi}{N} - \frac{\pi\epsilon}{N}\right)\Bigg\} \tag{4.2}$$

The Phase II burning occurs when the "flat" segment of Phase I burns out, which turns out occur when

$$(y+f) \geq \frac{R_p sin\left(\frac{\pi\epsilon}{N}\right)}{cos\left(\frac{\theta}{2}\right)} \tag{4.3}$$

The burn perimeter for Phase II can be written as in Eqn. 4.4.

$$S = 2N\Bigg\{(R_p + y + f)\left(\frac{\pi}{N} - \frac{\pi\epsilon}{N}\right) + (y+f)\left(\left(\frac{\pi}{2} - \frac{\theta}{2} + \frac{\pi\epsilon}{N}\right) - tan^{-1}\left(\frac{\sqrt{(y+f)^2 - \left(R_p sin\frac{\pi\epsilon}{N}\right)^2}}{R_p sin\frac{\pi\epsilon}{N}}\right) - \frac{\theta}{2}\right)\Bigg\} \tag{4.4}$$

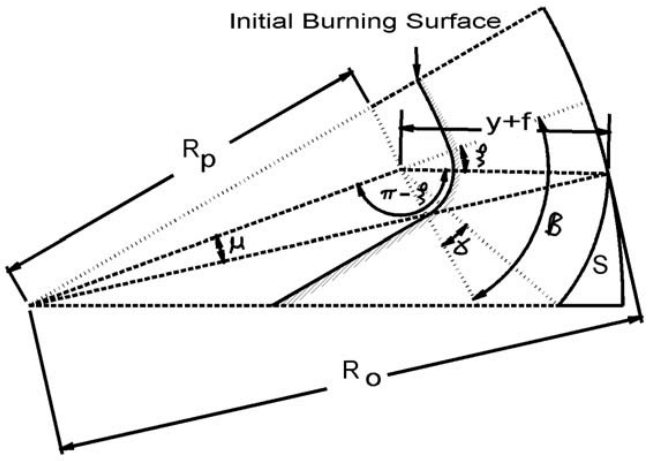

Figure 4.4: Phase III Burning of the Star Grain[5]

Phase III burning or the tail-off phase burning is the most important phase and the burning is due to leftover geometry after Phase II burning known as Sliver. The amount of sliver determines the active burning time of the SRM. The Phase II burning occurs when the burn distance (y) is more than wed thickness (w). The equations of this phase are developed by considering the initial burning surface as shown in Fig. 4.4.

The angles β and γ are used in the development of phase II burning and can be represented as:

$$\beta = \left(\frac{\pi}{2} - \frac{\theta}{2} + \frac{\pi\epsilon}{N}\right) \tag{4.5}$$

$$\gamma = \left(\frac{\sqrt{(y+f)^2 - \left(R_p sin\left(\frac{\pi\epsilon}{N}\right)\right)^2}}{R_p sin\left(\frac{\pi\epsilon}{N}\right)}\right) - \frac{\theta}{2} \tag{4.6}$$

Using the law of cosines, ξ can be determined as follows:

$$\xi = \pi - cos^{-1}\left(\frac{-\left(R_o^2 - R_p^2 - (y+f)^2\right)}{2R_p(y+f)}\right) \tag{4.7}$$

The burn perimeter, a section of an arc, becomes:

$$S = 2N\left\{(y+f)[\beta - \gamma - \xi]\right\} \tag{4.8}$$

The maximum burn distance that can occur can be arrived at using the diagram in Figs. 4.3

and 4.4 and may be expressed as in Eqn. 4.9.

$$y_{max} = \sqrt{\left(R_o - R_p cos\left(\frac{\pi\epsilon}{N}\right)\right)^2 + \left(R_p sin\left(\frac{\pi\epsilon}{N}\right)\right)^2} - f \tag{4.9}$$

The burning surface for all the three phases can be obtained by multiplying the burn parameter with length of the solid propellant or motor.

$$A_b = S \times L \tag{4.10}$$

4.2.3 Internal Ballistic Performance Analysis

The accuracy of simulation/prediction of internal ballistic analysis is dependent on the physical model used. There are some physical and chemical processes that occur during the motor operation still not well understood, hence, the physical models have been simplified based on several assumptions. A Zero-Dimensional (0-D) analytical model that is based on the mass conservation law with constant temperature and thermodynamic relations was used in the current work.

4.2.3.1 Ballistic Parameters

4.2.3.1.1 Burn Rate

The rate at which a propellant burns is described by a reference value at a specific pressure (usually 1000 psi or 6.895$\frac{N}{m^2}$). With appropriate constants the burn rate (r) can be represented by an analytical expression that defines burn rate as a function of pressure (P_c) at a given grain conditioned temperature.

$$r = aP_c^n \tag{4.11}$$

, where a is the burn rate coefficient and n is the burn rate exponent. The above equation is also known as Saint Robert's burn rate law. Note that the above power law is obtained by plotting log/log plot of experimental pressure vs burn rate. It is an empirical relation rather than a mathematically derived equation. If we relax the constant temperature assumption, additional "temperature-sensitivity constants" are required to define burning rate values at different grain temperatures.

4.2.3.1.2 Characteristic Velocity

Characteristic Velocity (c^*), is a function of propellant properties and combustion cham-

ber design and independent of nozzle design. The characteristic velocity can be written as in Eqn. 4.12.

$$\begin{aligned} c^* &= \frac{P_c A_t}{\dot{m}} \\ &= \frac{\sqrt{\gamma R T_c}}{\gamma \sqrt{\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}}} \end{aligned} \tag{4.12}$$

We can see that c^* is proportional to $\sqrt{\frac{T_c}{M_c}}$. The c^* is used in comparing the relative performance of different chemical rocket propulsion system designs and propellants.

4.2.3.1.3 Specific Impulse

Specific Impulse (I_{sp}) is a measure of the impulse or momentum change that can be produced per unit mass propellant consumed. I_{sp} is the ratio of motor thrust to the mass flow rate multiplied by gravity constant and is an important parameter in determining propellant weight necessary to meet the ballistic requirements.

$$I_{sp} = \frac{c^* C_F}{g_0} \tag{4.13}$$

Actual c^* and I_{sp} delivered in the motor are less than the theoretical values by a significant amount. The reduction in the values are due to fluid losses including two-phase flow, heat-losses to motor hardware and combustion inefficiency. The total value losses in the c^* is expressed as η_θ known as c^* efficiency factor. Performance losses in the C_F are due to nozzle divergent loss (λ) and nozzle efficiency factor (η_F). An accurate estimate of the delivered specific impulse in the motor is obtained from,

$$I_{sp,d} = \eta_\mu I_{sp} \tag{4.14}$$

, where $I_{sp,d}$ is the delivered specific impulse, η_μ is the deliverable motor efficiency($\approx \eta_F \eta_\theta$). The value of η_μ varies from 0.93 or less in small motors to 0.96 in large motors.

The above parameters are the propellant properties or independent parameters which are fixed values for a given propellant.

4.2.3.1.4 Nozzle Area Ratio

Nozzle expansion ratio is defined as the ratio of nozzle exit area to the nozzle throat area as given below. The theoretical expansion ratio for the optimum expansion can be calculated by using this isentropic formula as given in Eqn. 4.15.[3].

$$\frac{A_t}{A_e} = \left(\frac{\gamma+1}{2}\right)\left(\frac{P_e}{P_c}\right)^{\frac{1}{\gamma}} \sqrt{\frac{\gamma+1}{\gamma-1}\left[1-\left(\frac{P_e}{P_c}\right)^{\frac{\gamma-1}{\gamma}}\right]} \tag{4.15}$$

4.2.3.1.5 Thrust Coefficient

Thrust coefficient C_F is defined as the thrust divided by the chamber pressure and the throat area A_t. The ideal vaccum thrust coefficient $C_{F,vac}$ and ideal thrust coefficient C_F are shown in Eqns. 4.16 4.17.

$$C_F = \sqrt{\frac{2\gamma^2}{\gamma-1}\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}\left[1-\left(\frac{P_e}{P_c}\right)^{\frac{\gamma-1}{\gamma}}\right]} + \left(\frac{P_e - P_{amb}}{P_c}\right)\left(\frac{A_e}{A_t}\right) \tag{4.16}$$

$$C_{F,vac} = \sqrt{\frac{2\gamma^2}{\gamma-1}\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}\left[1-\left(\frac{P_e}{P_c}\right)^{\frac{\gamma-1}{\gamma}}\right]} + \left(\frac{P_e}{P_c}\right)\left(\frac{A_e}{A_t}\right) \tag{4.17}$$

The actual thrust will be less than the ideal thrust because of losses that are mentioned in above section. The factors λ and η_F are applied to the momentum term in the ideal thrust equation to obtain actual thrust. The vacuum thrust coefficient is only a function of exit pressure to chamber pressure ratio and nozzle area ratio. It is convenient to write actual thrust coefficient in terms of vacuum thrust coefficient[14]. The ideal thrust can be written as in Eqn. 4.18.

$$\begin{aligned} C_{F,act} &= \lambda\eta_F \left(\sqrt{\frac{2\gamma^2}{\gamma-1}\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}\left[1-\left(\frac{P_e}{P_c}\right)^{\frac{\gamma-1}{\gamma}}\right]}\right) + \left(\frac{P_e - P_{amb}}{P_c}\right)\left(\frac{A_e}{A_t}\right) \\ &= \lambda\eta_F \left(C_{F,vac} - \left(\frac{P_e}{P_c}\right)\left(\frac{A_e}{A_t}\right)\right) + \left(\frac{P_e - P_{amb}}{P_c}\right)\left(\frac{A_e}{A_t}\right) \end{aligned} \tag{4.18}$$

The nozzle divergent loss can be written as Eqn. 4.19.

$$\lambda = \left(\frac{1 + cos\alpha}{2}\right) \tag{4.19}$$

, where α is the nozzle deflection angle.

4.2.3.2 Zero-Dimensional Ballistic Analysis

4.2.3.2.1 Steady State Mass Balance The condition for steady state operation is that the propellant mass flow rate generated in the combustion chamber should be equal to the mass flow rate of gas products discharged through the nozzle.

$$\dot{m}_c = rA_b\rho_{SP} \qquad \dot{m}_e = \frac{P_cA_t}{c^*}$$

$$\dot{m}_c = \dot{m}_e$$

$$P_c = \left(\frac{A_b a c^* \rho_{SP}}{A_t}\right)^{\frac{1}{1-n}} \tag{4.20}$$

The thrust can be written in terms of Eqn. 4.20 as Eqn. 4.21.

$$F = C_{F,act}P_cA_t \tag{4.21}$$

The independent parameters (propellant properties, nozzle configuration) the values of burn rate constants, gas properties and nozzle area ratio are known values, using the Eqn. 4.15 the exit pressure to chamber pressure ratio is calculated using Newton-Raphson method and is constant throughout the process. The analysis is started by selecting the geometry parameters N,ϵ, w, f, R_i, then the evolution of burning surface is calculated by increasing the burn distance at every instant by amount say, 1 mm. From this, the chamber pressure is calculated at every instant and from chamber pressure the thrust is calculated at every instant to get the thrust time profile.

4.3 Problem Statement

The whole process described above is done repeatedly using optimization solver till the geometry parameters maximize the total impulse subjected to chamber pressure and burn time constraints. The problem statement can be expressed as :

$$
\begin{aligned}
&Maximize \qquad \int_{o}^{tb} F dt \\
&With\ respect\ to\ z = \{N, \epsilon, w, f, R_i\} \\
&Subject\ to \\
&\qquad\qquad P_c \leq P_{MEOP} \\
&\qquad\qquad tb \leq t_0
\end{aligned}
$$

4.4 Implementation

In the current work, P_{MEOP} is 95 bar and t_0 is 150 sec. The lower and upper bounds of the design variables are shown in Tab. 4.2. The problem statement for the SRM design as a part of system is presented in Chapter 7. The above optimization problem is solved using Hybrid Optimization method PSO guided SQP. The results are shown in Chapter 8.

Table 4.2: Lower and Upper bounds for the design variables[11]

	Lower Bound	Upper Bound
N	3	15
ϵ	0.2	0.9
w	0.3	0.7
f	0.01	0.15
R_i	0.01	0.5

Chapter 5

Aerodynamic Analysis

The aerodynamic analysis of the vehicle consists of calculating the coefficient of drag at various mach numbers and visualising the shock formation and flow contours at subsonic and supersonic mach numbers. ANSYS Fluent is used for the analyses and the drag coefficient variation was plotted against mach number with the help of MATLAB.
The flow is considered to be axisymmetric (about the axis of the launch vehicle) and perfectly axial. These assumptions hold accurately at the velocities that we are analysing the flow over the launch vehicle at. Also, sea level conditions have been taken as reference values for the analyses.

5.1 Geometry Modeling

DesignModeler is used for generating the geometry in 2-D (because axisymmetric flow is being assumed). The dimensions of the launch vehicle have been found out to be diameter of 2 m and 18 m length. The nose radius has been taken to be 0.5 m, and the fillet radius is taken to be 1.5 m as shown in Fig. 5.1.

A space of 30 m has been provided around the vehicle for analysing the fluid behaviour over and around the launch vehicle as shown in Fig. 5.2. Also, this enclosure has been divided into various parts so that meshing the surface is easier and the quality of the mesh generated is high.

The geometry is such that if it is rotated about the axis of the launch vehicle, then we would get an axisymmetric enclosure of the shape of the fluid body we are analysing around the (hollow) rocket.

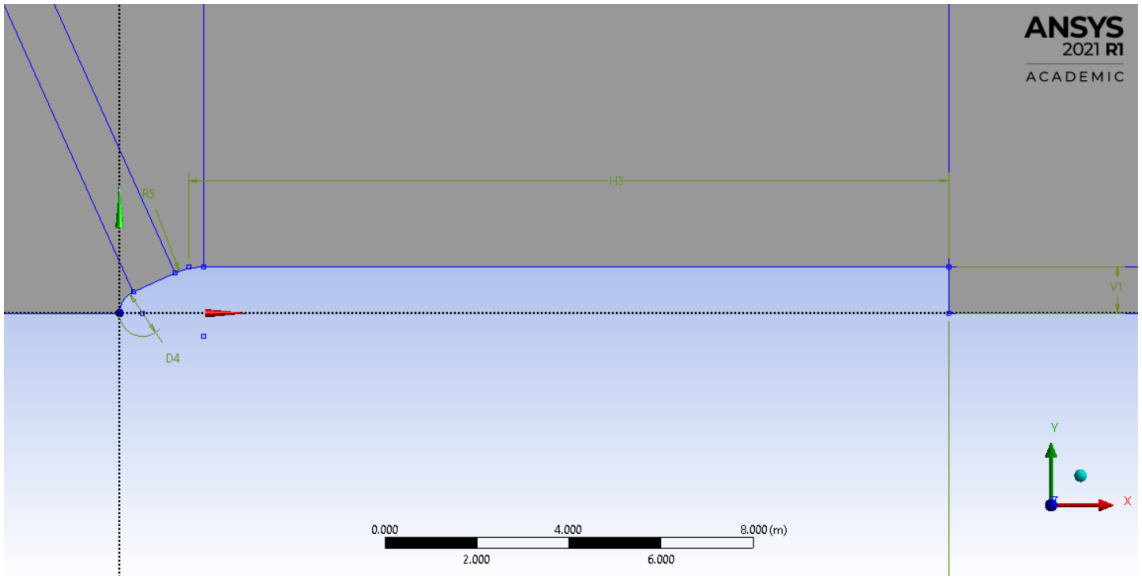

Figure 5.1: Rocket Geometry

5.2 Meshing

Face meshing was used to mesh the planar geometry as shown in Fig. 5.3 on each of the surfaces individually. Thereafter, mesh sizing was applied to each of the edges to refine the mesh appropriately and biasing was given so that the mesh is more concentrated in the vicinity of the rocket body as compared to the further reaches of the enclosure, as is observed in Fig. 5.4. 184921 nodes and 184000 elements were formed.

The mesh quality parameters has been tabulated in Tab. 5.1 and the named selections created for ease of analysis have been displayed in Fig. 5.5.

Table 5.1: Mesh Quality Parameters

Mesh Metric	Average Value	Minimum or Maximum Value
Element Quality	0.749	Minimum 0.130
Aspect Ratio	2.515	Maximum 15.212
Skewness	8.1522×10^{-4}	Maximum 0.00913
Orthogonal Quality	0.99999	Minimum 0.9999

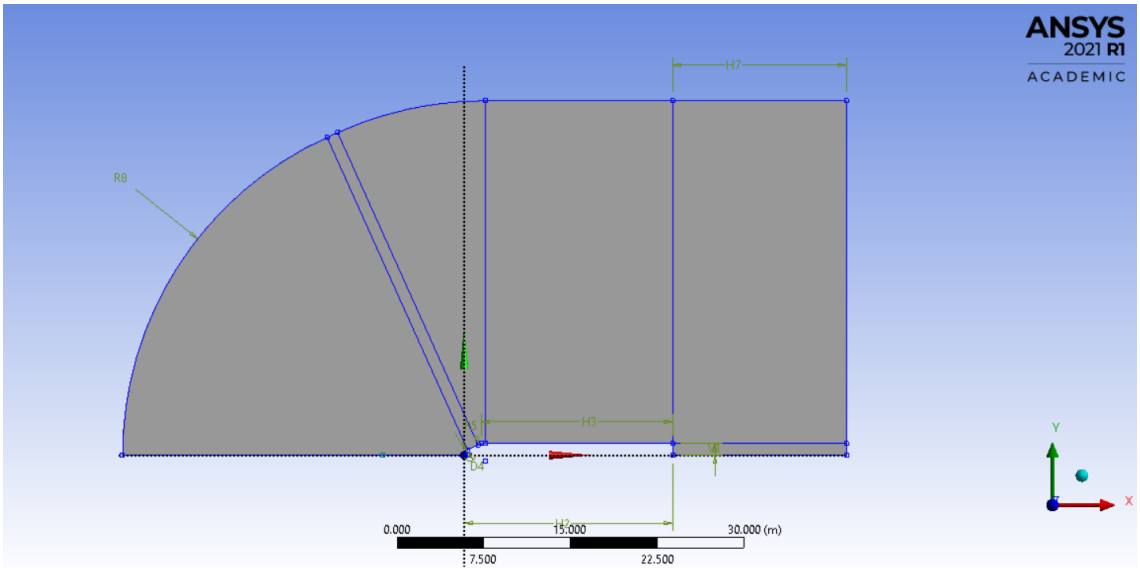

Figure 5.2: Enclosure

5.3 Flow Analysis in Fluent

Density-based, steady, axisymmetric solver has been used. Energy equation is enabled and Standard k-omega model is used for viscosity. Air is considered to be an ideal gas and Sutherland's 3-coefficient law is used to model its viscosity. Fluent automatically detects zones based on the named selections defined during meshing. The inlet is given an axial velocity according to the mach number the flight is being simulated and the corresponding mach number is designated at the pressure far-field. The reference values have been taken as sea level values and are shown in Fig. 5.6 along with the mesh as visible in ANSYS. The cross sectional area of the launch vehicle is taken as the reference area and the reference velocity corresponds to the mach number being simulated.

First order upwind schemes are used for flow, turbulent kinetic energy and specific dissipation rate. The drag coefficient is defined in the Report Definitions tab and the solution is initialised by computing values from the pressure far-field. The calculation is then allowed to run for 2000 iterations, but the solution converges much sooner as shown in Fig. 5.7. The results have been discussed in Chapter 8.

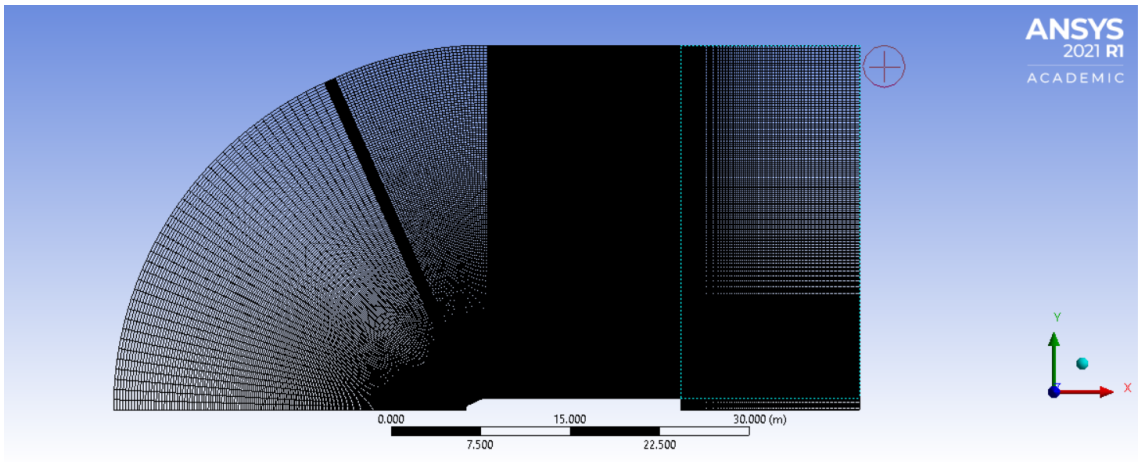

Figure 5.3: Face Mesh

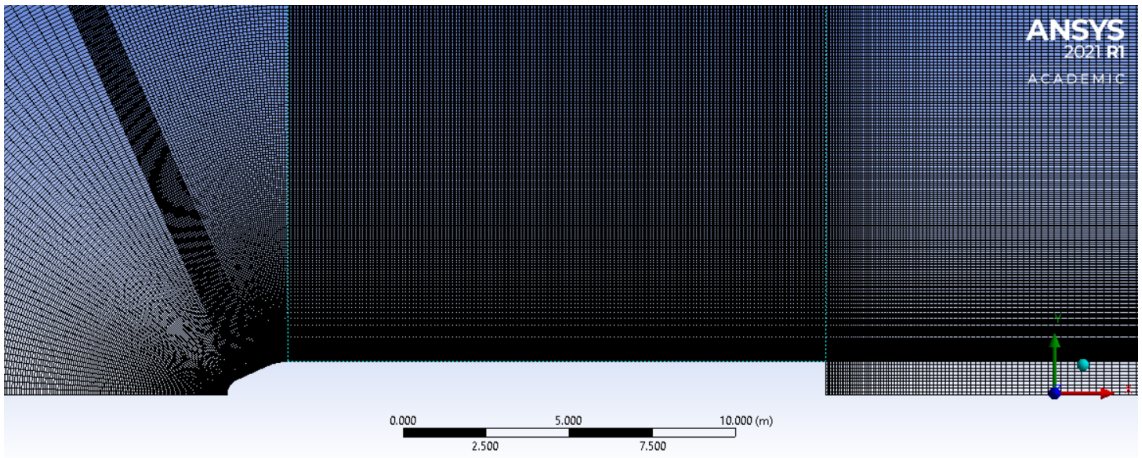

Figure 5.4: Mesh Biasing to Refine Mesh Closer to Rocket Body

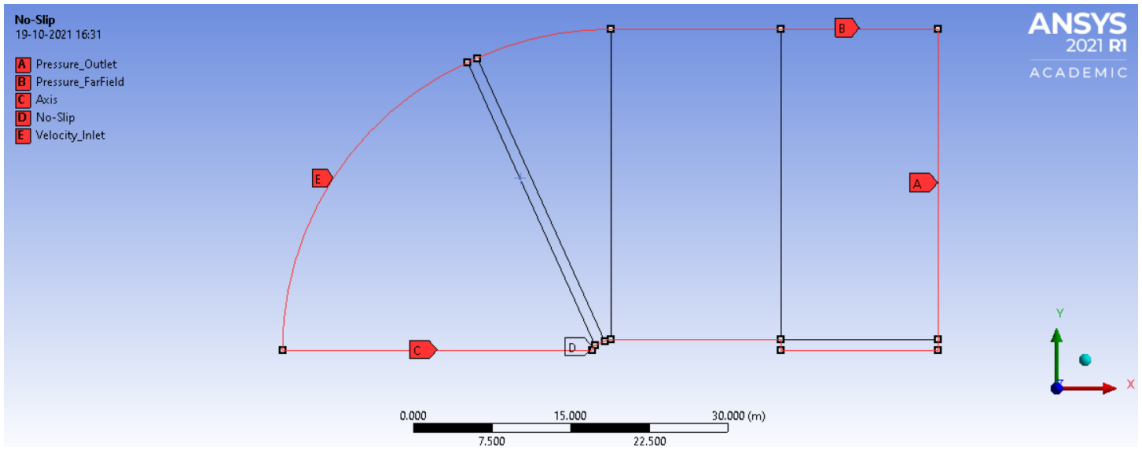

Figure 5.5: Named Selections

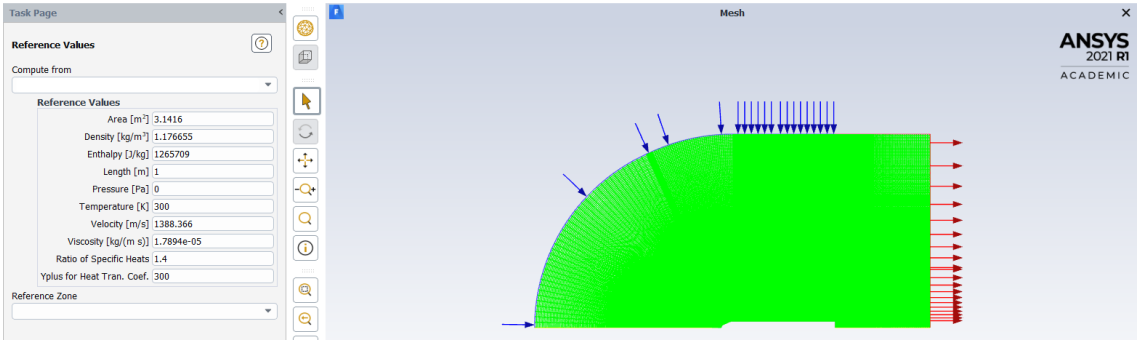

Figure 5.6: Reference Values and ANSYS Fluent Mesh

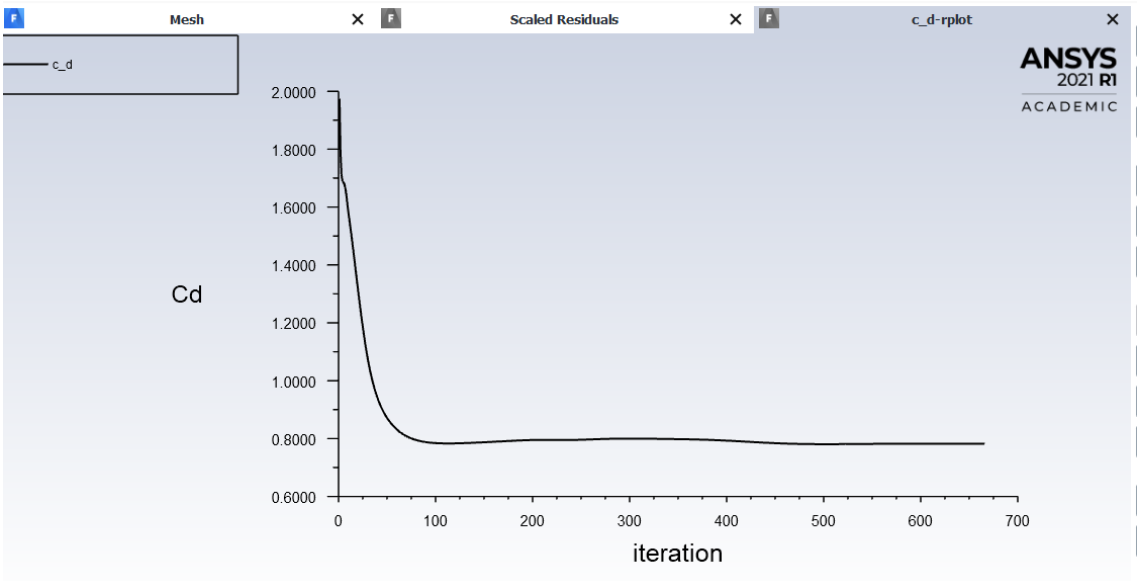

Figure 5.7: Drag Coefficient Convergence

Chapter 6

Trajectory Module

The optimisation of the trajectory of a launch vehicle is perhaps the most fundamental component of designing the vehicle. But before simulating the ascent trajectory, the various co-ordinate frames need to be defined and discussed. These co-ordinate frames are not just crucial for solving the equations of motion, but also for the definition of various attitude angles which describe the orientation of the spacecraft and its trajectory. The formulation followed here is inspired from the previous work[2].

6.1 Co-ordinate Frames

The co-ordinate frames employed for determining and calculating the various necessary parameters for trajectory design are described:

6.1.1 Earth-Centered Inertial (ECI) Frame

The origin of ECI frame is the center of the Earth, the z-axis points towards the north pole and the x-axis points towards the point of intersection of the Greenwich Meridian and the equatorial plane at time zero. The y-axis is defined such that it completes a right hand system. This coordinate frame is used to solve the translational equations of motion.

The Earth-Centered Inertial Co-ordinate Frame has been displayed in Fig. 6.1. This frame is fixed at time zero, i.e. at the beginning of the simulation, hence the z-axis points towards the North Pole along the true-of-date rotational axis of the Earth. The Equatorial Plane is also the true-of-date equatorial plane at time zero.

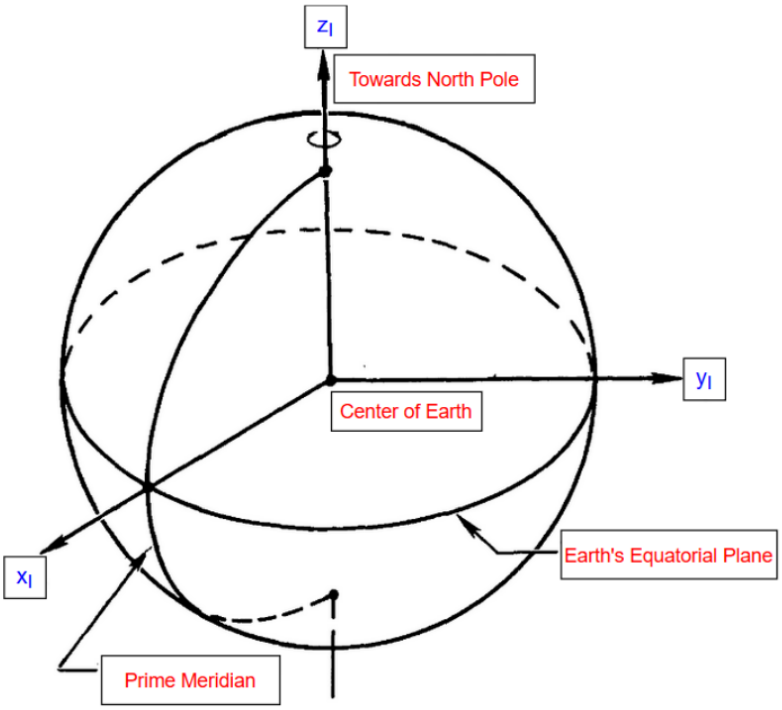

Figure 6.1: Earth-Centered Inertial Co-ordinate Frame [6]

6.1.2 Geographic (G) Frame

The G frame is on the surface of the Earth and its origin is the location of interest, in this case, the latitude and longitude of the spacecraft. The x-y plane of the frame is the local horizontal plane, and the x-axis points towards North and the y-axis towards East. The z-axis completes a right hand system. This frame is also referred to as the Local Horizontal-Local Vertical (LH-LV) frame, and has been showed in Fig. 6.2.

6.1.3 Inertial Launch (L) Frame

The L frame is located at the geodetic latitude and inertial longitude of the vehicle at the beginning of the trajectory, i.e. when it is stationed at the Launch Pad at time $t = 0$. This frame is used as an inertial reference system from which the inertial attitude angles (roll, pitch, and yaw) are measured.

The latitude and longitude of the Launch Station (ϕ_L and θ_L respectively), and the Launch Azimuth (Az_L), which is the angle measured clockwise from the local North at the Launch Station to the projected direction of the trajectory on the surface of the Earth, are shown in the L frame in Fig. 6.2.

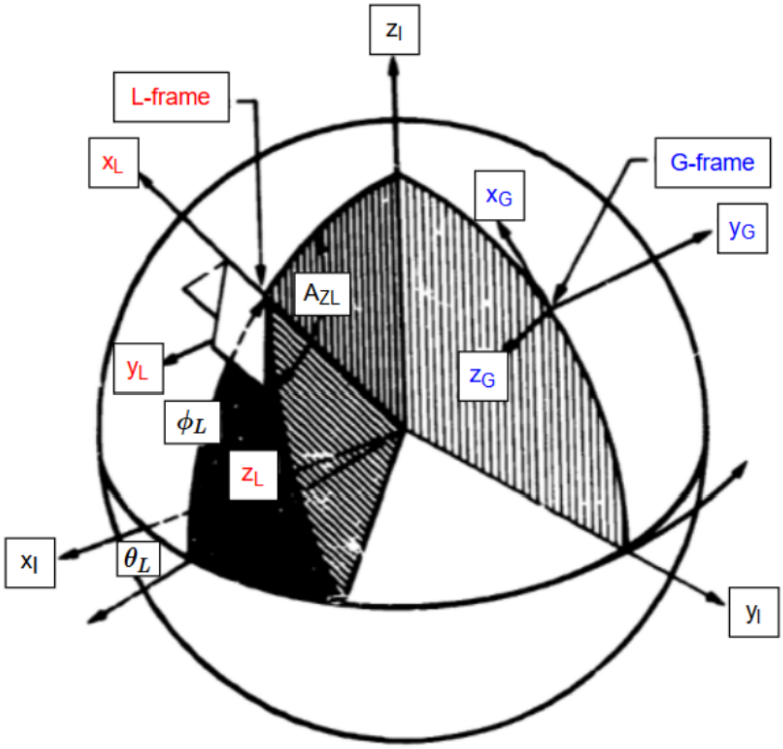

Figure 6.2: Geographic and Inertial Launch Co-ordinate Frames [7]

6.1.4 Body (B) Frame

The origin of the B frame the center of gravity of the spacecraft and its axes are aligned with those of the spacecraft as shown in Fig. 6.3: the x-axis is directed along the longitudinal axis of the spacecraft, the y-axis points outward along the right wing and the z-axis points downward so as to complete a right handed system. All the aerodynamic and thrust forces are calculated in the B frame and then transformed to the Inertial frame, where they are combined with gravitational forces.

6.1.5 Atmospheric Relative Velocity System (ARVS) Frame

The ARVS frame is shown in Fig. 6.4. Once the translational equations of motion are solved in the ECI frame, the key variables computed are used in the calculation of the relative position and velocity, which take into account the effect of the rotation of the Earth. However, calculation of the relative components must also account for the local atmospheric velocity conditions, which is carried out using the ARVS. The local relative wind

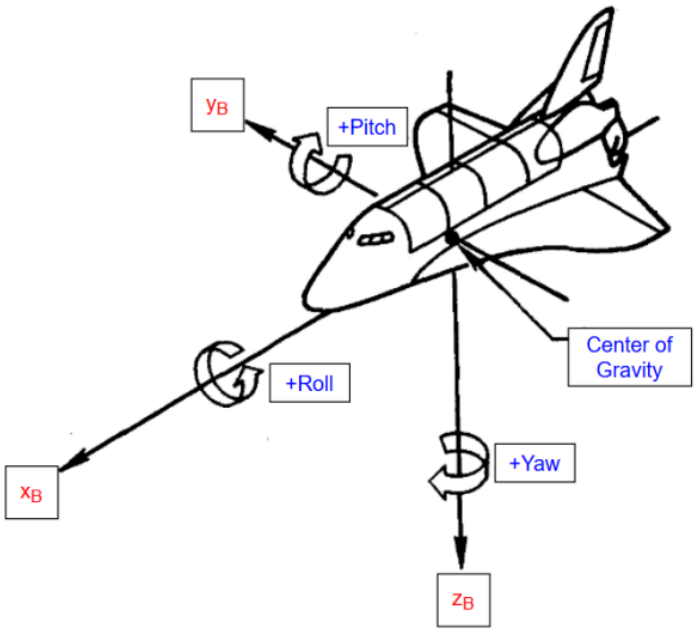

Figure 6.3: Body Co-ordinate Frame [6]

vector can be resolved into components along the body frame, or the angles between the corresponding axes can be used for steering determination, as is highlighted in the Gravity Turn phase of the trajectory, where this frame is especially useful.

6.2 Equations of Motion

The equations of motion for a launch vehicle in an inertial co-ordinate system can be written as in the equations of Eqn. 6.1.

$$\begin{aligned} \dot{\mathbf{r}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \mathbf{g}(\mathbf{r}) + \frac{\mathbf{T}}{m(t)} + \frac{\mathbf{A}}{m(t)} + \frac{\mathbf{N}}{m(t)} \\ \dot{m} &= \frac{-T_v}{g_0 I_{sp}} \end{aligned} \tag{6.1}$$

, where $\mathbf{r}$ is the position vector from the center of the Earth to the center of gravity of the launch vehicle, $\mathbf{v}$ is the inertial velocity vector of the vehicle, $\mathbf{g}$ is the acceleration due to gravity varying with radius vector (or varying with altitude), g_0 is the magnitude of the acceleration due to gravity at the surface of the Earth, i.e. at radius r_0, T_v is the magnitude

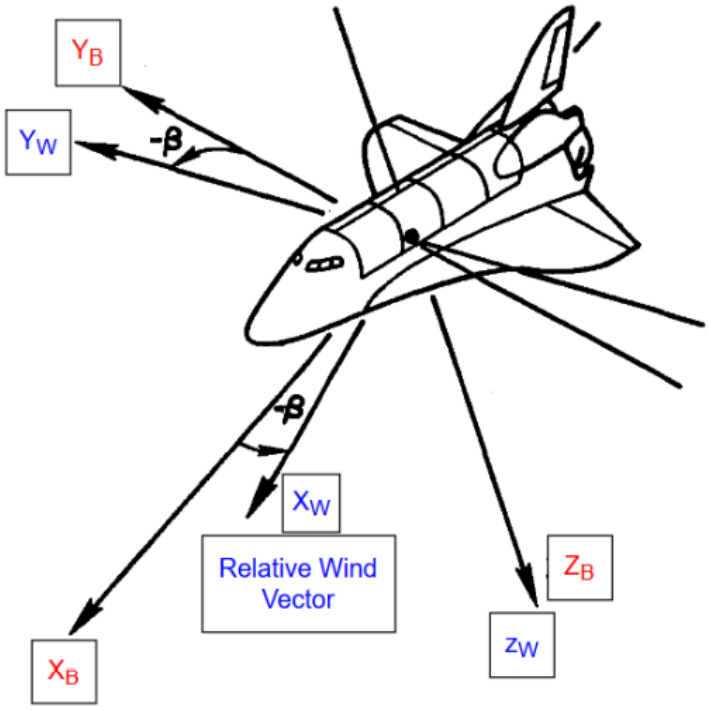

Figure 6.4: Relative Velocity Conditions in the Atmospheric Relative Velocity System Co-ordinate Frame [6]

of thrust in vacuum (maximum thrust), and I_{sp} is the specific impulse of the vehicle at the current instant. $\mathbf{T}$ is the current thrust vector, and $\mathbf{A}$ and $\mathbf{N}$ are the axial and normal aerodynamic forces.

The contributions to the vehicle's acceleration due to the thrust and aerodynamic forces is taken directly from the body frame and transformed to the inertial frame using the appropriate transformation matrix. The above equations of motion can be non-dimensionalized in order to get better computational results.

6.3 Attitude Angles

Euler angles and aerodynamic angles are used to characterise the attitude of the spacecraft and they can be defined in both inertial and relative frames.

- **Euler angles :** The inertial Euler angles are the attitude angles measured in the inertial launch frame whereas the relative Euler angles are described as follows:

1. Roll angle (ϕ) - The relative roll angle is measured as the rotation about the x-axis in the body frame.
2. Yaw angle (ψ) - The azimuth angle of the x_B-axis measured clockwise from the x_G-axis.
3. Pitch Angle (θ) - The elevation of the x_B-axis above the local horizontal axis (in the G frame).

Roll, yaw and pitch motions can be schematically visualised in Fig. 6.3.

- **Aerodynamic angles :** The relative aerodynamic angles are defined in the ARVS system whereas inertial aerodynamic angles are defined in the inertial frame.

  1. Angle of Attack (α) - A nose-up motion corresponds to a positive α.
  2. Sideslip Angle (β) - A nose-left motion corresponds to a positive β.
  3. Bank Angle (σ) - A positive rotation about the velocity vector (relative or inertial), such that the right wing dips downward, corresponds to a positive σ.

6.4 Co-ordinate Transformations

To be able to maintain consistency between the various coordinate systems while transferring data, proper transormations are required. A crucial transformation is the Inertial to Body Frame Transformation Matrix because the inverse of this matrix is used to transform the acceleration determined in the body frame to the ECI frame. The IB matrix can be determined in different ways depending on the angles to be used. The inverses of these matrices can be obtained by transposing the elements because they are orthonormal matrices. The transformation matrices to be used during the simulation of the trajectory for the current work are described as follows:

1. ECI to Inertial Launch Frame:
   This matrix depends on the latitude and longitude of the launch pad as well as the Launch Azimuth. For the Indian launch pad stationed at Kulasekharapatnam, the latitude and longitude are respectively:

   $\phi_L = 8.3989^\circ$ N
   $\theta_L = 78.0524^\circ$ E

$$IL = \begin{bmatrix} \cos(\phi_L)\cos(\theta_L) & \cos(\phi_L)\sin(\theta_L) & \sin(\phi_L) \\ \sin(\phi_L)\cos(\theta_L)\sin(Az_L) & \sin(\phi_L)\sin(\theta_L)\sin(Az_L) & -\cos(\phi_L)\sin(Az_L) \\ -\cos(Az_L)\sin(\theta_L) & +\cos(Az_L)\cos(\theta_L) & \\ -\sin(\phi_L)\cos(\theta_L)\cos(Az_L) & -\sin(\phi_L)\sin(\theta_L)\cos(Az_L) & \cos(Az_L)\cos(\phi_L) \\ -\sin(Az_L)\sin(\theta_L) & +\sin(Az_L)\cos(\theta_L) & \end{bmatrix}$$

2. Inertial Launch Frame to Body Frame:
   The inertial attitude angles are used to compute this matrix.

$$LB = \begin{bmatrix} \cos(\psi_I)\cos(\theta_I) & \cos(\phi_I)\sin(\psi_I)\cos(\theta_I) & \sin(\phi_I)\sin(\psi_I)\cos(\theta_I) \\ & +\sin(\phi_I)\sin(\theta_I) & -\cos(\phi_I)\sin(\theta_I) \\ -\sin(\psi_I) & \cos(\phi_I)\cos(\psi_I) & \sin(\phi_I)\cos(\psi_I) \\ \cos(\psi_I)\sin(\theta_I) & \cos(\phi_I)\sin(\psi_I)\sin(\theta_I) & \sin(\phi_I)\sin(\psi_I)\sin(\theta_I) \\ & -\sin(\phi_I)\cos(\theta_I) & +\cos(\phi_I)\cos(\theta_I) \end{bmatrix}$$

3. Geographic to ARVS Frame:
   The atmospheric relative flight azimuth and flight path angles are used to compute the GA matrix.

$$GA = \begin{bmatrix} \cos(\gamma_A)\cos(\lambda_A) & \cos(\gamma_A)\sin(\lambda_A) & -\sin(\gamma_A) \\ -\sin(\lambda_A) & \cos(\lambda_A) & 0 \\ \sin(\gamma_A)\cos(\lambda_A) & \sin(\gamma_A)\sin(\lambda_A) & \cos(\gamma_A) \end{bmatrix}$$

4. ARVS to Body Frame:

$$AB = \begin{bmatrix} \cos(\alpha)\cos(\beta) & -\cos(\alpha)\sin(\beta)\cos(\sigma) & -\cos(\alpha)\sin(\beta)\sin(\sigma) \\ & +\sin(\alpha)\sin(\sigma) & -\sin(\alpha)\cos(\sigma) \\ \sin(\beta) & \cos(\beta)\cos(\sigma) & \cos(\beta)\sin(\sigma) \\ \sin(\alpha)\cos(\beta) & -\sin(\alpha)\sin(\beta)\cos(\sigma) & -\sin(\alpha)\sin(\beta)\sin(\sigma) \\ & -\cos(\alpha)\sin(\sigma) & +\cos(\alpha)\cos(\sigma) \end{bmatrix}$$

5. ECI to G Frame:
   The IG matrix can directly be computed using the geocentric latitude and inertial longitude, which are determined as:

$$\phi_c = \sin^{-1}(\frac{r_{z,I}}{r_I})$$

$$\theta_I = \sin^{-1}(\frac{r_{y,I}}{r_{x,I}})$$

   , where $r_{x,I}$, $r_{y,I}$ and $r_{z,I}$ are the components of the inertial position vector and r_I is the magnitude of that vector.

$$GA = \begin{bmatrix} -\sin(\phi_c)\cos(\theta_I) & -\sin(\phi_c)\sin(\theta_I) & \cos(\phi_c) \\ -\sin(\theta_I) & \cos(\theta_I) & 0 \\ -\cos(\phi_c)\cos(\theta_I) & -\cos(\phi_c)\sin(\phi_c) & -\sin(\theta_I) \end{bmatrix}$$

6. ECI to Body Frame:
   One way to compute the IB matrix is using the IL and LB matrices.

$$IB = IL \times LB$$

   Alternatively,

$$IB = (IG \times GA) \times AB$$

The sequences of transformations are shown in the network of Fig. 6.5.

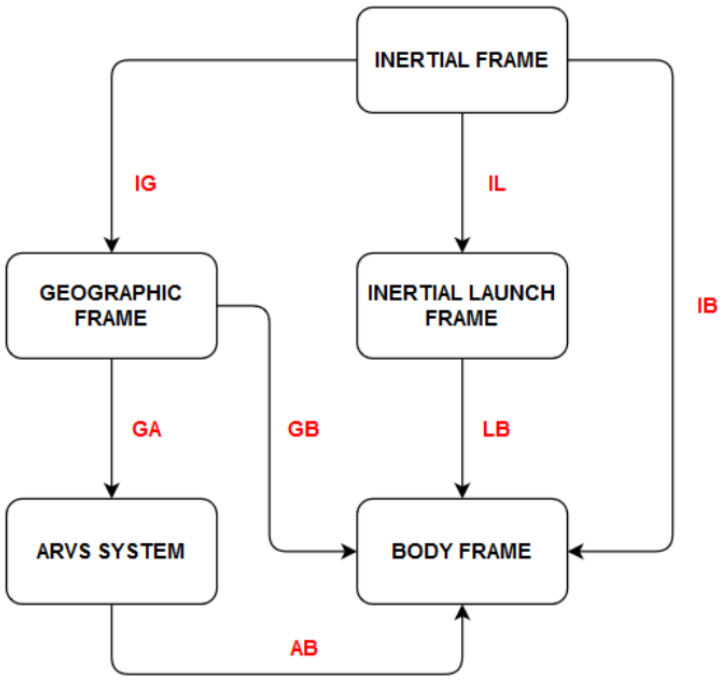

Figure 6.5: Sequences of Co-ordinate Transformations[2]

6.5 Phases of the Ascent Trajectory

As it can be seen from Fig. 1.1, the ascent trajectory is happening in two major atmospheric phases: endo-atmospheric phase and exo-atmospheric phase. The optimization of trajectory is omitted in the endo-atmospheric phase due to the presence of dense atmosphere which may result in large aerodynamic stresses that the vehicle would not withstand. The trajectory is optimized in the exo-atmospheric phase due to the absence of atmosphere and atmospheric loads. The trajectory within the endo-atmospheric phase is in turn divided into several phases which are explained below.

1. **Vertical Take-off Phase:** The launch vehicle lifts off vertically for a number of reasons one of which is to cross the dense atmosphere at relatively low speeds with spending less time. The main purpose of vertical take-off phase is to safely clear the launch pad and umbilical tower as failing to do so may results in collision of launch vehicle with tower which would result in catastrophic losses. The necessary condition for the take-off is that the initial thrust produced must be greater than the total weight of the launch vehicle other the vehicle has to bear the unnecessary cost of fuel consumption. The vertical take-off phase lasts till the launch vehicle reaches

50 m above the ground (100 m for heavy launch vehicles).

2. **Constant Turn-over/Pitch-over:** Constant Turn-over/Pitch-over phase starts after the completion of vertical take-off phase. This phase is very important in endo-atmospheric phase because the gravity will decelerate the vehicle most when it is in vertical flight and the vehicle has to pitch down and orient itself suitable for the subsequent gravity turn phase. Another advantage is that the velocity of the vehicle is lesser in this phase and it is the best choice of doing pitching maneuver. The pitch-rate is constant, and it is observed that the trajectory is highly sensitive to this value. The maximum value of pitch rate allowable is 2 degree/sec either in clock wise or anti-clockwise. The Pitch-over maneuver completes in 12 to 20 seconds post vertical take-off phase. In addition to the pitching, yaw maneuver can also be performed to efficiently turn the vehicle.

3. **Gravity Turn Phase:** In gravity turn phase, launch vehicle undergoes turning under the influence of gravity and thrust is only required to accelerate the vehicle. In this phase, vehicle axis is aligned with velocity vector profile continuously which ensure the zero angle of attack and thereby reducing the vehicle loads to minimum[1].

    - The importance of ARVS frame comes into picture in this phase. Fig. 6.6 shows the angle of attack and side-slip angles.
    - The gravity turn maneuvers are done specifically to make angle of attack zero which is ensured by the previous constant turn-over phase, through out the gravity turn phase so as to minimize the aerodynamic loads. This requirement is translated to obtain the appropriate pitch (and yaw/roll) rates such that the rate at which gravity turns the vehicle and the rate at which the vehicle turns itself are synchronised to maintain a zero angle of attack.
    - The known requirements are that the angle of attack (α), the side-slip angle (β) and the bank angle (σ) be zero throughout the Gravity Turn phase.
    - This implies that the AB matrix is fully known. Using the AB and GA matrices, the IB matrix can be computed.
    - The IB matrix can also be represented in terms of the roll, yaw and pitch angles. These can now be found by equating the terms of the two IB matrices at every point to get the appropriate steering angles.

    Since the trajectory in the gravity turn is mostly planar, the steering angles can also be obtained by an alternate method. Consider the flight of the launch vehicle as shown

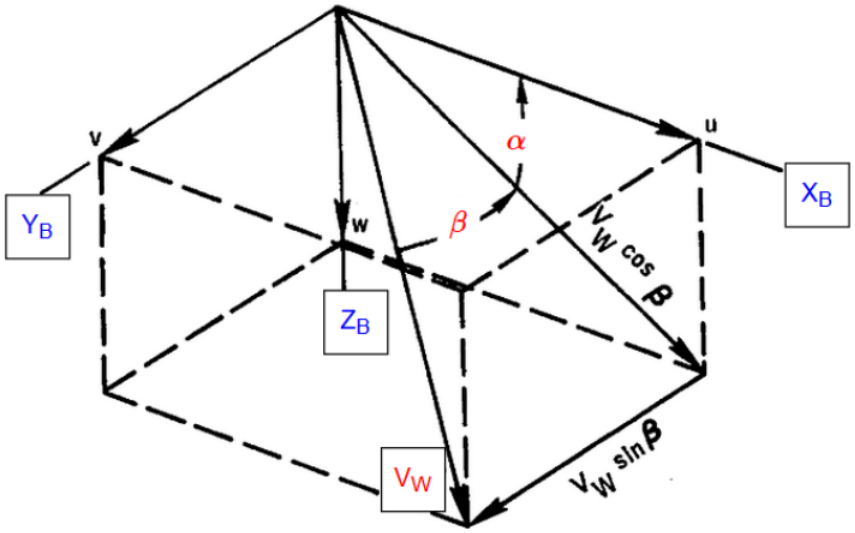

Figure 6.6: Relative Wind Vector and its resolution along the Body Frame axes[6]

in Fig. 6.7. The equations of motion can be written as[1].

$$m\frac{dV}{dt} = T - D - mgcos\theta \tag{6.2}$$

$$mV\frac{d\theta}{dt} = mgsin\theta \tag{6.3}$$

The rate of the angle θ can be written as

$$\dot{\theta} = \frac{d\theta}{dt} = \frac{gsin\theta}{V} \tag{6.4}$$

From the above equation 6.4, the value of θ will be known at each instant (numerical integration). From the value of the θ, the steering angle (pitch angle) can be obtained (θ in this case is not to be confused with pitch angle).

Optimal Trajectory: The endo-atmospheric phase completes after reaching a height where the dynamics pressure is negligible, for a generic launch vehicle it occurs after the first powered stage. After this, the vehicle do not have to follow zero angle of attack condition. The trajectory can be optimized using one of the techniques which is described in the following sections.

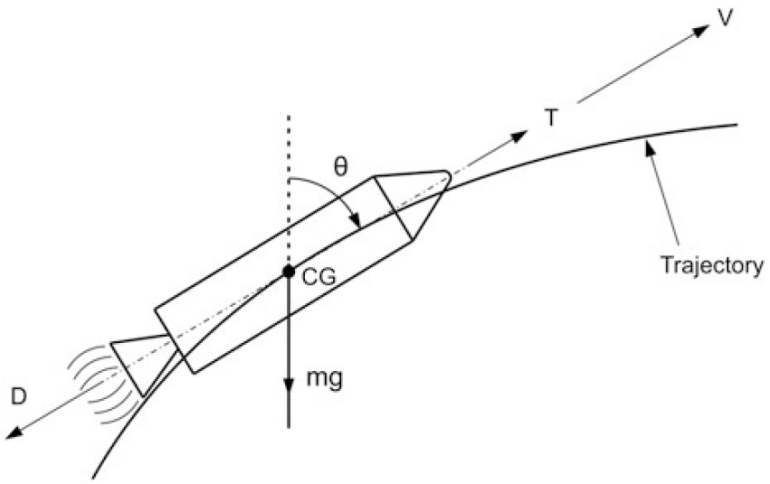

Figure 6.7: Gravity Turn Trajectory[1]

6.6 Steering of Launch Vehicle

The flight path angle (γ) is defined as the angle between the local horizontal and the velocity vector in the LH-LV frame as shown in Fig. 6.8. A low γ is equivalent to a shallow trajectory, and conversely, a high γ is equivalent to a steep trajectory.

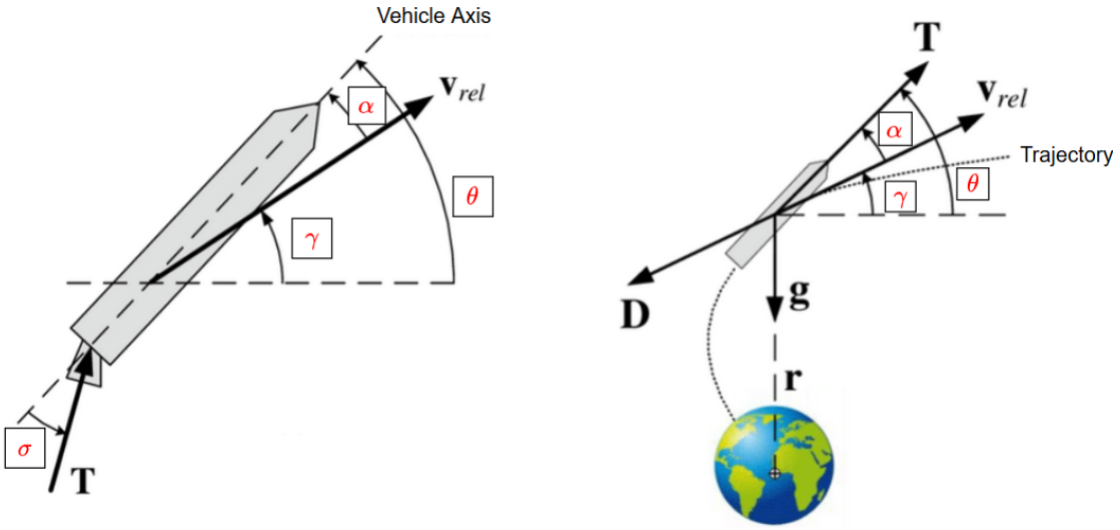

Figure 6.8: LH-LV frame and associated angles[8]

Every launch vehicle starts from steep trajectory and finally attains shallow trajectory. When the flight path angle γ is large the deceleration due to earth gravity is huge ($g \sin \gamma$) this is known as gravity loss, and it is minimized by minimizing the flight path angle (γ). This can be done by rotating the launch vehicle by adding the component of acceleration

normal to the velocity vector. The only way to add the normal component is by the thrust vectoring, then the normal component of acceleration becomes

$$a_{normal} = \frac{T}{m} sin(\alpha) \tag{6.5}$$

, where α is angle between the thrust vector and velocity vector or angle of attack. This component rotates the velocity vector which helps in reducing the flight path angle and loss due to gravity. For circular orbit, the terminal flight path angle must be zero and velocity vector must be horizontal, so the normal component of acceleration must be added continuously to obtain the horizontal velocity condition. This is known as steering of the launch vehicle.

As the vehicle nears the insertion point, the velocity of the launch vehicle becomes significantly high such that the effect of gravity on rotating the vehicle is greatly reduced. So thrust has to continually be employed to not aid the velocity but to only rotate it. This is known as *Steering Loss.*

The normal component of acceleration depends on both $\frac{T}{m}$ and α, so to achieve required acceleration the value of $\frac{T}{m}$ must be higher if α is small and the value of $\frac{T}{m}$ must be smaller if the value of α is large.

A trade-off must be done between the gravity loss and steering loss to obtain optimal trajectory. If we want to minimize the gravity loss, the vehicle has to be steered at the earliest so that the vehicle attains the near horizontal flight (γ is small) but the vehicle need to rotates large angle which maximizes steering loss. If the vehicle is steered gradually, the steering loss will be small but gravity loss is higher, therefore a trade-off must be made between these two.

6.7 Definition of Pitch Angle

From the Fig. 6.8, it can be seen that the pitch angle can be written as sum of flight path angle and angle of attack. At the beginning of the launch, the angle of attack is zero and flight path angle is 90° which implies that the pitch angle should be 90°. But, according to the convention used in POST Formulation Manual, the Launch Frame is located at the geodetic latitude and inertial longitude of the launch vehicle at the beginning of the launch. The x-axis is along the positive radius vector, or along the local vertical. This implies that the x-axes of the Body and Launch frames at the launch are coincident. Subsequently, the y and z axes are also coincident. The inertial pitch angle, used in obtaining the transformation

matrix LB should then be such that initially the LB matrix should be an identity matrix. The definition of the LB matrix is used in all further calculations following this convention, which suggests that the initial pitch angle is 0°.

6.8 Orbital Inclination

The orbit inclination can be defined as the angle between the orbital plane and earth's equatorial plane measured counter-clockwise from the equatorial plane. There exist an interesting relation between the orbital inclination (i), Launch Azimuth (Az_L) and Latitude of the location of launch site (ϕ_L).

$$cos(i) = sin(180 - Az_L) * cos(\phi_L) \tag{6.6}$$

$$= sin(Az_L) * cos(\phi_L) \tag{6.7}$$

The Launch Azimuth is the angle between the local North direction and the initial launch direction measured clockwise. It is due to this relation the minimum orbital inclination possible is equal to the latitude of launch site at launch azimuth $Az_L = 90^{\circ}$ i.e., a perfect eastward launch. From the above equation we can see that for a given launch site, the inclination is directly related to launch azimuth, changing the launch azimuth changes the inclination. Another factor which changes inclination is the out-of-plane or yaw maneuver, in which a lateral rotation of the velocity contributes to changing the orbital inclination. Modification of the Launch Azimuth is preferable to a yaw maneuver, because of a number of reasons, mainly that out-of-plane maneuvers entail significant fuel consumption. However, landmass and safety restrictions often limit the allowable Launch Azimuth, so a yaw maneuver is inevitable in that case.

6.9 Impact Points

Impact points are the points on the earth surface at which the spent stages of the launch vehicle would touch down during the trajectory after jettisoned from the launch vehicle. This point is specified in terms of the latitude-longitude coordinates on the Earth. Impact point considerations are very crucial for the mission and safety constraints indicate that these impact points are far away from the land masses, oil rigs, etc. If any spent stages fall on the land masses, oil rigs, etc the consequences would be catastrophic, because of this reason the launch vehicles are designed with self-destruct option, if the actual trajectory

violates from its predefined path the vehicle will be destroyed before it will hit any other land mass.

6.9.1 Calculation of Impact Point

It is essential that the calculation of the impact point be done in the Earth Centered Rotating Frame or Earth-fixed frame; thus the relative velocity of the stage with respect to the rotational velocity of the Earth needs to be considered. There are two methods for calculating the impact points, they are:

6.9.1.1 Propagation of the State Vector

- It is assumed that there is no atmosphere, wind such that the aerodynamic forces acting on the spent stages are zero. This assumption is made because to avoid the complexity drag and drag coefficient calculations of the spent stage.
- Since no thrust acts on the spent stage once it is jettisoned, the only contribution to the acceleration is that of gravity.
- This acceleration, on being transformed to the ECR frame, is used to calculate the relative velocity vector and subsequently the position vector (using numerical integration).
- Thus the state vector is propagated using the equations of motion till the norm of position vector equals the radius of the Earth.

6.9.1.2 Analytical Computation: Keplerian Impact Point

- Here, it is assumed that there is no atmosphere, wind and no perturbation due to non-spherical earth, third body interactions, etc which results the spent stage after jettison follows a Keplerian elliptical orbit with the earth at one of its foci.
- The initial state vector $(\bar{r}, \bar{v})$ of the spent stage just after jettison is converted into orbital elements for its elliptical orbit.
- The touchdown point will be that point of the resulting elliptical orbit at which the magnitude of the the radius vector is equal to the radius of the Earth.

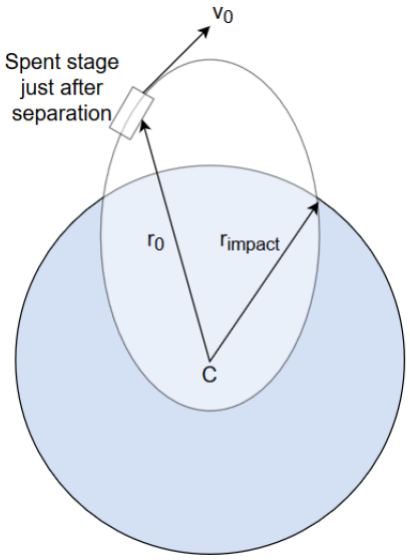

Figure 6.9: Geometry for Impact Point Calculation[2]

- It follows that there will be two such points in the ellipse where the above condition holds; the impact point is the one where the true anomaly is greater than 180°. The initial momentum of the spent stage contributes to its initial rise in altitude.

6.10 Dog-Leg or Yaw Maneuvers

Generally, in the case of safety restrictions, the launch vehicle is launched with a safety launch azimuth angle such that there will be no land masses and at some point after the launch vehicle clears the land mass, it performs plane change or yaw-maneuver which changes the relative velocity Azimuth and rotates the velocity vector till the required inclination is achieved. A yaw maneuver before the vehicle begins the gravity turn phase is beneficial because it is far easier to change the velocity vector when its magnitude is less. But the chances of clearing all the land masses before gravity turn are very rare, therefore the yaw maneuvers have to done in the later stages of the trajectory. The point at which the yaw maneuver can be performed is obtained by optimization techniques or the yaw rates also given as design variables in addition to pitch rates and are optimized to get the desired trajectory.

ISRO's Polar Satellite Launch Vehicle (PSLV) performs the same maneuver to avoid the Sri Lanka land mass constraint shown in Fig. 6.10. It is called a dogleg maneuver because of it resemblance with hind leg of a dog. This maneuver, like the aforementioned Impact Point Constraints, also requires the Impact Point to be calculated. But the difference

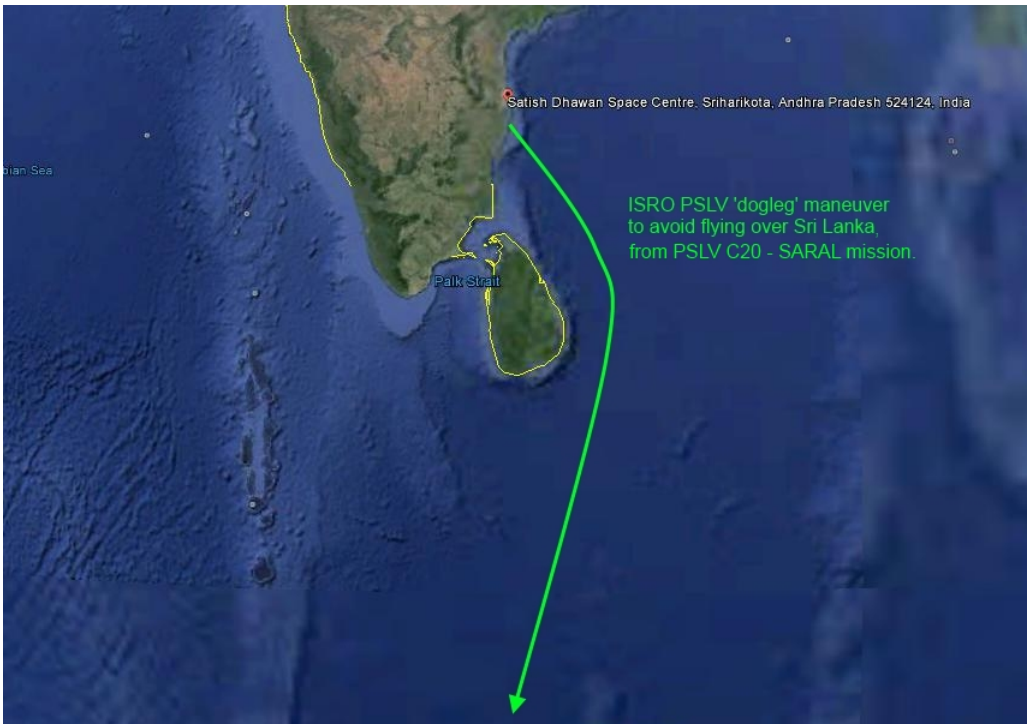

Figure 6.10: Dogleg maneuver during ISRO's PSLV C20 SARAL mission[9]

is that simulation of the former involves the calculation of the impact point throughout the entire trajectory instead of a single specific time point. This is known as the Instantaneous Impact Point Trace (IIP trace). The Instantaneous Impact Point denotes the touch-down point of the vehicle assuming an immediate end of its propelled flight[20]. The feasible range of impact points is specified a priori and is an important consideration for trajectory optimization. *Yoon et. al*[20] detailed the procedure for calculating the IIP and formulating the trajectory optimization problem including IIP constraints.

6.11 Optimization of Ascent Trajectory

The ascent trajectory is optimized using direct method i.e., the trajectory optimization is formulated as a Non-Linear Programming Problem (NLP) subjected to various equality and inequality constraints. For the current work of launch vehicle design, Kulasekharapattinam is considered as launch site, therefore the trajectory will be planar and impact point constraints will not come into picture. Also, the feasibility study has been done for the launch from SHAR and three-dimensional ascent trajectory is optimized separately with constant thrust profile and the results are shown in chapter 8.

6.11.1 Objective Function

The goal of trajectory optimisation is to maximize either the payload or the insertion orbit radius.

1. Maximising the Orbit Altitude:
   The difference between the norm of the final inertial radius vector and the radius of the Earth is defined as the final orbit altitude. The trajectory simulation includes a sizing routine which determines the optimal mass distribution for the launch vehicle, for a pre-specified altitude. However, optimization can help determine a greater altitude still, for the same mass breakup.

2. Maximizing the Final Payload:
   This is the most common objective of trajectory optimization. But because it needs to be provided for the sizing routine a priori, it cannot directly be written as a function of the decision variables. To bypass this issue, the difference in the terminal velocity of the spacecraft for an orbit of a particular altitude and the orbital velocity of that altitude can be added to the payload as equivalent mass, if the former is greater:

   $$\Delta v = v_{terminal} - v_{orbit}$$

   , where the orbital velocity is obtained from the Vis-Viva equation in Eqn. 6.8.

   $$v_{orbit} = \sqrt{\mu_E \left(\frac{2}{r} - \frac{1}{a}\right)} \tag{6.8}$$

6.11.2 Optimisation Variables

The vector **z** contains the following variables that are to be optimised:

1. The pitch rate during the constant turn-over phase.

2. The coasting time after stage-2

3. The remainder of the trajectory after the dense atmosphere is discretized into ten segments each for stage-2 and stage-3 and each segment is assigned a constant pitch-rate. These are considered as design variables. Discretization of the time duration is possible because the time until exhaustion is known and fixed.

4. In addition to the pitch-rates during the second and third stages, the yaw rates can also be taken as optimization variables when out-of-plane maneuvers are required, like in Instantaneous Impact Point trace constraints.

Apart from the steering rates, additional optimization variables can also be taken as:

1. The time duration for the constant turn-over phase, which decides when the gravity turn phase begins.

2. The amount of tilt during the gravity turn.

3. Throttling durations and values, since the thrust profile and propulsion system interact very closely with the trajectory and influence it.

These are all important parameters as they decide the trajectory that the launch vehicle follows and hence, they decide how much losses of which kind it faces and how much fuel is spent overcoming the losses. If the vehicle turns too little or too late in its flight, excess fuel will be expended in overcoming gravity than overcoming drag. On the contrary, if it turns too much or too early, it will travel a longer distance through the atmosphere, requiring more fuel to regain the speed lost to overcome drag.

6.11.3 Constraints

6.11.3.1 Terminal Constraints

1. Terminal Velocity:
   The terminal velocity obtained should be equal to the orbital velocity for the altitude obtained as in Eqn. 6.8. Since the final orbit altitude is variable in the case of maximization of altitude, the orbital velocity also varies accordingly. Ideally, the terminal velocity should be exactly the same as the orbital velocity. However, a terminal velocity in its vicinity, i.e. within an interval of 10 m/s around it, can also be accepted without affecting the mission requirements greatly. Since inequality (range) constraints are easier to be achieved as compared to equality constraints, the velocity constraint can also be taken as an inequality constraint as shown in Eqn. 1.

$$v_{orbit} - 10 \leq v_t \leq v_{orbit} + 10$$

2. Final Flight Path Angle (γ):
   The flight path angle for a circular terminal orbit must be exactly 0, but we allow an

interval of $\pm 2^\circ$ for uncertainty:

$$-2^\circ \leq \gamma \leq 2^\circ$$

3. Orbital Inclination:
   Out-of-plane maneuvers influence the final inclination achieved, hence the yaw rate is the most important optimization variable affecting the inclination. This can be given as an equality constraint, with 90 degrees for polar orbits and 0 degrees for equatorial orbits and so on.

6.11.3.2 Path Constraint

The launch vehicle can only withstand dynamic pressure below a specified limit, so the maximum dynamic pressure encountered during flight becomes an important constraint having a huge impact on the structural characteristics of the vehicle. Higher the dynamic pressure, greater the aerodynamic loads on the vehicle. It peaks in the first stage during the gravity turn phase and then falls to zero by the end of the dense atmospheric region. Minimum dynamic pressure possible is obtained for a straight vertically upward trajectory, but since the actual trajectory curves after the first few metres, the objective is to minimize the dynamic pressure peak. The maximum dynamic pressure aloowable is 55kPa.

6.12 Problem Statement

$$Maximize \qquad r\ (or)\ \triangle V \tag{6.9}$$

$$With\ \ Respect\ to \qquad z_{trajectory} = \{\dot{\theta}_1^{(2)}, ..., \dot{\theta}_{10}^{(2)}, \dot{\theta}_1^{(3)}, ..., \dot{\theta}_{10}^{(3)}, \dot{\theta}_P, t_{coast,2}\} \tag{6.10}$$

$$Subject\ to \tag{6.11}$$

$$1) Path\ Constraints \tag{6.12}$$

$$c_1 :\ Q_{max} \leq 55kPa \tag{6.13}$$

$$2) Terminal\ Constraints \tag{6.14}$$

$$c_2 :\ |V_{terminal} - V_{orbital}| \leq 10m/s \tag{6.15}$$

$$c_3 :\ |\gamma_{terminal} - \gamma_{orbit}| \leq 2^\circ \tag{6.16}$$

$$c_4 :\ |r_{terminal} - r_{orbital}| \leq 10km \tag{6.17}$$

$$c_5 :\ i = 90^\circ \tag{6.18}$$

$$\tag{6.19}$$

6.13 Implementation

1. The upper and lower bounds of the pitch rates for both stage-2 and stage-3 are taken as [-2,+2] deg/sec, and the upper and lower bounds for coast time are taken as [8, 15].

2. The constraint c_4 can be relaxed when optimizing for maximum orbit radius and the constraint c_2 can be relaxed when optimizing for maximum terminal velocity.

Chapter 7

Optimization

Launch vehicle design is a complex process in which the search of the best performance at the less cost is essential. This process takes account of many disciplines such as Sizing, Propulsion, Aerodynamics, Trajectory, cost, etc, which have to be well handled in order to ensure the optimality of the launch vehicle designed. These disciplines, which often have conflicting objectives, and require adapted design tools which allow to integrate the constraints inherent to each discipline and to facilitate the compromise search[24]. This chapter provides an account of the launch vehicle design optimization problem formulation, the methods of optimization used in the current work, and finally, the implementation of these techniques to obtain an optimal design.

In general, a classical optimization problem involves maximising or minimising a function (f) defined over a region. The feasible set within the region is specified by constraints. The solution of the optimization problem is specified by the values of the decision variables (x). Thus an optimization problem can be described as below, with lb and ub denoting the domain for x.

$$
\begin{aligned}
&Maximize/Minimize && f(x)\\
&With\ respect\ to && x\\
&Where && lb < x < ub\\
&Subject\ to &&\\
&1.\ Equality\ Constraints &&\\
& && A_{eq}x = b_{eq}\\
& && c_{eq}(x) = 0\\
&2.\ In-Equality\ Constraints &&\\
& && Ax \leq b\\
& && c(x) \leq 0
\end{aligned}
$$

7.1 Sequential or Fixed Point Iteration Method

Sequential or Fixed Point Iteration method is the traditional way to solve the launch vehicle design problem. Each discipline solves its own optimization problem and aims to be consistent with the other ones. This method is built on the designer expertise and does not necessarily ensure the global optimality[24]. The design variables in the preceding discipline gets frozen and only the design variables in the present discipline are allowed to vary, as a result the final design is likely to be sub-optimal. FPI method is applied to optimize three disciplines Launch vehicle Sizing, Propulsion, Aerodynamics and Ascent Trajectory and their individual problem statements and methodologies are presented in chapters3, 4, 5 and 6.

7.2 Multi-Disciplinary Design Optimization (MDO)

Multi-Disciplinary Design Optimization is a field of engineering that focuses on the use of numerical optimization for the design of systems that involve a number of disciplines or subsystems. The main motivation for using MDO is that the performance of a multi-disciplinary system is driven not only by the performance of the individual disciplines but also by their interactions. Considering these interactions in an optimization problem generally requires a sound mathematical formulation. By solving the MDO problem early

in the design process and taking advantage of advanced computational analysis tools, designers can simultaneously improve the design and reduce the time and cost of the design cycle[25].

In this section, the mathematical formulation of a general MDO is presented and then MDO method used in the Launch vehicle design is described and the problem statement is presented at last. The terminology and notation presented here are inspired from[10].

7.2.1 Mathematical Formulation of a General MDO Problem

7.2.1.1 Terminology and Mathematical Notation

Before introducing directly the mathematical formulation, it is better to know the terminology and notation for clear understanding and comparison of various MDO problem formulations and architectures.

7.2.1.1.1 Types of Variables

1. **Design Variables (z):** The design variable is a quantity in the MDO problem which is under the explicit control under the optimizer, these variables evolve all along the optimization process in order to find the optimal design. The design variables may be local to one particular subsystem or they may be shared by several subsystems, the total design variables are denoted by $z = \{z_{sh}, \bar{z}_k\}$, where, the subscript *sh* represents the shared design variable between different subsystems and $\bar{z}_k$ denotes the design variables which are local or specific to the subsystem 'k'.

2. **Coupling Variables (y):** The coupling variables are used to link different subsystems and to evaluate the consistency of the design with regard to the coupling.

3. **State Variables (x):** State variables or disciplinary variables are the variables which are computed in a discipline analysis(by solving discipline equations, etc). State variables may or may not be controlled by the optimization, depending on the formulation employed. Unlike z, the state variables are not independent degrees of freedom but depend on the design variables z, the coupling variables y and the state or disciplinary equations. In some cases the state variables are written explicitly as a function of deign and coupling variables, but in most of the cases, state variables are implicit functions of design and coupling variables.

7.2.1.1.2 Disciplinary Analysis:

A discipline analysis is a simulation that models the behavior of one aspect of a multi-disciplinary system by taking design and coupling variables as input. Running a discipline analysis consists of solving a system of equations (such as the Navier–Stokes equations in fluid mechanics, the static equilibrium equations in structural mechanics, or the equations of motion in a control simulation) to compute a set of state variables. Generally the disciplinary equations are in the residual form $R_i(x_i, y_i, z_i) = 0$

7.2.1.1.3 Disciplinary Evaluation:

Disciplinary Evaluation takes the design and coupling variables as input and evaluates the residual $R_i(x_i, y_i, z_i)$. The difference between the Disciplinary Analysis and Evaluation is shown in Fig. 7.1.

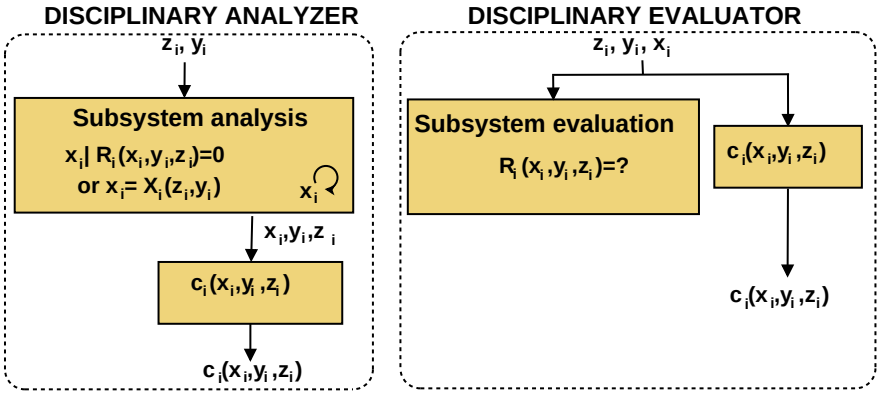

Figure 7.1: Disciplinary Analyser and Evaluator[10]

7.2.1.1.4 Coupling:

From the variables y_i and z_i coming into the subsystem i and the state variables x_i, the coupling variables which come out of the subsystem can be calculated with the coupling functions $c_{ji}(x_i, y_i, z_i)$. The double indexation $c_{ji}(x_i, y_i, z_i)$ denotes that these coupling variables are transmitted from j^{th} subsystem to i^{th} subsystem. The coupling is said to be consistent when the set of coupling variables y_i is equal to the set returned by the different coupling functions.

7.2.1.1.5 Multi-Disciplinary Analysis (MDA):

Multi-Disciplinary Analysis (MDA) is a process in which the disciplinary analysis is performed for every subsystem such that the couplings are consistent. In other words MDA

consists in finding, for all the subsystems, the variables x_i and y_i such that the disciplinary equations or Residual equations are satisfied and the couplings are consistent, i.e. in mathematical terms,

$$\begin{cases} y_i = \{c_{ji}(x_i, y_i, z_i)\}_j & \forall i \in \{1, ..., n\}, \forall j \neq i \\ R_i(x_i, y_i, z_i) = 0 & \forall i \in \{1, ..., n\} \end{cases}$$

Different numerical schemes are available for Multi-Disciplinary Analysis.

7.2.1.1.6 Feasibility Concepts:

1. **Individual Disciplinary Feasibility:** A process is qualified as "individual disciplinary feasible" if at each iteration, the state equations of the different disciplines are satisfied. In other words the disciplinary or residual equations are solved to get the state variables at each iteration but the consistency of the couplings is not guaranteed.

2. **Multi-Disciplinary Feasibility:** A process is qualified as "multidisciplinary feasible" if at each iteration, the state and coupling variables (respectively x_i and y_i) can be found such that the "individual disciplinary feasibility" is realized and the couplings are consistent. It is nothing but individual disciplinary feasibility but with consistent couplings.

Individual Disciplinary Feasible (IDF) and Multi-Disciplinary Feasible (MDF) MDO methods defined based on these concepts.

7.2.1.2 Mathematical Problem Formulation

The general formulation of a MDO problem can be written as follows

$$Maximize \qquad f(x, y, z) \tag{7.1}$$

$$With\ Respect\ to\ z \in Z \tag{7.2}$$

$$Subject\ to \qquad g(x, y, z) \leq 0 \tag{7.3}$$

$$h(x, y, z) = 0 \tag{7.4}$$

$$\forall i \in \{1, ..., n\}, \forall j \neq i y_i = \{c_{ji}(x_i, y_i, z_i)\}_j \tag{7.5}$$

$$\forall i \in \{1, ..., n\} R_i(x_i, y_i, z_i) = 0 \tag{7.6}$$

7.2.2 MDO Architectures and Diagrams

The basic MDO formulation is shown in the above section, there exist many formulations based on different approaches for a given problem, known as *Architectures*. The MDO architectures defines both how the different models are coupled and how the overall optimization problem is solved. The MDO architectures are divided into two types, monolithic and distributed architectures[25]. In monolithic architectures a single and unified optimization problem is solved, it is also known as Single Level Optimization. In distributed architectures, there may be more than one optimization problem for system.

In the current work Extended Design Structure Matrix or XDSM is used for representing the MDO architecture. As the name suggests the XDSM was based on the Design Structure Matrix (DSM), a common diagram in systems engineering that is used to visualize the interconnections among components of a complex system but the meaning of the connections are not explained, whereas XDSM, simultaneously communicates data dependency and process flow between computational components of MDO architectures on a single diagram. The XDSM for the current problem is elaborated in section 6.2.6.

7.2.3 Multi-Disciplinary Feasible Method (MDF)

With all the terminology and notations discussed above, the Multi-Disciplinary Feasible (MDF) method can be defined as a single level or monolithic architecture MDO method in which the solution is *Multi-Disciplinary Feasible* at each iteration i.e, the couplings are always consistent and the residual equations are always zero at each iteration, so the two constraints from the general MDO problem, Eqns. (6.5) and (6.6) can be removed. The interesting fact here is that one can represent state and coupling variables as functions of design variables. The MDF method is simple possible MDO method because there are only The architecture consists of an optimizer to the classical optimization problem but Multi-Disciplinary Analyser is added to the optimizer which is performed at each iteration of the optimization process. The problem statement for the MDF method is described below.

$$Maximize \qquad f(x, y, z) \tag{7.7}$$

$$With\ Respect\ to\ z \in Z \tag{7.8}$$

$$Subject\ to \qquad g(x, y, z) \leq 0 \tag{7.9}$$

$$h(x, y, z) = 0 \tag{7.10}$$

MDF is the most usual MDO method, it is also called “Nested Analysis and Design” (NAND), “Single NAND-NAND” (SNN) and “All-in-One”. Notable monolithic architectures other than MDF are Individual Disciplinary Feasible (IDF) Method and All At Once (AAO) method. As discussed earlier, IDF method show Individual Disciplinary Feasibility in which only the disciplinary or residual equations are satisfied and coupling variables are treated as design variables, the consistency of coupling is given as additional constraint. In AAO method Disciplinary Evaluation instead of disciplinary analysis is performed and both state, coupling variables are treated as design variables, here consistency of coupling and disciplinary equations are given as additional constraints. Multi-Disciplinary Feasibility is satisfied only at convergence for both the methods.

The main advantage of MDF over other architectures is that, MDF is as small as it can be for a monolithic architecture, since only the design variables, objective function, and design constraints are under the direct control of the optimizer, whereas in IDF and AAO there will be additional design variables(coupling and state) and this increases the dimensionality of the problem. If the MDF method is terminated midway, the intermediate solution is Multi-Disciplinary Feasible which is not guaranteed for other architectures.

7.2.4 Launch Vehicle Design using MDF method

The specificity of the Launch Vehicle Design problem as a MDO problem is that the presence of a heavily constrained trajectory optimization with stage separations. The trajectory is optimized by using an optimal control law calculation subject to equality constraints (mission specifications). Therefore it includes the optimization of not only design variables but also a control law. This task is responsible for the performance estimation and presents a relatively important difficulty in the optimization process. The Launch Vehicle Design problem is generally decomposed into the different physical disciplines. In the current work, four disciplines are considered for optimization, they are, Launch Vehicle Sizing/Configuration, Propulsion, Aerodynamics and Trajectory.

The disciplines are discussed clearly in the previous chapters. In most of the cases the disciplines in launch vehicles can be arranged in an order such that a discipline can only have inputs from the preceding discipline and gives only output to the succeeding so that there will be no interconnecting inputs(feedback), In other words the disciplines are arranged in a sequential order such that one’s output will be input for others. MDF method is most natural kind of method to solve these type of problems[24]. In the literature nearly 80% of authors used MDF method with different optimizing solvers[10].

7.2.4.0.1 Trajectory Handling: As discussed earlier, trajectory is the one which is making launch vehicle design problem complex and difficult. From the definition, the variables in the trajectory(pitch, yaw rates in the current problem) are under the category of state variables because they only appear in the trajectory and also and optimal variables are obtained by solving equations of motion($R(x, y, z) = 0$). But in the case of launch vehicle design, the trajectory variables plays a major role in the final design and attaining mission objective. In the current work the trajectory variables are also considered as design variables. The overall problem statement is shown below

7.2.5 Launch Vehicle Design Problem Statement

$$Maximize \qquad r\ (or)\ \triangle V \tag{7.11}$$

$$With\ \ Respect\ to \qquad \{z_{propulsion}, z_{trajectory}\} \tag{7.12}$$

$$z_{propulsion} = \{N, \epsilon, w, f, R_i, \frac{L}{D}\} \tag{7.13}$$

$$z_{trajectory} = \{\dot{\theta}_1^{(2)}, ..., \dot{\theta}_{10}^{(2)}, \dot{\theta}_1^{(3)}, ..., \dot{\theta}_{10}^{(3)}, \dot{\theta}_P, t_{coast,2}\} \tag{7.14}$$

$$Subject\ to \tag{7.15}$$

$$1) Path\ Constraints \tag{7.16}$$

$$c_1 :\ Q \leq 55kPa \tag{7.17}$$

$$2) Terminal\ Constraints \tag{7.18}$$

$$c_2 :\ |V_{terminal} - V_{orbital}| \leq 10m/s \tag{7.19}$$

$$c_3 :\ |\gamma_{terminal} - \gamma_{orbit}| \leq 2^\circ \tag{7.20}$$

$$c_4 :\ |r_{terminal} - r_{orbital}| \leq 10km \tag{7.21}$$

$$c_5 :\ i = 90^\circ \tag{7.22}$$

$$3) Propulsion\ Constraints \tag{7.23}$$

$$c_6 :\ P_c \leq 95bar \tag{7.24}$$

$$c_7 :\ t_b \leq 150sec \tag{7.25}$$

Here, the the constraint c_2 is relaxed when maximizing the total $\triangle V$ and c_3 is released when maximizing the r. The sequence of the disciplines, data flow, etc are shown in XDSM below.

7.2.6 Extended Design Structure Matrix for Launch Vehicle Design

The Extended Design Structure Matrix (XDSM) for the current problem is shown in Fig. 7.2. In XDSM the disciplines or components are laid along the diagonal with a special component, known as a driver, that controls the iteration and is represented at top by a rounded rectangle(optimizer in this case). The data flow between the disciplines is represented by thick gray lines. The components take data inputs from the vertical direction and output data in the horizontal direction. Thus, the connections above the diagonal flow from left to right and top to bottom, and the connections below the diagonal flow from right to left and bottom to top.

The off-diagonal nodes in the shape of parallelograms are used to label the data. Us-

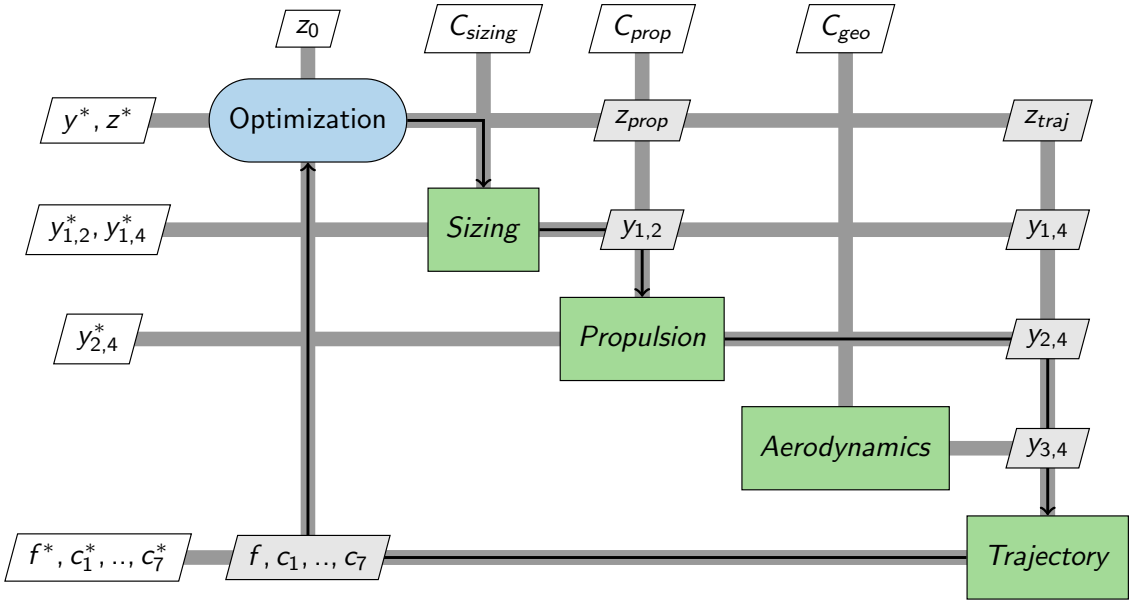

Figure 7.2: XDSM for Launch Vehicle Design

ing this convention, it is easy to identify the inputs of a given component by scanning the columns above and below the component, while the outputs can be identified by scanning the row. External inputs and outputs are placed on the outer edges of the diagram, in the top row and leftmost column. Here in the current work, external inputs are the initial guesses (z_0) of design variables and system constants such as diameter, structural ratios, propellant properties and other constants ($C_{sizing},\ C_{prop},\ C_{geo}$) and the external outputs are optimized variables($x^*,\ y^*$), optimized objective function (f^*) and optimized constraint values ($c^*_1, .., c^*_7$) . Each discipline analysis computes its own set of coupling variables that

is passed to other discipline analyses. The thin black lines show the process flow. The direction of these lines follows the convention for the data-flow lines. In the current work, since the disciplines are arranged such that there are no feedback(only sequential) among them, there is no need of additional solver for Multi-Disciplinary Analysis. Generally, Guass-Siedel Method with Newton Rhapson solver is used for MDA. The above XDSM diagram is drawn using the convention and rules by [25], [**?**].

7.3 Hybrid Optimization Solver

An Hybrid Optimization solver is employed for the job of optimizer in both MDO and FPI methods. An Hybrid Optimization solver is a combination of both heuristic and gradient based optimization methods using the benefits of both the methods. Heuristic methods are very good in global exploration of the search space but weak in local exploitation whereas gradient based methods have the capability of local exploitation and convergence but requires good initial guess, the initial guess is given by heuristic methods and local exploitation is done by gradient based methods, thus using the benefits of two methods. In the current work *Particle Swarm Optimization (PSO)* and *Sequential Quadratic Programming (SQP)* are combined to form Hybrid Optimization Solver. It is also known as *Particle Swarm Optimization Guided Sequential Quadratic Programming*(PSOGSQP). The two optimization methods are discussed briefly in this section.

7.3.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a heuristic population-based global optimization method which is based on simulation of social behavior of the animal colonies or societies(Swarm Intelligence). Unlike Evolutionary Algorithms like Genetic Algorithm, Differential Evolution, etc, which mimics evolutionary processes, PSO relies on the exchange of information between individuals, called *particles*, of the population, called *swarm* or population itself. Each particle in the swarm has a position and velocity associated with it. Particles change the position and velocity to seek food, avoid predators, etc, and each particle memorizes the best location and position identified by it. Particles communicate the information regarding the best location explored by them and each particle adjusts its trajectory towards its own previous best position, and towards the best previous position attained by any member of its swarm.

Mathematically, let us represent PSO using the case of minimization of objective or

fitness function f, in a D-dimensional search space, the position of the i^{th} particle in the swarm is represented by a D-dimensional vector $X_i = (x_{i1}, x_{i2}, x_{i3}, ..., x_{iD})$ and velocity be $V_i = (v_{i1}, v_{i2}, v_{i3}, ..., v_{iD})$. Let $P_{i,best} = (p_{i1}, p_{i2}, p_{i3}, ..., p_{iD})$ be the personal best position of the particle i, for which the objective function is minimum, and $g_{best} = (g_{i1}, g_{i2}, g_{i3}, ..., g_{iD})$ be the global best position of the swarm i.e., the position with best function value of all the particles. For an iteration $(t + 1)$ the update rule for the velocity and position of the particle i is given by,

$$V_i^{t+1} = \chi \left[w V_i^t + c_1 r_1 (P_{i,best}^t - X_i) + c_2 r_2 (g_{best} - X_i) \right] \tag{7.26}$$

$$X_i^{t+1} = X_i^t + V_i^{t+1} \tag{7.27}$$

where $i = 1, 2, 3, ..., N$ and N is the size of the swarm or population, χ is a constriction factor which is used to control and constrict velocities, w is the inertia weight, c_1 and c_2 are two positive constants, called the cognitive and social parameter respectively, r_1 and r_2 are random numbers uniformly distributed within the range [0, 1]. The interesting fact here is that the position X_i of every particle gets updated irrespective of the fitness value in each iteration but the greedy selection is performed to update $P_{i,best}$ and g_{best}. The detailed algorithm is shown in Algorithm 7.1.

The Input parameters (χ, c_1, c_2, w) play a major role in the performance of Particle Swarm Optimization. In many literature the value of χ is taken as 0.73 and the cognitive and social parameters are taken as 2. PSO is highly sensitive to the inertia parameter 'w', the inertia weight is employed to control the impact of the previous history of velocities on the current velocity. Thus the parameter w regulates the trade-off between the global (wide ranging) and the local (nearby) exploration abilities of the swarm. Generally, the value of w lies in the range [1.2,0.1], the value of w need not be constant and recently many researchers implemented with varying w, which enhanced the performance of PSO. In the current work, linearly varying inertia parameter is used,

$$w = 1.2 - \frac{1.2 - 0.1}{T} t \tag{7.28}$$

where T is the total number of iterations and t is the t^{th} iteration. The value of w at initial iterations is larger which helps in Global-Exploration at the initial iterations and the value decreases with increase in number of iterations which helps in local exploitation at final iterations.

Algorithm 7.1: Particle Swarm Optimization

Input: Fitness Function, lb, ub, T, N_P, χ, w, c_1 and c_2.
Initialize a Random Population (P) and Velocity (V).
Evaluate the fitness(f) of P.
Assign p_{best} as P and f_{pbest} as f.
Identify the solution with best fitness and assign it as g_{best} and fitness function as f_{gbest}.
for $t = 1$ to T **do**
  for $i = 1$ to N_p **do**
    Determine the new velocity (V_i) of the i_{th} particle.
    Determine the new position (X_i) of the i_{th} particle.
    Bound X_i.
    Evaluate the fitness (f_i) of the i_{th} particle.
    Update the population by including (X_i) and f_i.
    Update $p_{i,best}$ and f_{pbest}
    Update g_{best} and f_{gbest}
  end for
end for
Output: g_{best}

7.3.1.1 Constraint Handling in PSO

The PSO algorithm has proven to be very efficient for solving real valued global unconstrained optimization problems, but it is incapable of handling constraints. Generally, the Constrained Optimization (CO) problems are transformed into Unconstrained Optimization problems using *Penalty Function Method*.

Penalty Function methods reduces the constrained NLP into a series of unconstrained optimization problems. The search space in CO consists of two kinds of points, Feasible points and infeasible points, based on which penalty functions are classified into two types *Exterior Penalty Function Method* and *Interior Penalty Function Method*. In Exterior method, initially unconstrained minimum lies in infeasible region and converges to global minimum in feasible region and the converse is for Interior method. Generally, exterior method is widely used in cases involving heuristic optimization methods because for high dimensionality problem find the initial members of the population in the feasible region is

not possible.

There are two types of exterior penalty function methods, *Stationary Exterior Penalty Function Method* and *Non-Stationary Exterior Penalty Function Method*. Stationary penalty functions use penalty values throughout the minimization while in non-stationary penalty functions, the penalty values are dynamically modified. In literature, results obtained using non-stationary penalty functions are almost always superior to those obtained through stationary functions. In the current work, non-stationary exterior penalty function is chosen. For a given minimization problem

$$\begin{aligned} &Minimize && f(x) \\ &Subject\ to, && g_i(x) \leq 0,\ i = 1, ...m \end{aligned}$$

The Non-Stationary Penalty Function can be written as[21]

$$F(x) = f(x) + h(t) \sum_{i=1}^{m} \theta(q_i(x)) q_i(x)^{\gamma(q_i(x))}$$

Where t is the t^{th} iteration and,

$$q_i(x) = max\{0,\ g_i(x)\},\ h(t) = t\sqrt{t}$$

$$\theta(q_i(x)) = \begin{cases} 10 & if\ q_i(x) \leq 0.001 \\ 20 & if\ q_i(x) \leq 0.1 \\ 100 & if\ q_i(x) \leq 1 \\ 300 & otherwise \end{cases}$$

$$\gamma(q_i(x)) = \begin{cases} 1 & if\ q_i(x) < 1 \\ 2 & otherwise \end{cases}$$

Where 't' is the iteration number, Here the equality constraints ($h_i(x) = 0$) are represented as inequality constraints by $h_i(x) \leq 0$ and $-h_i(x) \leq 0$.

7.3.2 Sequential Quadratic Programming (SQP)

Sequential Quadratic Programming is one of the most robust methods to solve non-linear constrained problems. The underlying idea of SQP is to model using non-linear programming the approximate solution by a quadratic programming sub problem, and then to construct a better approximate to the solution using the sub problem. This process is iterated to form a sequence of approximates until convergence. This method can be considered an extension of the Newton and quasi-Newton methods. Thus SQP shares the characteristic of rapid convergence with an initial guess close to the solution with these methods. One of the major advantages of using SQP is that neither the initial guess nor any of the subsequent iterates are required to satisfy all the constraints of the optimization problem. In the presence of equality constraints only, the solution of SQP (like all quadratic methods) reduced to the solution of a system of linear equations. In the current work, fmincon in MATLAB is used for implementing Sequential Quadratic Programming[2].

Chapter 8

Results and Discussions

This chapter consolidates all the results that are obtained by optimizing the problem statements formulated in the previous chapters.

8.1 Sizing Optimization

In the current work the structural ratios (ϵ_k), Thrust to weight ratios are taken as fixed constants, the propellant composition, nozzle configuration are fixed, therefore the specific impulse obtained will also be known. These are fixed constants and will not be changed during any optimization process, the optimized stage masses obtained from these values are also fixed. The known parameters or fixed constants are shown in Table. 8.1. The optimized stage masses are obtained by optimizing the problem formulated in chapter 3 and are shown in Table. 8.2.

8.1.1 Known Parameters

Table 8.1: Known Parameters

Parameter	Value		
	Stage-1	Stage-2	Stage-3
Structural Ratios (ϵ_k)	0.12	0.12	0.15
Vacuum Specific Impulse (I_{sp})	284.7784	284.7784	310
Thrust to Weight Ratios (T/W)	1.4	1.4	0.6
Total Payload	500kg		
Required Orbit	500km Circular orbit		
Total ΔV for the mission	10387.92 m/s		

8.1.2 Optimized values

Table 8.2: Optimized Values

Parameter	Value		
	Stage-1	Stage-2	Stage-3
Total initial mass(m_i)	67388.2809kg	12861.2991kg	2454.6258kg
Propellant mass(m_p)	47983.7440kg	9157.8725kg	1661.4319kg
Structural mass(m_s)	6543.2378kg	1248.8007kg	293.1939 kg

8.2 Aerodynamic Analysis

8.2.1 Post-Processing

The flow field can be visualised and the pressure and velocity variations can be plotted as contours in 2-D, as shown in Fig. 8.1 and Fig. 8.2 respectively. These figures have been plotted for a mach number of 1.2, and the shock formation and expansion fans are visible at the payload fairing and nozzle respectively.

The drag coefficient is computed to be 0.7822 for Mach 1.2 as is observed to have stabilised in Fig. 5.7.

8.2.2 Drag Coefficient vs Mach Number Plot

The same procedure for simulating flow over the launch vehicle was repeated for various mach numbers from 0.2 to 8 and the plot of Fig. 8.3 was obtained.

The drag coefficient shoots up sharply as mach number approaches 1.3 from the subsonic regime and it slowly falls as mach number is increased further.

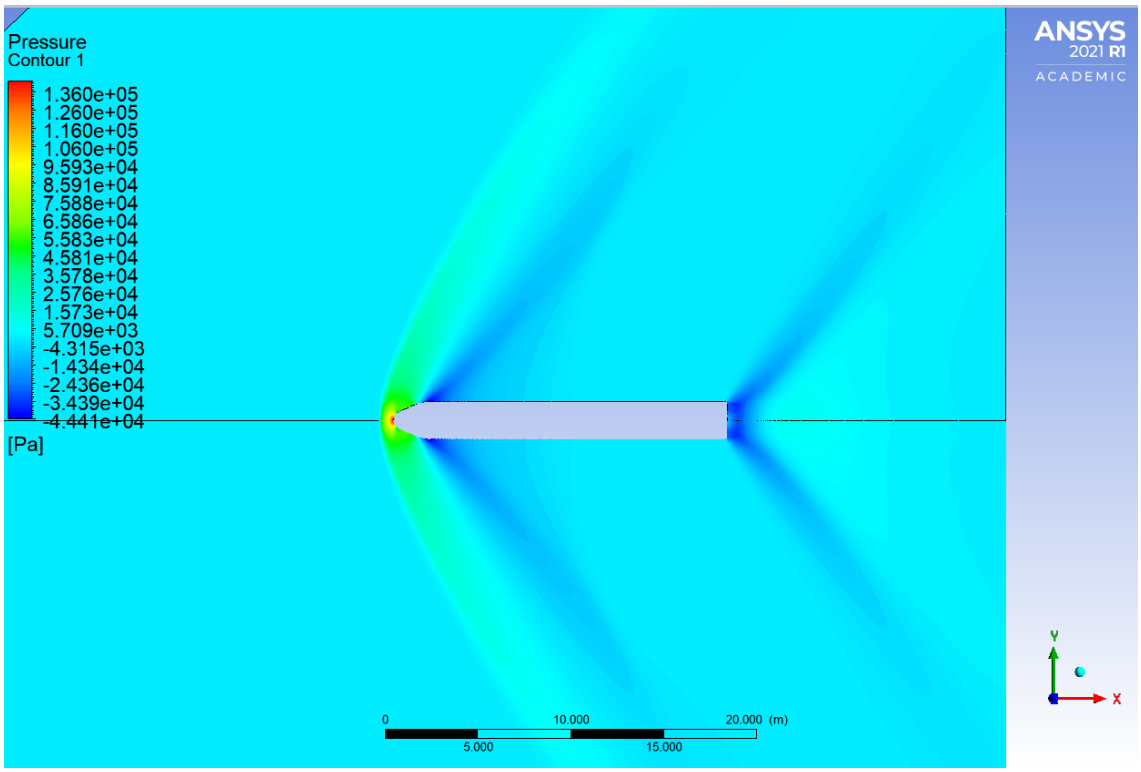

Figure 8.1: Pressure Contours for Flow over Launch Vehicle

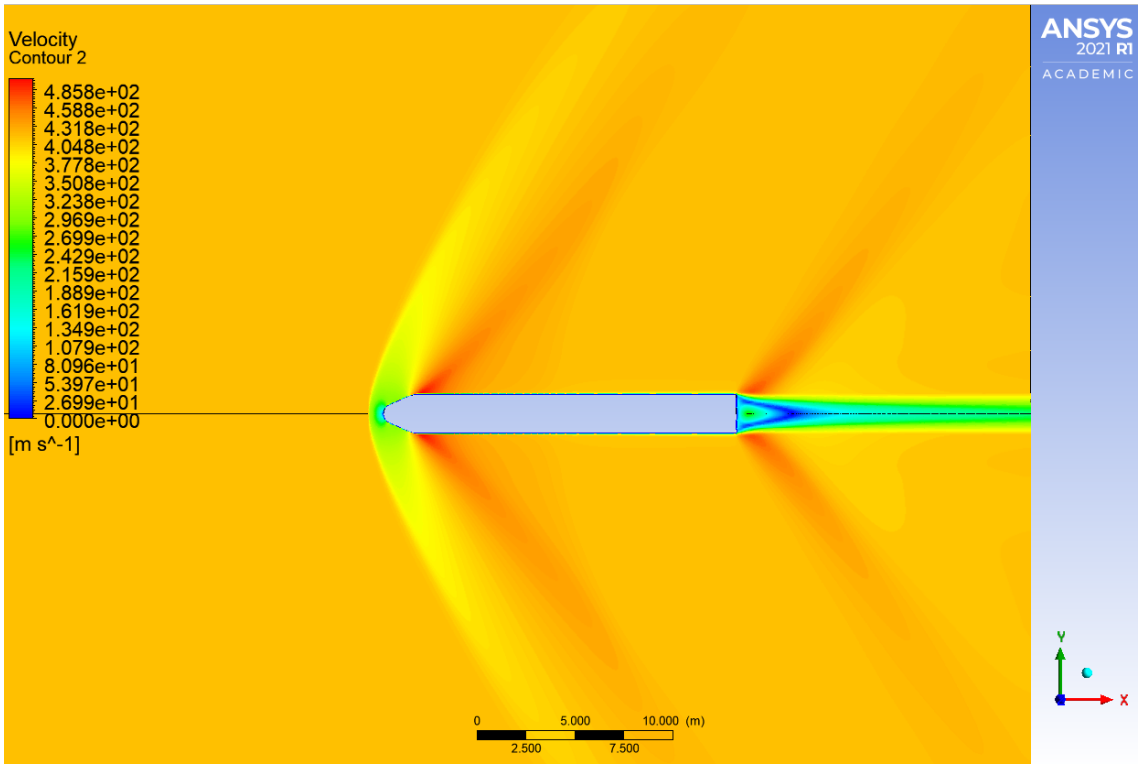

Figure 8.2: Velocity Contours for Flow over Launch Vehicle

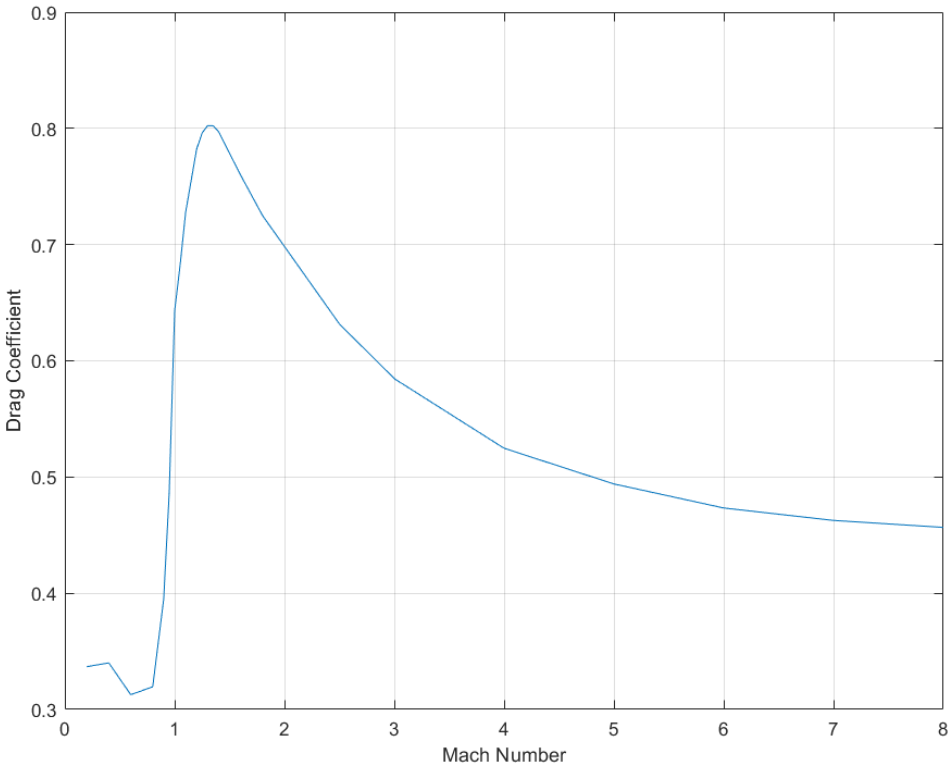

Figure 8.3: Drag Coefficient vs Mach Number

8.3 Sequential or Fixed Point Iteration Method(FPI)

The first set of results are those obtained by FPI i.e., optimizing each discipline separately with their individual problem statements and are finally combined to obtain total vehicle design. The results for each discipline and total vehicle are presented below.

8.3.1 Solid Rocket Motor Optimization

The SRM optimization is done by solving the problem statement which is described in Chapter 4. The NLP problem is solved using the Hybrid Optimization Solver. From the mass of stage-1 propellant obtained from sizing optimization, the length of first stage SRM for the diameter of 2 m is calculated as 8.6292 m. But the obtained thrust profile is not matching with the trajectory. So, $(\frac{L}{D})$ ratio is also considered as additional design variable and its search space is considered from 4.3146 to 5. The value 5 is selected as upper bound using trail and error method.

Table 8.3: Optimized Grain Geometry Parameters

N	ϵ	w	f	R_i	$\frac{L}{D}$
3	0.2500	0.5800m	0.01m	0.01m	5.0000

The additional length of the motor will add extra mass to the Gross Lift off Mass of launch vehicle. The final gross lift-off mass is found to be

$$M_{GLOM} = 75010.7260kg \tag{8.1}$$

which is approximately 75 tonnes.

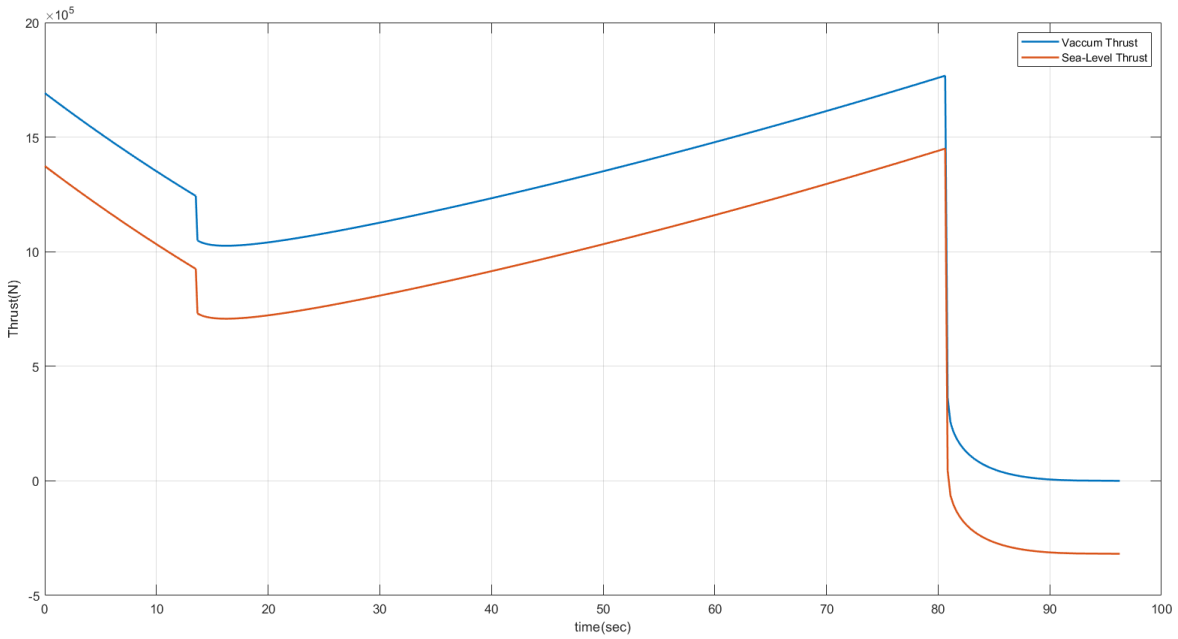

Figure 8.4: Vacuum and Sea-Level Thrust time profile for the optimized grain geometry

The actual thrust-time profile will be in between the two extremes as shown in Fig. 8.4. First, the actual thrust profile will start as sea-level thrust profile, as the altitude increases the exit pressure decreases as a result the actual thrust profile move towards the vacuum thrust profile.

Using the above results the Two-Dimensional trajectory is optimized and results are shown in the below section.

8.3.2 Two-Dimensional Trajectory Optimization

With the updated Gross Lift-off Mass, the Two-Dimensional Ascent Trajectory for polar orbit having inclination 90° is optimized in two different cases, one having the objective function as maximizing the orbit radius and the other having maximizing the terminal velocity. Since the launch is from Kulasekharapattinam, there will be no land mass constraints, the launch azimuth is taken as $A_{zl} = 180°$.

8.3.2.1 Maximizing Altitude as objective function

The terminal values obtained for the case of maximizing altitude are shown in Table. 8.5, the pitch and yaw rates optimized using PSO and SQP (with initial guess from PSO) are shown in Table. 8.4. The population size in the PSO is taken to be 100 and is performed 100 iterations and the SQP (`fmincon`l) is implemented till 1500 functional evaluations. The optimized Altitude time history plot is shown in Fig. 8.5, Velocity time history is shown in Fig. 8.6 the time history of Flight path angle and Dynamic pressure are shown in Fig. 8.7 and 8.8.

Table 8.4: Optimized Pitch Rates for second and third stages

PSO	Stage-2	-0.9913	0.8953	1.0837	-0.1469	-0.6598	1.1064	0.9875	1.8129	-0.4639	-1.1473
	Stage-3	0.4162	1.1403	-1.3701	1.7107	0.9626	-1.4520	1.0888	-0.4089	-0.3625	-1.0317
SQP	Stage-2	1.1390	-0.0996	-0.0162	-0.0067	0.3163	0.1463	0.1010	0.1062	-0.1330	0.8974
	Stage-3	1.0371	-0.0318	0.2094	0.1697	0.1084	0.0004	0.1223	0.1963	0.2749	0.2754

Table 8.5: Optimized results obtained from PSO and SQP

	PSO	**SQP (fmincon)**
Final Altitude (km)	455.1115	490.6674
Orbital velocity for this altitude (m/s)	7637.5509	7617.7577
Terminal velocity obtained (m/s)	5614.8200	6118.2836
Initial Pitch Kick-rate (deg/sec)	0.9587	1.4717
Maximum Dynamic Pressure (kPa)	47.7008	54.9949
Orbital Inclination (deg)	90.8413	90.7787
Final Flight Path Angle (deg)	-1.0127	1.4779
Total time till insertion (sec)	598.1	598.1
Optimal Coast time after stage-2 (sec)	9.2270	9.2270

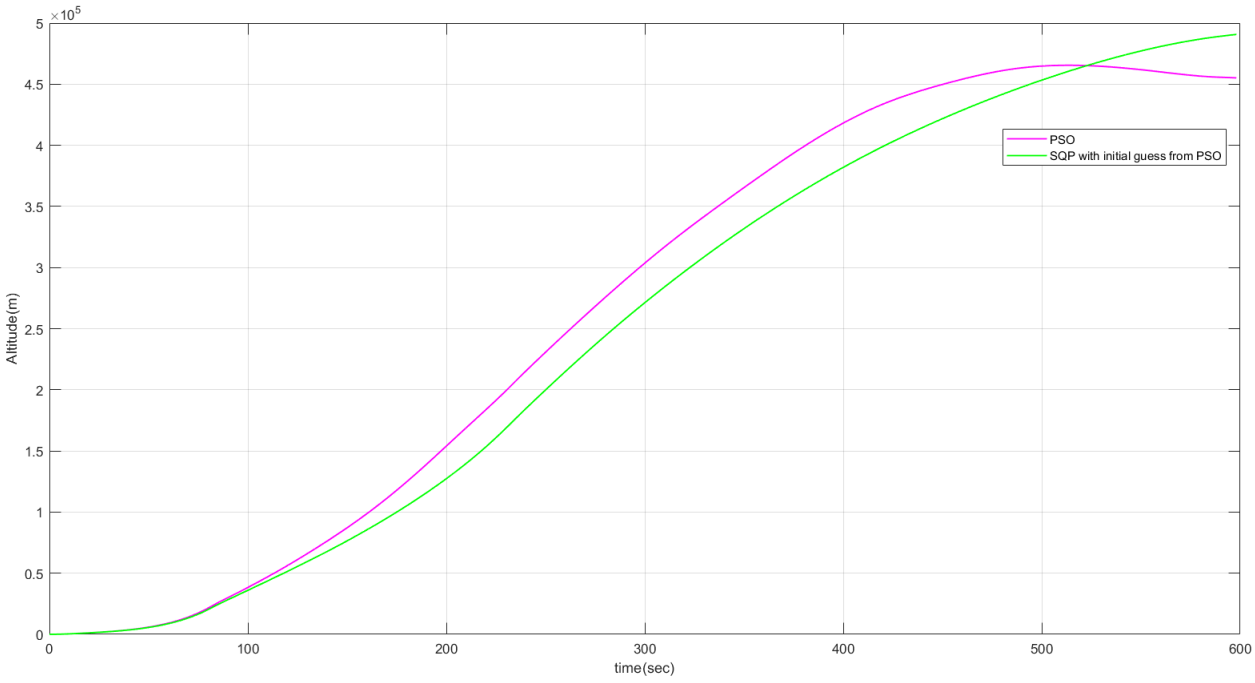

Figure 8.5: Altitude Time History for the case of maximizing altitude

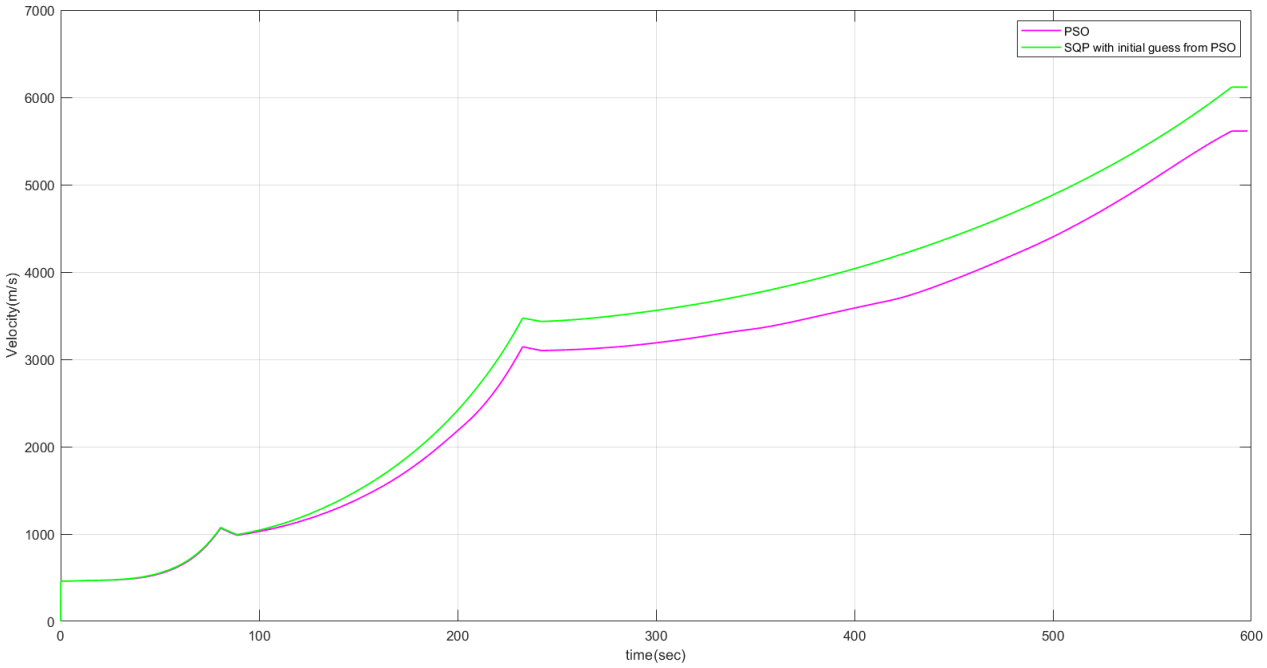

Figure 8.6: Velocity Time History for the case of maximizing altitude

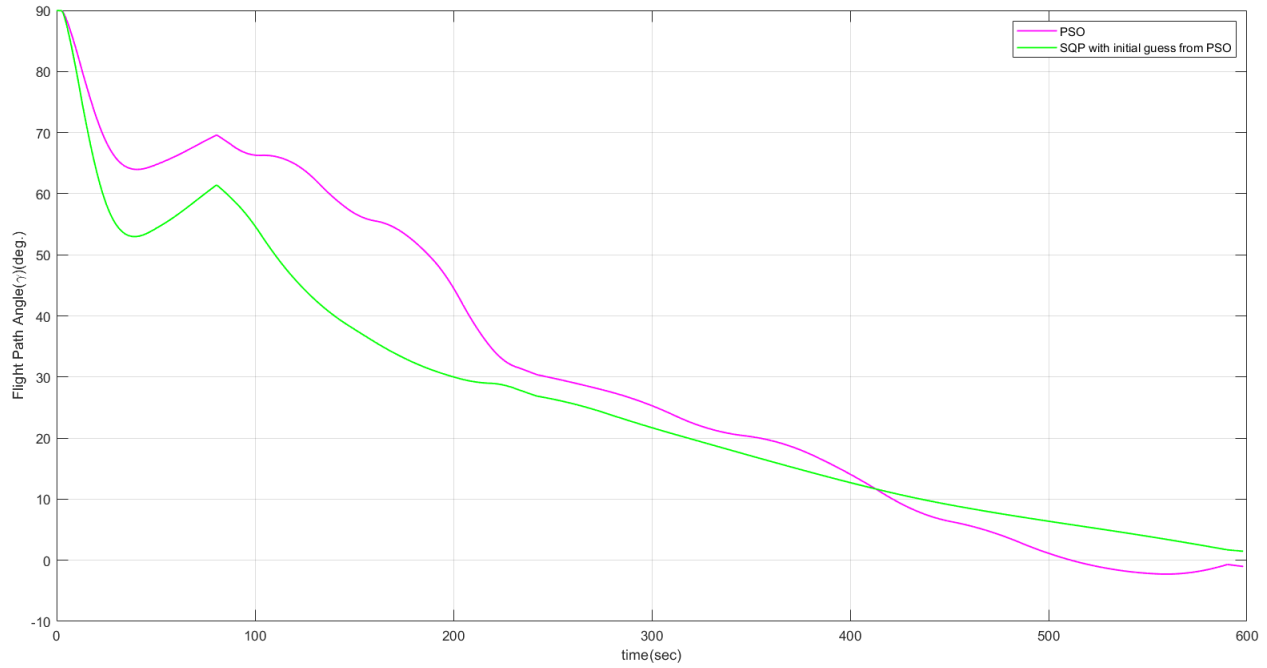

Figure 8.7: Flight Path Angle Time History for the case of maximizing altitude

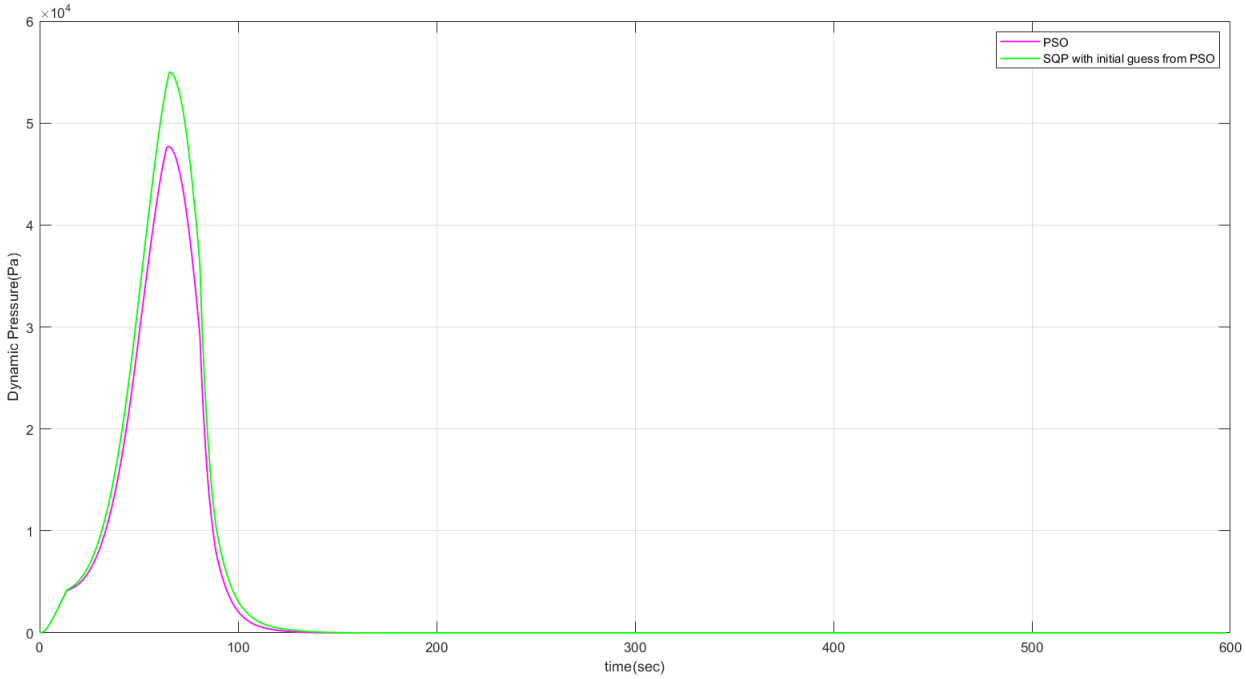

Figure 8.8: Dynamic Pressure Time History for the case of maximizing altitude

8.3.2.2 Maximizing Velocity as objective function

The terminal values obtained for the case of maximizing velocity are shown in Table. 8.7, the pitch and yaw rates optimized using PSO and SQP (with initial guess from PSO) are shown in Table. 8.6. The population size in the PSO is taken to be 100 and is performed 100 iterations and the SQP (`fmincon`) is implemented till 1500 functional evaluations. The optimized Altitude time history plot is shown in Fig. 8.9, Velocity time history is shown in Fig. 8.10 the time history of Flight path angle and Dynamic pressure are shown in Fig. 8.11 and 8.12.

Table 8.6: Optimized Pitch Rates for second and third stages

PSO	Stage-2	-1.9102	1.4697	0.5590	1.5648	-0.1440	1.3782	0.1269	1.6583	-0.4472	-1.6303
	Stage-3	0.4476	0.0643	0.5072	0.4718	0.8720	-1.8249	1.0350	0.3553	0.3001	0.3489
SQP	Stage-2	1.0574	-0.1702	0.1883	0.1769	0.1065	0.0729	0.0107	0.1722	-0.0455	0.9291
	Stage-3	1.1309	-0.2021	0.2115	0.1222	0.1082	0.1019	0.1574	0.2039	0.2138	0.2106

Table 8.7: Optimized results obtained from PSO and SQP

	PSO	**SQP (fmincon)**
Final Altitude (km)	501.9450	495.7358
Orbital velocity for this altitude (m/s)	7611.5118	7614.9487
Terminal velocity obtained (m/s)	5430.0985	6103.3886
Initial Pitch Kick-rate (deg/sec)	0.7171	1.4717
Maximum Dynamic Pressure (kPa)	45.3471	54.9933
Orbital Inclination (deg)	91.0138	90.7923
Final Flight Path Angle (deg)	1.8564	1.6650
Total time till insertion (sec)	600.3	600.3
Optimal Coast time after stage-2 (sec)	11.3163	11.3163

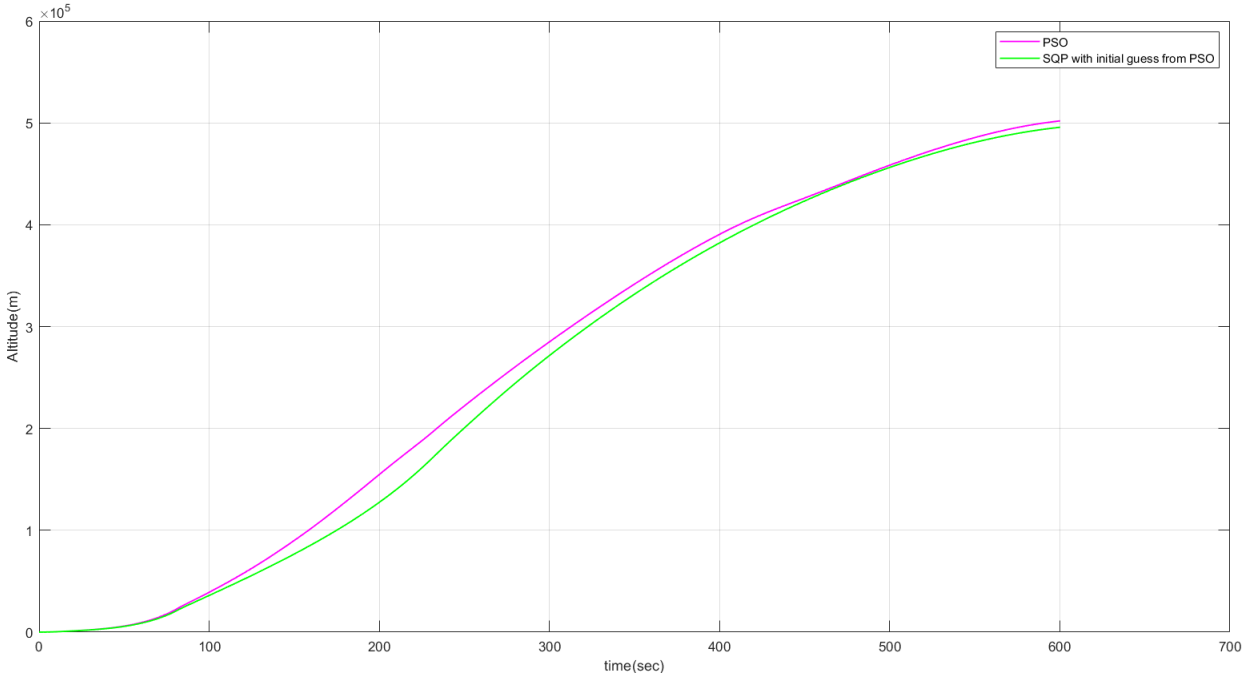

Figure 8.9: Altitude Time History for the case of maximizing velocity

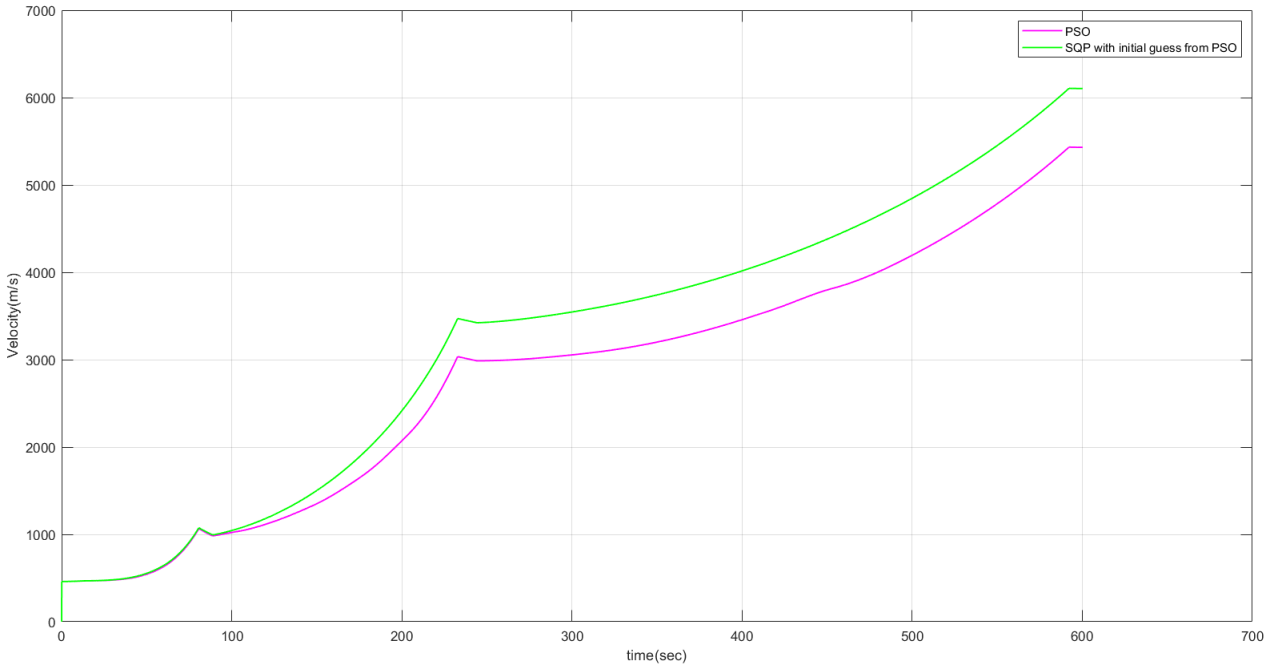

Figure 8.10: Velocity Time History for the case of maximizing velocity

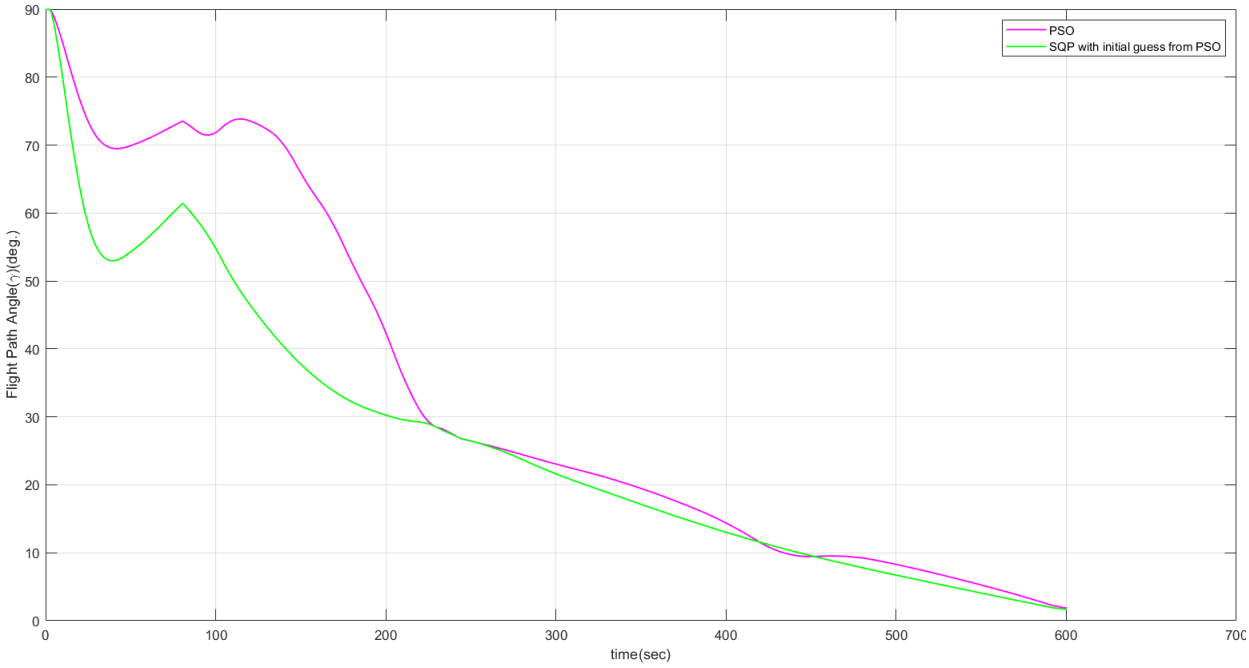

Figure 8.11: Flight Path Angle Time History for the case of maximizing velocity

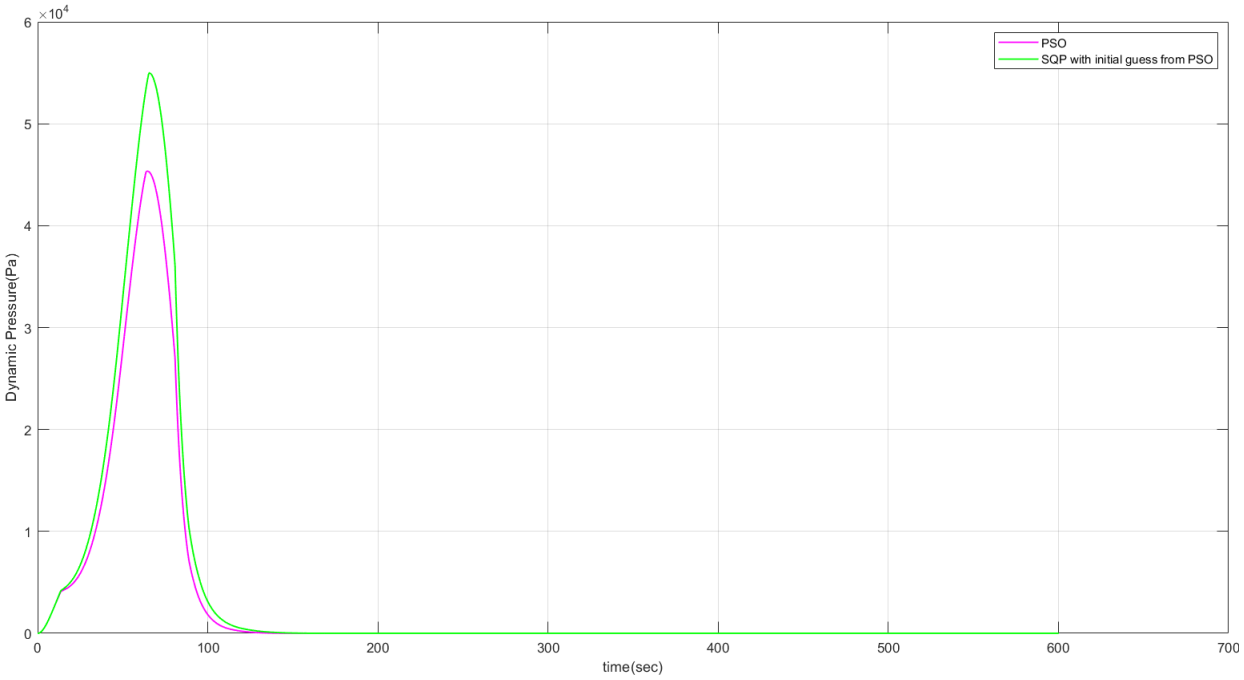

Figure 8.12: Dynamic Pressure Time History for the case of maximizing velocity

8.4 Multi-Disciplinary Feasible Method(MDF)

As discussed earlier, the values obtained from launch vehicle configuration/sizing and aerodynamics analysis are same for all methods. All the disciplines are integrated according to the MDF architecture and are optimized as a single optimization problem with problem statement described in Chapter 7. Similar to FPI, two cases with different objective functions are considered here and optimized using MDF method. The results are shown below.

8.4.1 Maximizing Altitude

Maximizing the altitude is the global objective function and is optimized by using MDF method. All disciplines interact with each other and finally attains global optimum after the rigorous multi-disciplinary search process. For the hybrid optimization solver, the population size in the PSO is taken to be 100 and iterations to be 100 and the SQP (`fmincon`) is implemented till 1500 functional evaluations. The optimized values for the propulsion, trajectory for the case of maximizing altitude are displayed below.

8.4.1.1 Solid Rocket Motor Optimization

As discussed in the FPI method, ($\frac{L}{D}$) ratio is also considered as design variable with the above mentioned search space. The optimal Grain geometry parameters are shown in Table. 8.8. The optimal thrust profiles are shown in Fig. 8.13. With change in length of the

Table 8.8: Optimized grain geometry parameters

	N	ϵ	w	f	R_i	$\frac{L}{D}$
PSO	5	0.2415	0.4678	0.0137	0.1305	4.4993
SQP	5	0.2536	0.3961	0.0144	0.1619	4.4661

motor the gross lift-off mass changes from the value obtained from the launch vehicle configuration/sizing and the updated values for both PSO and SQP are shown in Table. 8.9.

Table 8.9: Updated Gross Lift-off Mass

	GLOM
PSO	69442.3669kg
SQP	69073.1602kg

8.4.1.2 Two-Dimensional Trajectory Optimization

The two-dimensional trajectory is optimized along with SRM using MDF method. The optimized terminal values obtained using PSO and SQP are shown in Table. 8.11, the optimized pitch rates are shown in Table. 8.10.

Table 8.10: Optimized Pitch Rates for second and third stages

PSO	Stage-2	0.8690	0.4556	-0.6660	0.6621	-1.5033	1.4154	-1.1029	0.3199	-1.2790	1.3694
	Stage-3	1.5385	-0.7465	-0.2684	0.2757	-0.0819	1.0138	-1.2499	-0.3575	1.3891	-0.8294
SQP	Stage-2	1.1185	0.7028	-0.4129	0.9103	-1.2484	1.5945	-0.8767	0.5168	-1.1320	1.3942
	Stage-3	1.4860	-0.6738	-0.1457	0.3749	0.0191	1.1469	-0.9431	-0.1435	1.3960	-0.7807

The optimized Altitude time history plot is shown in Fig. 8.14, Velocity time history is shown in Fig. 8.15 the time history of Flight path angle and Dynamic pressure are shown in Fig. 8.16 and 8.17.

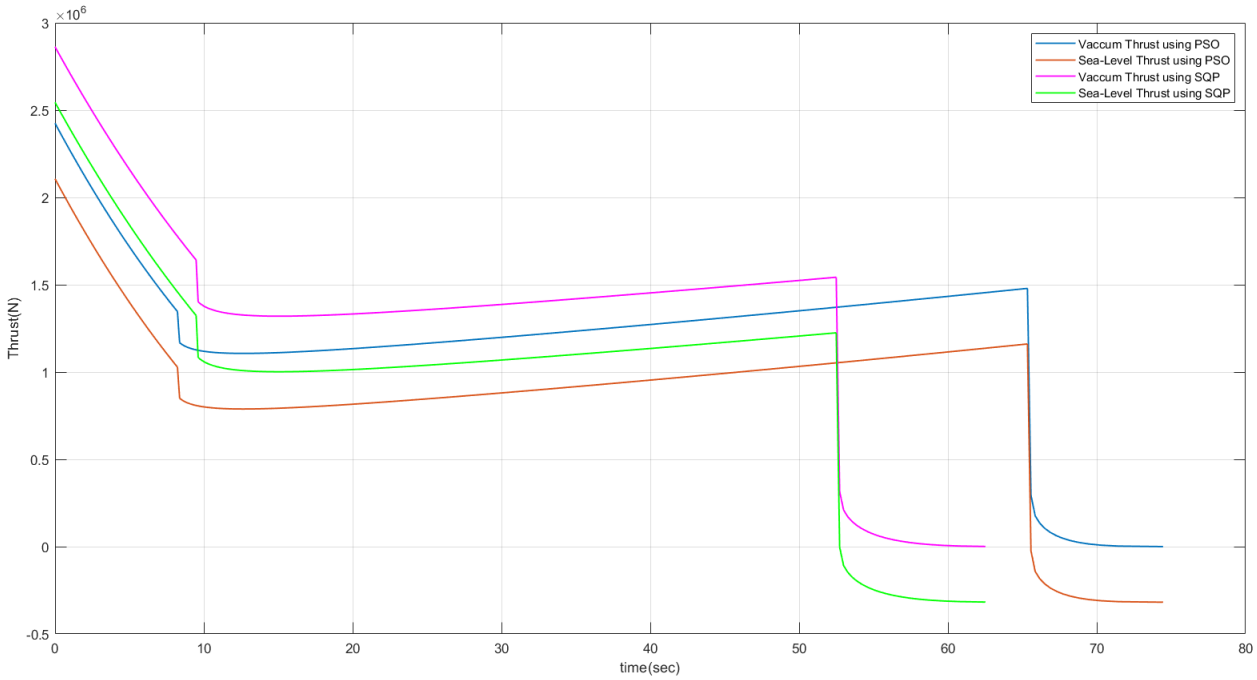

Figure 8.13: Vaccum and Sea-level Thrust time profile for optimized grain geometry using MDF

Table 8.11: Optimized results obtained from PSO and SQP

	PSO	**SQP (fmincon)**
Final Altitude (km)	678.1253	487.3985
Orbital velocity for this altitude (m/s)	7515.8897	7619.5710
Terminal velocity obtained (m/s)	4745.4332	5650.2839
Initial Pitch Kick-rate (deg/sec)	1.0229	1.1957
Maximum Dynamic Pressure (kPa)	43.8633	51.9853
Orbital Inclination (deg)	91.5123	90.8637
Final Flight Path Angle (deg)	11.3644	2.3149
Total time till insertion (sec)	583	570
Optimal Coast time after stage-2 (sec)	9.4596	9.4596

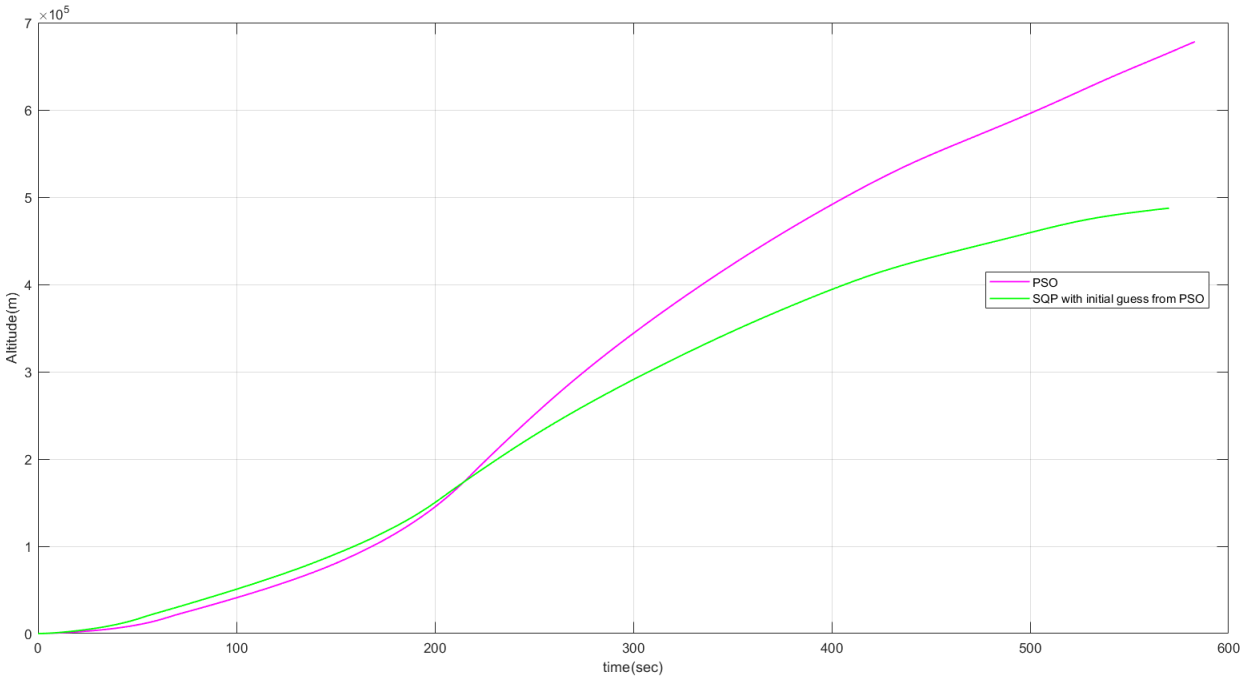

Figure 8.14: Altitude Time History for the case of maximizing velocity

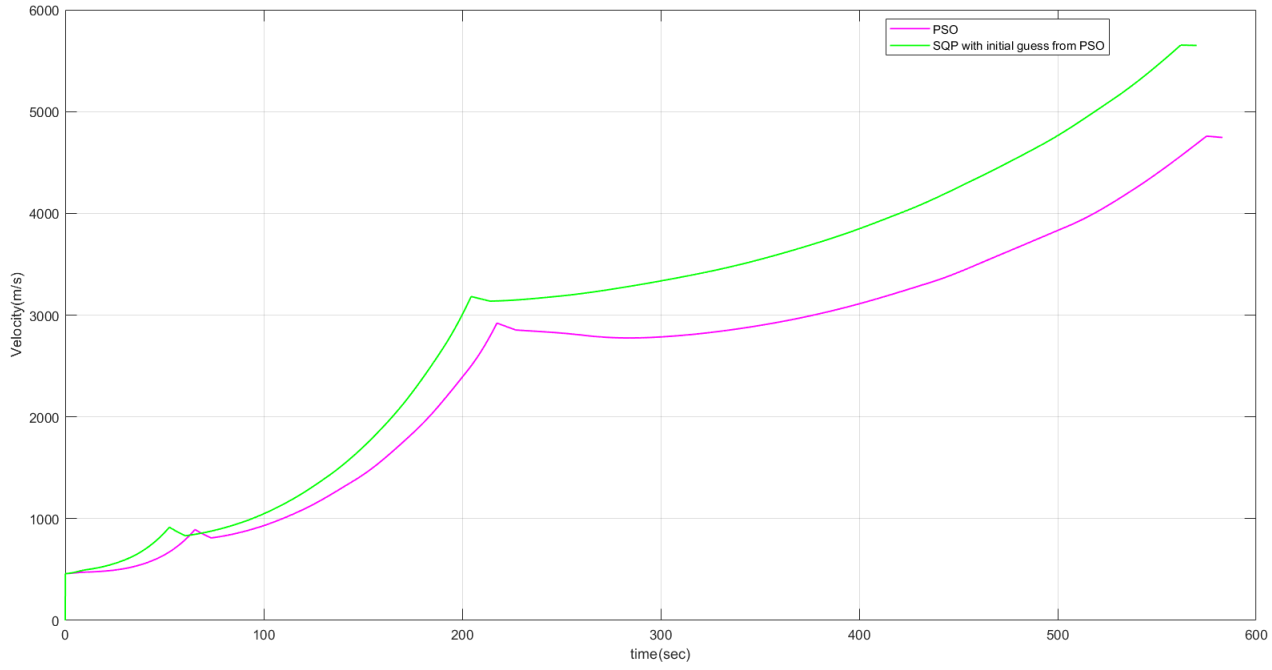

Figure 8.15: Velocity Time History for the case of maximizing altitude

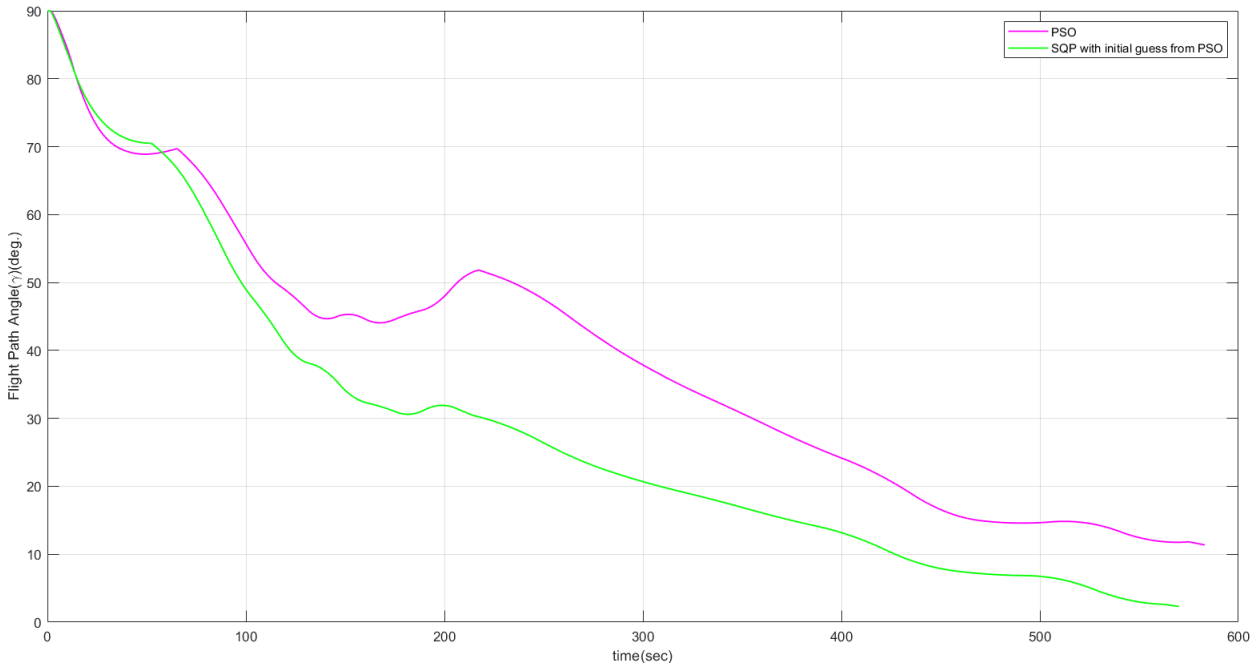

Figure 8.16: Flight Path Angle Time History for the case of maximizing altitude

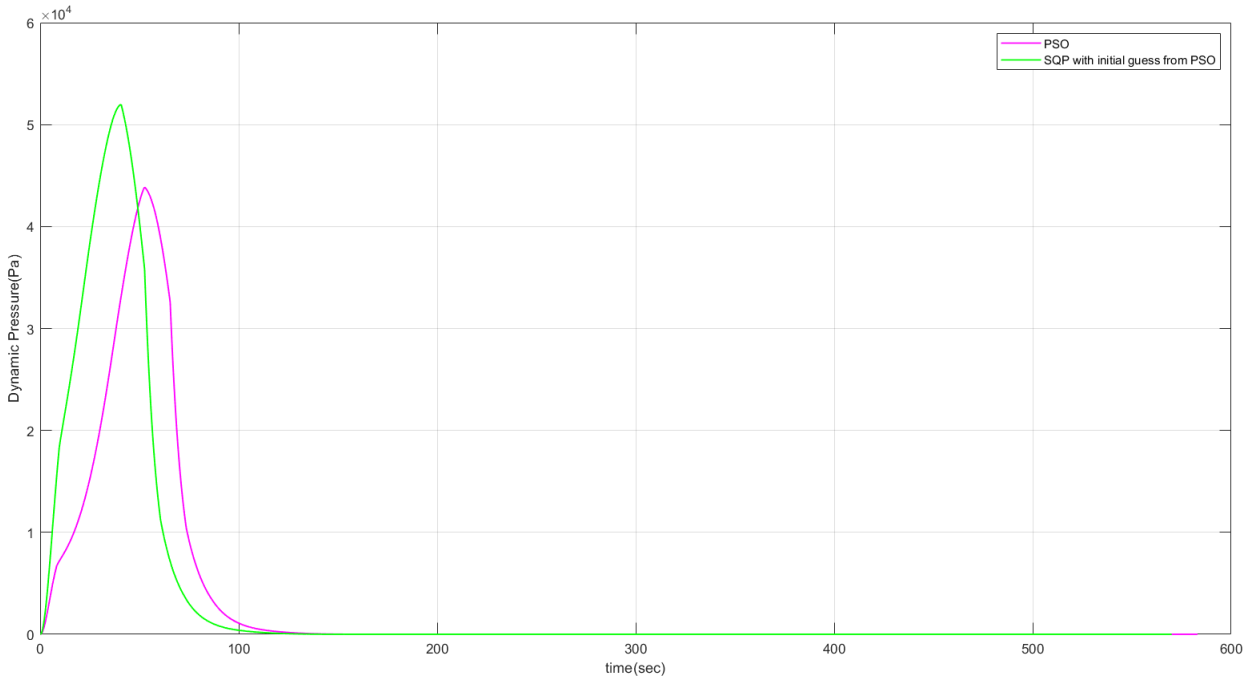

Figure 8.17: Dynamic Pressure Time History for the case of maximizing altitude

8.4.2 Maximizing Velocity

Maximizing the velocity is the global objective function and is optimized by using MDF method. All disciplines interact with each other and finally attains global optimum after the rigorous multi-disciplinary search process which maximized the velocity. For the hybrid optimization solver, the population size in the PSO is taken to be 100 and iterations to be 100 and the SQP (`fmincon`)is implemented till 1500 functional evaluations. The optimized values for the propulsion,trajectory for the case of maximizing velocity are displayed below.

8.4.2.1 Solid Rocket Motor Optimization

The optimal grain geometry parameters are shown in Table. 8.12. With change in length of the motor the gross lift-off mass changes from the value obtained from the launch vehicle configuration/sizing and the updated values for both PSO and SQP are shown in Table. 8.12. The optimal thrust profiles are shown in Fig. 8.18.

Table 8.12: Optimized Grain Geometry Parameters

	N	ϵ	w	f	R_i	$\frac{L}{D}$
PSO	3	0.1285	0.4005	0.0543	0.3114	4.4205
SQP	3	0.2820	0.4066	0.02160	0.0554	4.3500

With change in length of the motor the gross lift-off mass changes from the value obtained from the launch vehicle configuration/sizing and the updated values for both PSO and SQP are shown in Table.‘ 8.13.

Table 8.13: Updated Gross Lift-off Mass

	GLOM
PSO	68565.7352kg
SQP	67783.0002kg

8.4.2.2 Two-Dimensional Trajectory Optimization

The two-dimensional trajectory is optimized along with SRM using MDF method with maximizing velocity as the objective function. The optimized terminal values obtained

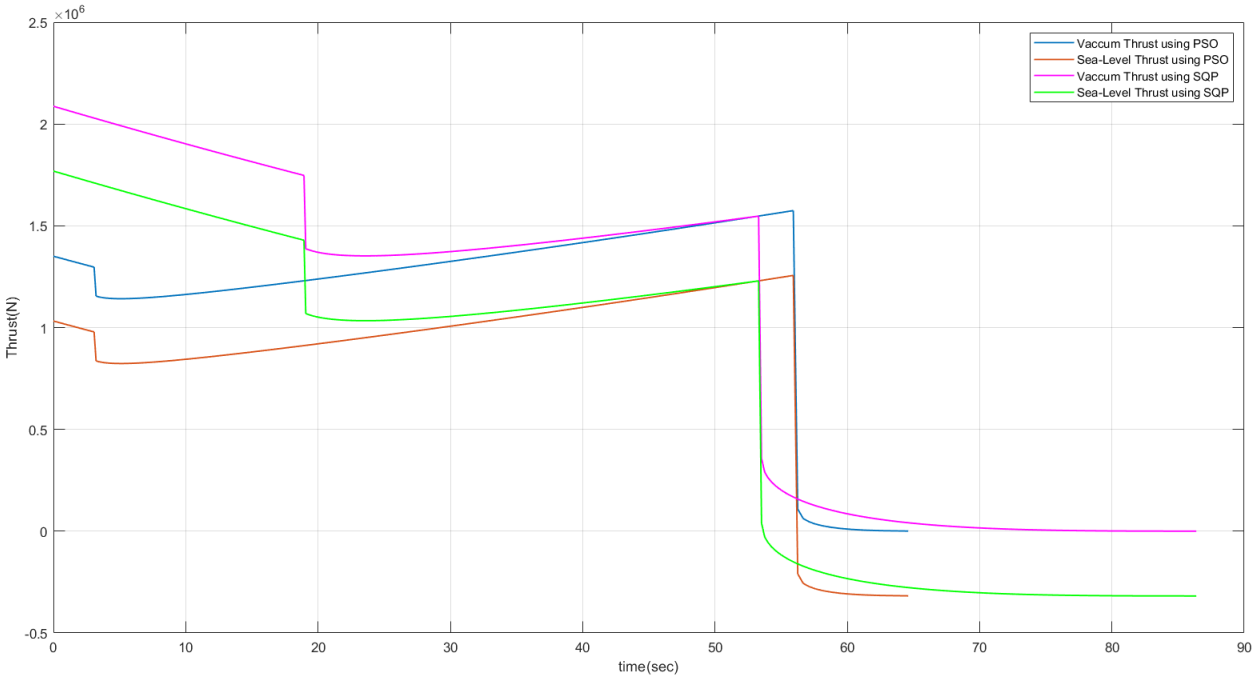

Figure 8.18: Vaccum and Sea-level Thrust time profile for optimized grain geometry using MDF

Table 8.14: Optimized Pitch Rates for second and third stages

PSO	Stage-2	-0.5312	-0.0299	-1.0432	1.1756	1.5007	0.3175	-0.8647	0.6073	-0.3183	-1.6009
	Stage-3	-0.5786	0.3867	1.0014	0.1807	1.6799	-0.0775	0.5733	-1.5609	0.3433	0.6489
SQP	Stage-2	1.1401	1.0335	-0.5764	0.4750	0.3196	0.0952	-0.4291	0.6516	0.5155	0.2515
	Stage-3	1.0908	0.0343	0.0683	-0.0953	0.3548	-0.1380	0.2306	-0.1151	0.0748	-0.0252

Table 8.15: Optimized results obtained from PSO and SQP

	PSO	**SQP (fmincon)**
Final Altitude (km)	500.7135	476.3581
Orbital velocity for this altitude (m/s)	7612.1931	7625.7049
Terminal velocity obtained (m/s)	4071.9105	5860.8212
Initial Pitch Kick-rate (deg/sec)	0.6171	0.8811
Maximum Dynamic Pressure (kPa)	60.2473	56.6446
Orbital Inclination (deg)	91.0081	90.8102
Final Flight Path Angle (deg)	-10.5499	2.0780
Total time till insertion (sec)	574.4	571.7
Optimal Coast time after stage-2 (sec)	10.3662	10.3662

using PSO and SQP are shown in Table. 8.15, the optimized pitch rates are shown in Table. 8.14.

The optimized Altitude time history plot is shown in Fig. 8.19, Velocity time history is shown in Fig. 8.20 the time history of Flight path angle and Dynamic pressure are shown in Fig. 8.21 and 8.22.

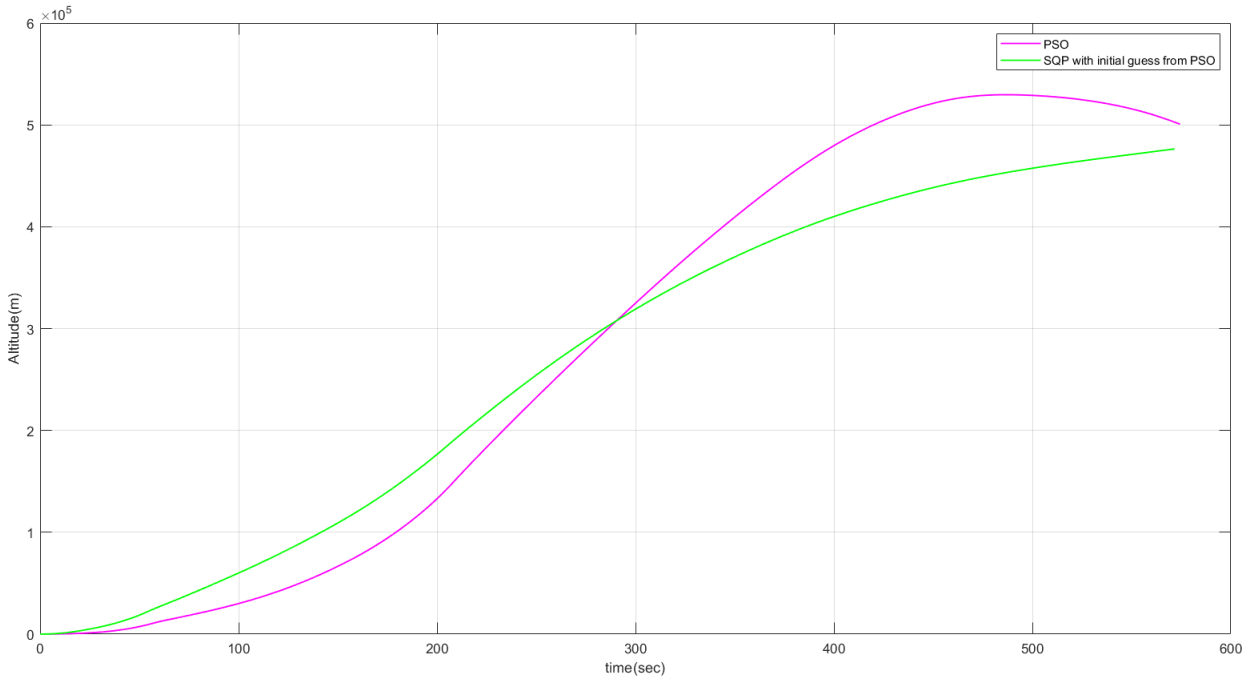

Figure 8.19: Altitude Time History for the case of maximizing velocity

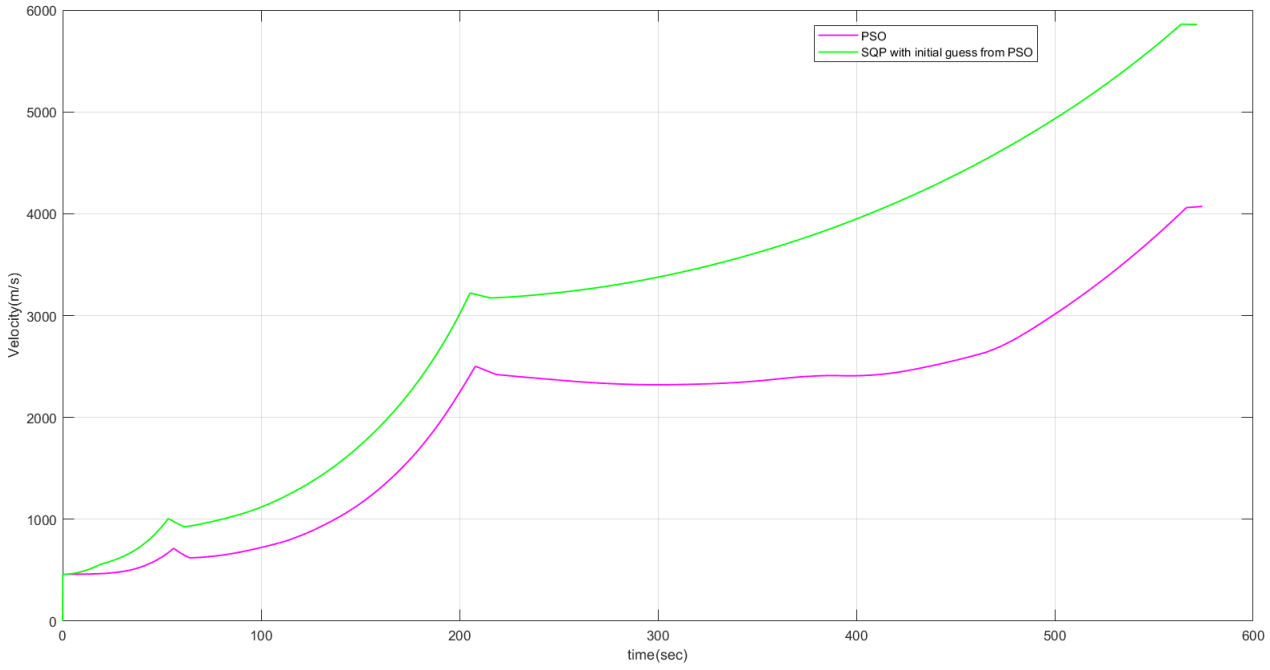

Figure 8.20: Velocity Time History for the case of maximizing velocity

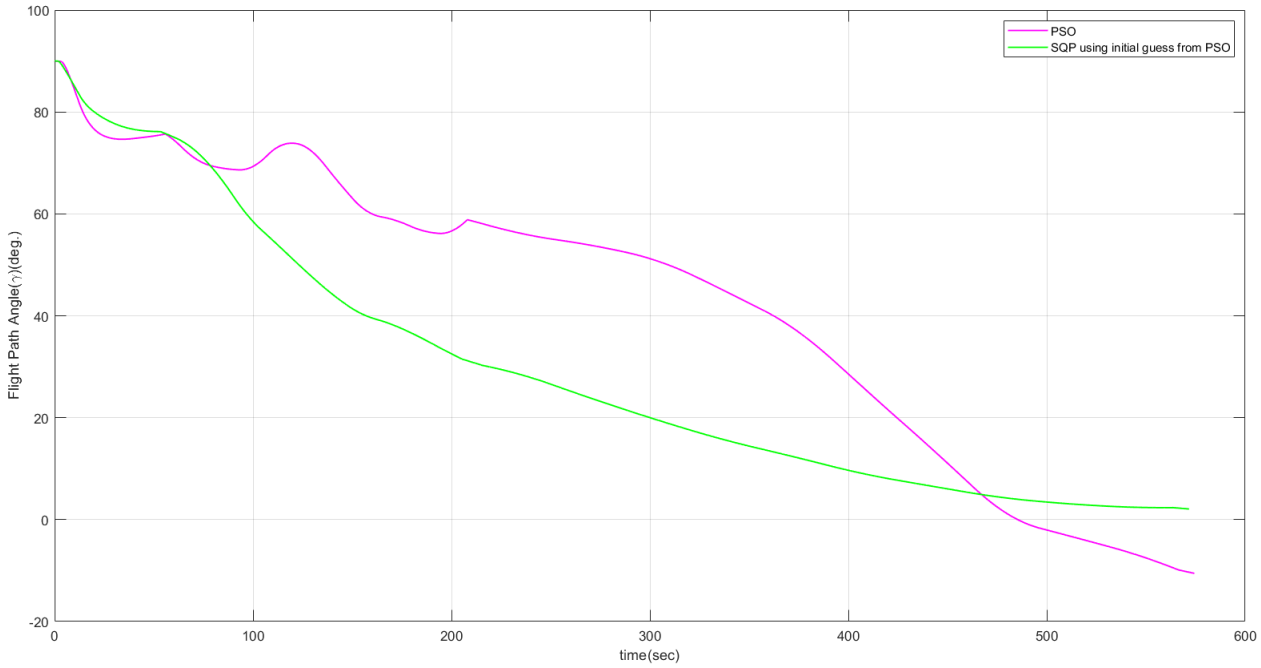

Figure 8.21: Flight Path Angle Time History for the case of maximizing velocity

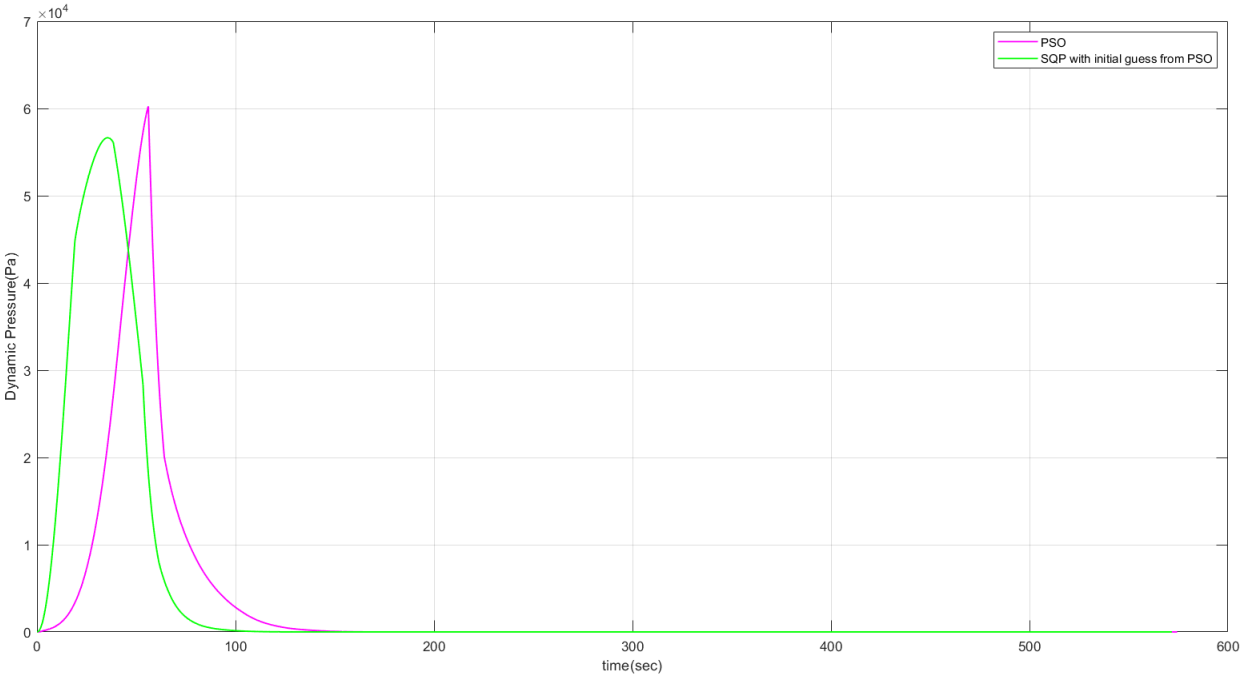

Figure 8.22: Dynamic Pressure Time History for the case of maximizing velocity

From the thrust profiles computed in both optimization methods, it can be seen that SRM is burning in three phases as assumed in formulation, the phases are regression which is due to the spikes of the star geometry followed by slightly progressive phase and finally tail-off phase which is due to leftover sliver.

The altitude, velocity, flight path angle, dynamic pressure time histories provides a fair idea of the trajectory and acts as sources to compare different trajectories. From the above results, It is observed that the solution from SQP (`fmincon`) is always superior to the solution obtained by only PSO in terms of altitude, velocity, Flight Path angle, dynamics pressure. This is the advantage of combining both heuristic and gradient based methods which helps in both global exploration and local exploitation of the search space. In the velocity-time plot, the sharp peaks indicates the separation of the former stage followed by coasting phase and ignition of later stage. The Flight path angle starts from 90° at ground(vertical flight) and reaches the optimal terminal value(tangential injection). The dynamic pressure as expected follows the bell shaped curve which is minimum at initial phase because of less velocity and increases with the velocity till it reaches the peak value thereafter the density of atmosphere drops drastically which decreases the dynamic pressure.

The best solutions from both FPI and MDO methods, attained the required altitude correctly but the terminal velocities obtained using both methods are nearly 1.5km/s short of required orbital velocities. This is due to the shot burning times of the SRMs which are not supplying the needed velocity to the launch vehicle first stage. One can emphasize the fact that the best terminal velocities obtained in both methods for case of the maximizing velocity as objective function. Though the terminal Velocity obtained by FPI method is nearly 250m/s greater than that of MDF but the reduction in total Gross-Lift off Mass using MDO methods is tremendous which is found to be nearly 7.23 tonnes lesser than that of obtained using FPI method. This illustrates the potential of MDO methods in searching global maximum over traditional FPI methods.

8.5 Three-Dimensional Ascent Trajectory Optimization

The feasibility of launching the small-size expendable launch vehicle from Sriharikota (SHAR) to polar orbit(i=90°) is explored. The important concern is here is that the presence of range safety constraints. The spent stages of launch vehicle must not fall into the land masses. Range safety constraints are incorporated into the trajectory as Impact Point Constraints. Analytical calculation of Impact Point Constraints are described in chapter 6.

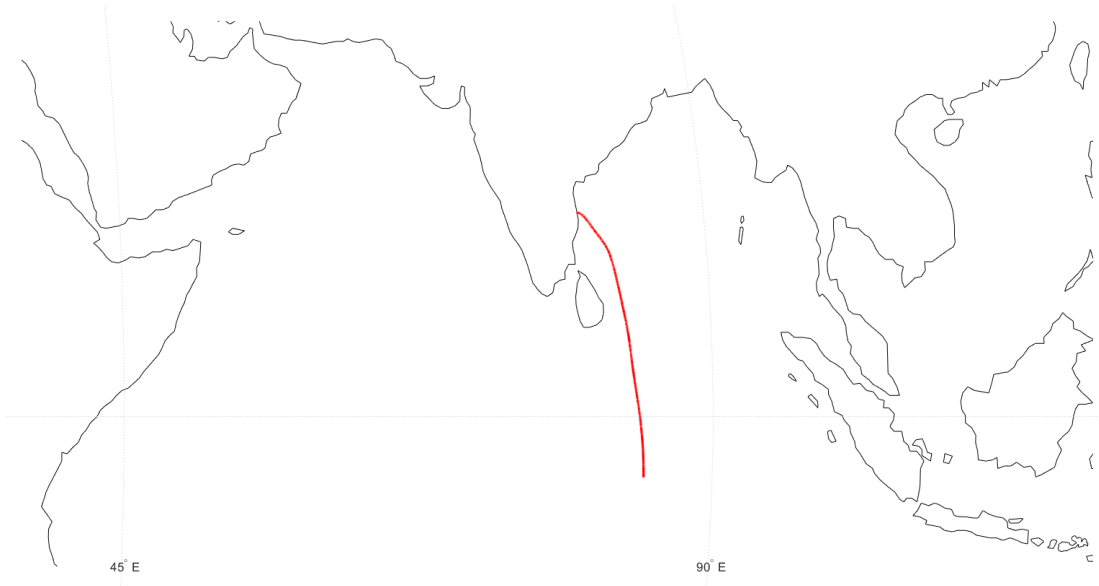

Figure 8.23: Ground Track of Three-Dimensional Ascent Trajectory with Yaw-Maneuvers

The stage-2 and stage-3 are divided into five segments each and yaw rate is assigned to each segment and also the yaw-kick rate is also considered in addition to pitch kick rate in constant turn over phase, finally making a total of 32 design variables. The launch vehicle is launched at a different, safety launch azimuth angle($A_{zl} = 140°$), and yaw rates in addition to pitch rates are also optimized to attain the required orbit inclination. Only the Particle Swarm Optimization with population size of 50 and 100 iterations is used to optimize the trajectory due to increase in computational time with addition of extra design variables. Constant thrust is used instead of SRM thrust for all the stages. The ground track of the three-dimensional trajectory is shown in Fig. 8.23. The optimized results are shown below.

Table 8.16: Optimized Pitch Rates for second and third stages

PSO	Stage-2	0.5469	1.0029	2.0000	-1.5423	1.0689	-1.5727	-1.9348	-1.1887	0.3911	-1.5958
	Stage-3	2.0000	0.0924	-0.0738	-1.4613	1.7914	0.5907	-0.9157	0.2324	-1.5442	-0.0481

Table 8.17: Optimized Yaw Rates for second and third stages

PSO	Stage-2	-1.9533	-2.0000	-2.0000	-2.0000	-1.9535
	Stage-3	-2.0000	-2.0000	-2.0000	-2.0000	-1.7238

Table 8.18: Optimized results obtained from PSO and SQP

	PSO
Final Altitude (km)	642.2778
Orbital velocity for this altitude (m/s)	7535.0539
Terminal velocity obtained (m/s)	6540.2115
Initial Pitch Kick-rate (deg/sec)	2
Initial Yaw Kick-rate (deg/sec)	-1.4992
Maximum Dynamic Pressure (kPa)	53.3073
Orbital Inclination (deg)	92.7153
Final Flight Path Angle (deg)	1.6073
Total time till insertion (sec)	659.4
Optimal Coast time after stage-2 (sec)	8.0000

It can be observed that most of the turn is happening at initial phase of trajectory where the velocity is less compared to the other phases because it is easier to turn the vehicle at less velocities than in higher velocities. The terminal velocity of the launch vehicle is lesser than the required velocity because significant amount of thrust is used for three-dimensional maneuver of the launch vehicle, one can emphasize from this fact that three-dimensional/plane change maneuvers are very expensive and launching from launch sites such as Kulasekharapattinam avoids the need of three-dimensional maneuvers.

Chapter 9
Conclusions

It is observed that the results from both the Fixed Point Iteration Method (FPI) and Multi-Disciplinary Feasible (MDF) are satisfactorily obtained, except for the of terminal velocities, the terminal velocities obtained in both the methods are nearly 1.5 km/s lesser than the required orbital velocity. This is due to the short burning times of the Solid Rocket Motor which result in lesser impulse imparted to the launch vehicle after the first stage. It is observed that the length of the solid rocket motor obtained from the launch vehicle sizing is not sufficient and it also is considered as additional design variable and is optimized to accomplish the mission objective. It is found that optimal launch vehicle design obtained using the MDF Method outperformed the design obtained from Traditional FPI method. The gross lift-off mass obtained using MDF method is 9.64% i.e., 7.23 tonnes lesser than the gross lift-off mass obtained using FPI method. This is due to the fact that the disciplines or subsystems in the FPI have their own objective functions and are optimized accordingly, but the MDF method has only single global objective function and all the subsystems interact with each other to optimize the global objective function. This showcases the potential and robustness of Multi-Disciplinary Design Optimization methods in the field of Launch Vehicle Design. It can be seen that the results obtained by using Hybrid optimization solver (PSO+SQP) are more superior than those with single optimizer (PSO). The current study concludes that MDO methods are far more superior than that of traditional design methods at least in the case of launch vehicle design.

9.1 Future Scope

So far the techniques, methodologies used in the current work are a tip of an iceberg. More practical launch vehicle design can be obtained by including more newer techniques and methodologies. The future scope of the current work is described below.

1. The solid grain geometry considered in the current work is two-dimensional. It can be extended to three-dimensional (Finocyl Grains) to obtain the actual "M" shaped thrust profile.

2. The Solid Rocket Motor is of internal burning type and the combustion products coming from the top end cause *erosion* at nozzle throat. This effect can be incorporated with the SRM design.

3. The nozzle configuration and propellant composition have been considered as fixed; they can be included as design variables and can be optimized to obtain a more efficient design.

4. Only the thrust of first stage is optimized and for the remaining two stages the thrust is considered as constant; the thrust profile for other two stages can also be modelled and optimized.

5. Though the feasibility of three-dimensional trajectory is explored, it is not employed in FPI and MDF methods. It can also be incorporated in the design problem as well.

6. The geometry of the launch vehicle can also be incorporated as separate discipline.

7. Launch vehicle design is carried out by assuming zero gimbal angle, i.e. the thrust vector acts along the vehicle axis. The gimbal angle can be varied to obtain the optimal design.

8. Only dynamic pressure is considered as a path constraint. The heat flux and bending load constraints can also be incorporated to obtain a feasible design.

9. In the current work, only one type of Multi-Disciplinary Design Optimization method is used. There are many architectures that can be applied to launch vehicle design.

Bibliography

[1] B. N. Suresh and K. Sivan, *Integrated Design Aspects of Space Transportation System*. New Delhi: Springer India, 2015.

[2] S. Ghanekar, "Ascent trajectory optimization for a three-stage launch vehicle with constant thrust," Internship Thesis, Indian Institute of Space Science and Technology, July 2020.

[3] S. Acik, "Internal ballistic design optimization of a solid rocket motor," Master's thesis, The Graduate School of Natural and Applied Sciences of Middle East Technical University, May 2010.

[4] O. Yusel, "Ballistic design optimization of three-dimensional grains using genetic algorithms," Master's thesis, The Graduate School of Natural and Applied Sciences of Middle East Technical University, September 2012.

[5] R. Hartfield, R. Jenkins, J. Burkhalter, and W. Foster, "A review of analytical methods for solid rocket motor grain analysis," in *39th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*.

[6] L. D. Davis, *Coordinate Systems for the Space Shuttle Program*. NASA, 1974.

[7] G. L. Brauer, D. E. Cornick, A. R. Habeger, F. M. Petersen, and R. Stevenson, *Program to Optimize Simulated Trajectories (POST) Volume 1: Formation Manual*. NASA, 1975.

[8] E. C. COSKUN, "Multistage launch vehicle design with thrust profile and trajectory optimization," Ph.D. dissertation, THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF MIDDLE EAST TECHNICAL UNIVERSITY, 2014.

[9] "Isro pslv dogleg maneuver." [Online]. Available: https://commons.wikimedia.org/wiki/File:ISRO_PSLV_dogleg_maneuver.jpg

[10] M. Balesdent, "Multidisciplinary design optimization of launch vehicles," 2011.

[11] F. Castellini, "Multidisciplinary design optimization for expendable launch vehicles," Ph.D. dissertation, Politecnico di Milano, 2012.

[12] L. Federici, A. Zavoli, G. Colasurdo, L. Mancini, and A. Neri, "Integrated optimization of ascent trajectory and srm design of multistage launch vehicles," 2019.

[13] C. Geethaikrishnan, P. Mujumdar, K. Sudhakar, and V. Adimurthy, "A hybrid mdo architecture for launch vehicle conceptual design," in *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. Orlando, Florida: AIAA, April 2010.

[14] "Solid propellant grain design and internal ballistics," NASA, SP-8076.

[15] A. Ricciardi, "Generalized geometric analysis of right circular cylindrical star perforated and tapered grains," *Journal of Propulsion and Power*, vol. 8, no. 1, pp. 51–58, 1992.

[16] A. Rafique, Q. Zeeshan, A. Kamran, and L. Guozhu, "A new paradigm for star grain design and optimization," *Aircraft Engineering and Aerospace Technology*, vol. 87, no. 5, pp. 476–481, 2015.

[17] "Program to optimize simulated trajectories." [Online]. Available: https://ntrs.nasa.gov/api/citations/19750024073/downloads/19750024073.pdf

[18] V. Adimurthy, "Launch vehicle trajectory optimization," *Acta Astronautica*, vol. 15, no. 11, pp. 845–850, 1987.

[19] M. Pontani, "Particle swarm optimization of ascent trajectories of multistage launch vehicles," *Acta Astronautica*, vol. 94, no. 2, pp. 852–864, Feb. 2014.

[20] N. Yoon and J. Ahn, "Trajectory optimization of a launch vehicle with explicit instantaneous impact point constraints for various range safety requirements," *Journal of Aerospace Engineering*, vol. 29, no. 3, p. 06015003, 2016.

[21] K. E. Parsopoulos and M. N. Vrahatis, "Particle swarm optimization method for constrained optimization problems," 2002.

[22] A. Mahjub, N. M. Mazlan, M. Z. Abdullah, and Q. Azam, "Design optimization of solid rocket propulsion: A survey of recent advancements," *Journal Of Spacecraft and Rockets*, vol. 57, no. 1, January–February 2020.

[23] S. Patan, Y. Santhosh Kumar, and S. Nazamuddin, "Design and geometrical analysis of propellant grain configurations of a solid rocket motor," *IJEDR, ISSN: 2321-9939*, vol. 2, no. 4, pp. 3417–3427, 2014.

[24] M. Balesdent, N. Bérend, P. Dépincé, and A. Chriette, "A survey of multidisciplinary design optimization methods in launch vehicle design," *Structural and Multidisciplinary Optimization*, vol. 45, pp. 619–642, 2012.

[25] J. R. R. A. Martins and A. B. Lambe, "Multidisciplinary design optimization: A survey of architectures," *AIAA Journal*, vol. 51, pp. 2049–2075, 2013.

Appendix A

MATLAB CODES

A.1 Launch Vehicle Staging Optimization

```
function [m_i, m_s, m_prop, m_glom, T_m] = lv_staging_opt(
    eps,I_sp,T_by_W,n)
m_pl=500;%kg
g_0=9.80665;%m/sec^2
m_i=zeros(n, 1);
m_s=zeros(n, 1);
m_prop=zeros(n, 1);
T_m=zeros(n, 1);
m_glom=1;


%Exhaust Velocity(=g_0*C_k)
C_k = I_sp.*g_0;
[~,p]=newton_rhapson(eps,I_sp,T_by_W,n);
A_k=zeros(n, 1);
for i=1:n
A_k(i)=(A_k(i) + (1+p*C_k(i))/(C_k(i)*eps(i)*p));
m_glom = (m_glom*(1-eps(i))*A_k(i)/(1-eps(i)*A_k(i)));
end
m_glom=m_glom*m_pl;
m_i(1)=m_glom;
m_prop(1)=(m_i(1)*(A_k(1)-1)/A_k(1));
```

```
m_s(1)=(eps(1)*m_prop(1)/(1-eps(1)));
T_m(1)= (T_m(1) + m_glom*T_by_W(1)*g_0);
for i=2:n
m_i(i) = (m_i(i-1) - m_s(i-1) - m_prop(i-1));
m_prop(i) = (m_i(i)*(A_k(i) - 1)/A_k(i));
m_s(i) = (eps(i)*m_prop(i)/(1-eps(i)));
T_m(i) = (T_m(i) + m_i(i)*T_by_W(i)*g_0);
end

end


%2) Functions for Numerically Solving the final Equation
   obtained by lagrange Equation
function [h, p] = newton_rhapson(eps,I_sp,T_by_W,n)

%delv=10387.92;%m/sec
g_0=9.80665;%m/sec^2
%Known Parameters(Structural Ratios, Specific Impulse,
   Thrust-Weight Ratios)

%Exhaust Velocity(=g_0*C_k)
C_k = I_sp.*g_0;
o_ek = 1.-eps;
C_kme_k=C_k.*o_ek;
p_0=-1/(min(C_kme_k));
h=1;
while h>0.00000000001
 p=(p_0 - f(p_0, n, C_k, eps)/f_dash(p_0, n, C_k));
 h=(abs(f(p, n, C_k, eps)));
 p_0=p;
end
end

function f_p = f(p, n, C_k, eps)
```

```
delv=10387.92;%m/sec
f_p=0;
for i=1:n
    f_p=f_p+C_k(i)*log((1+p*C_k(i))/(C_k(i)*eps(i)*p));
end
f_p = f_p - delv;
end

function f_dash_p=f_dash(p, n, C_k)

f_dash_p=0;
for i = 1:n
f_dash_p = f_dash_p  - C_k(i)/(p*(1+p*C_k(i)));
end
end
```

A.2 Solid Propulsion Analysis

```
%% ----------Solid Propulsion Module----------%%
%Propellent Selected; Ammonium Perchlorate/Hydroxyl
    Terminated poly Butadiene/Aluminium Powder(68:14:18)%
%Design Variables: Initial Burn rate(r), Number of lobes(N),
     Angular Fraction(epsilon), Web thickness(w), Fillet
    radius at corner(f), L/D(Length/Diameter) ratio%
%Known Variables: Exhaust Gas Properties(gamma, MW, R, Tc)
%Functions: 1) function for burn back analysis, 2) function
    for calculating exit pressure%
%Objective Function: Maximise Isp, Minimize sliver

%function [P_c,Th_vac,Th_amb,Isp,t,dt,tb_vac,tb_amb] =
    solid_prop(N,e,w,f,r_i,l_by_d)
function [P_c,Th_vac,Th_amb,Isp,t,dt,tb_vac,tb_amb,m_dot] =
    solid_prop(N,e,w,f,r_i,l_by_d)
%function [P_c,Th_vac,Isp,t,dt,tb_vac] = solid_prop(N,e,w,f,
    r_i,l_by_d)
```

```
%All the inputs are in dimensional
%Increment(x) = 1mm
x = 1*10^(-3);
%Case-1: Using VEGA P241 solid rocket motor exhaust gas
    properties and VEGA P80 solid  rocket motor Configuration
    %
%Ammonium Perchlorate/Hydroxyl Terminated poly Butadiene/
    Aluminium Powder (68:14:18)
%Propellent and Exhaust Gas Properties and other constants
rho = 1770;%kg/m^3
T_cc = 3328.9;%K
g = 1.142;
MW = 27.4;%KJ/Kmol
R_gas = 303.4479;%K
a = 3.5958*10^(-5);% Burn rate coefficient
n = 0.35;%Burn rate exponent

r_out = 2/2;%m %Outer Radius of the solid rocket motor
Ae_At = 16;%Exit to throat area ratio
P_MEOP = 95*100000;%Pa Maximum Operating pressure
A_t = (pi*r_out^2)/(Ae_At);

%Calculating constants
r_p = r_out - w - f;
theta_by_2 = atan2(r_p*sin(pi*e/N)*tan(pi*e/N),(r_p*sin(pi*e
    /N)-r_i*tan(pi*e/N)));
ymax = sqrt((r_out - r_p*cos(pi*e/N))^2 + r_p*sin(pi*e/N)^2)
    -f;
iter = 0;
c_star = sqrt(g*R_gas*T_cc)/(g*sqrt( (2/(g+1))^( (g+1)/(g-1)
    ) ) );
H = r_p*sin(pi*e/N);
B = pi/2 - theta_by_2 + pi*e/N;

%Initializing the output variables
```

```
t = zeros(100000,1);
Th_vac = zeros(100000,1);
Th_amb = zeros(100000,1);
P_c = zeros(100000,1);
r_b = zeros(100000,1);
dt = zeros(100000,1);

%Calculating Pe/Pc (which is constant) for vaccum and
    ambient thrust
options=optimset('Display','off');
options.TolX=1e-10;
options.TolFun=1e-15;
func=@(Pe_Pc) 1/Ae_At - ( ((g+1)/2)^(1/(g-1)) )*(Pe_Pc^(1/g)
    )*sqrt( ((g+1)/(g-1))*(1-Pe_Pc^((g-1)/(g))) );
Pe_by_Pc = fsolve(func,0,options);
C_F_vac = sqrt( (2*g^2/(g-1))*(((2)/(g+1))^((g+1)/(g-1)))
    *(1-Pe_by_Pc^( (g-1)/(g) )) );
Isp = 0.96*c_star*(C_F_vac+(Pe_by_Pc)*Ae_At)/9.81;%n_u =
    0.96 Deliverable Specific Impulse

%Calculating Burning Area
if ( r_p*sin(pi*e/N)/(cos(theta_by_2)) - f) > 0

%Phase-1 Burning
for y=0:x:( r_p*sin(pi*e/N)/(cos(theta_by_2)) - f)
    iter = iter + 1;
    if (y <= r_p*sin(pi*e/N)/(cos(theta_by_2)) - f)
        s1 = r_p*sin(pi*e/N)/sin(theta_by_2) - (y+f)*cot(
            theta_by_2);
        s2 = (y+f)*(pi/2 - theta_by_2 + pi*e/N);
        s3 = (r_p+y+f)*(pi/N-pi*e/N);
        S = 2*N*(s1 + s2 + s3) ;
        A_b = S*l_by_d*2*r_out;
        P_c(iter) = (rho*c_star*a*A_b/A_t)^(1/(1-n));
        Th_vac(iter) = P_c(iter)*(0.99*C_F_vac +(Pe_by_Pc)*
```

```
            Ae_At)*A_t;
70          Th_amb(iter) = P_c(iter)*(0.99*C_F_vac + (Pe_by_Pc)*
            Ae_At-101325*Ae_At/P_c(iter))*A_t;
71      end
72      r_b(iter) = a*P_c(iter)^n;
73      dt(iter) = x/r_b(iter);
74      t(iter+1) = t(iter) + dt(iter);
75  end
76
77  if iter>2
78  %Phase-2 Burning
79  for y=( r_p*sin(pi*e/N)/(cos(theta_by_2)) - f):x:w
80
81      if (( r_p*sin(pi*e/N)/(cos(theta_by_2)) - f)<y) && (y<=w)
82          S = 2*N*((r_p+f+y)*(pi/N - pi*e/N) + (y+f)*( pi/2-
              theta_by_2+pi*e/N - atan2(sqrt((y+f)^2 - H^2),H) -
              theta_by_2));
83          A_b = S*l_by_d*2*r_out;
84          P_c(iter) = (rho*c_star*a*A_b/A_t)^(1/(1-n));
85          Th_vac(iter) = P_c(iter)*(0.99*C_F_vac +(Pe_by_Pc)*
              Ae_At)*A_t;
86          Th_amb(iter) = P_c(iter)*(0.99*C_F_vac + (Pe_by_Pc)*
              Ae_At-101325*Ae_At/P_c(iter))*A_t;
87      end
88
89      r_b(iter) = a*P_c(iter)^n;
90      dt(iter) = x/r_b(iter);
91      t(iter) = t(iter-1) + dt(iter);
92      iter = iter + 1;
93  end
94
95  %Phase-3 Burning
96  if ((y+f)^2 -H^2)>0
97  for y=w:x:ymax
98      if (w<=y) && (y<=ymax) && S>0
```

```
99          G = atan2(sqrt((y+f)^2 -H^2),H)-theta_by_2;
100         zeta = pi - acos( -(r_out^2-r_p^2-(y+f)^2)/(2*r_p*(y+
                f)) );
101         S = 2*N*((y+f)*(B-G-zeta));
102         A_b = S*l_by_d*2*r_out;
103         P_c(iter) = (rho*c_star*a*A_b/A_t)^(1/(1-n));
104         Th_vac(iter) = P_c(iter)*(0.99*C_F_vac +(Pe_by_Pc)*
                Ae_At)*A_t;
105         Th_amb(iter) = P_c(iter)*(0.99*C_F_vac + (Pe_by_Pc)*
                Ae_At-101325*Ae_At/P_c(iter))*A_t;
106     end
107     r_b(iter) = a*P_c(iter)^n;
108     dt(iter) = x/r_b(iter);
109     t(iter) = t(iter-1) + dt(iter);
110     iter = iter + 1;
111 end
112
113 end
114
115 end
116
117 end
118
119 Th_vac = Th_vac(Th_vac==real(Th_vac));
120 Th_amb = Th_amb(Th_amb==real(Th_amb));
121 t = t(t==real(t));
122 P_c = P_c(P_c==real(P_c));
123 r_b = r_b(r_b==real(r_b));
124 dt = dt(dt==real(dt));
125
126 id1 = find(t>0,1,'last');
127 Th_vac = Th_vac(1:id1);
128 Th_amb = Th_amb(1:id1);
129 P_c = P_c(1:id1);
130 m_dot = (A_t/c_star)*P_c;
```

```
dt = dt(1:id1);
plot(t(1:id1),Th_vac(1:id1),'Linewidth',1.5);
hold on
plot(t(1:id1),Th_amb(1:id1),'Linewidth',1.5);
grid on

%Findind tb
Th_temp_vac = Th_vac(200:id1);
t_temp_vac = t(200:id1);
[~,id2] = max(Th_temp_vac);
tb_vac = t_temp_vac(id2);

Th_temp_amb = Th_amb(200:id1);
t_temp_amb = t(200:id1);
[~,id3] = max(Th_temp_amb);
tb_amb = t_temp_amb(id3);

end
```

A.3 Solid Propulsion Optimization using PSO

```
%% ------- Particle Swarm Optimization -------%%

%problem settings
%% Boundaries of Design Variables from Literature
lb=[5,0.2,0.3,0.01,0.01,4.3146] ; %Lower Bound
ub=[15,0.9,0.7,0.15,0.5,5];  %Upper Bound

fun = @func;                    %Fitness Function
%ALgorithm Parameters
w_max=1.2;
w_min=0.1;                      %Linearly Decreasing Inertia
    Weight
c1=2;                           %self acceleration coefficient
c2=2;                           %social acceleration coefficient
```

```
14 Np=100;                          %Size of population
15 T=100;                           %Number of iterations
16
17 %% Algorithm %%
18 %Initialization
19 %Creating random population
20 f = zeros(Np,1);%Array to store fitness values of population
21 D = length(lb);%No.of variables in each particle of
       population
22
23 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);%
       position of Initial Random Population
24 %P = gpuArray(P);
25 v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); %
       velocity of Initial Random Population
26 %v = gpuArray(v);
27
28 for p = 1:Np
29     f(p) = funct(P(p,:),1);   %Evaluating fitness function
          for the initial population
30 end
31
32 pbest = P;  %Initialize the personal best solutions
33 f_pbest = f;%Initialize the fitness of the personal best
       solutions
34
35 [f_gbest, ind] = min(f_pbest);  %Determining the fitness
       function for global best solution
36 gbest = P(ind,:);                %Determining the global best
        solution
37
38 %Iteration Loop
39 for i=1:T
40     w=w_max - (w_max - w_min)*i/T;  %Linearly Varying
          Inertial Weight
```

```
41      for p=1:Np
42          v(p,:) = 0.73*(w*v(p,:) + c1*rand(1,D).*(pbest(p,:)
              - P(p,:)) + c2*rand(1, D).*(gbest - P(p,:))); %
              Calculate the Velocity
43          P(p,:) = P(p,:) + v(p,:);                       %
              Update the position
44
45          P(p,:) = max(P(p,:), lb);                       %
              Bounding the violating variables to their lower
              bound
46          P(p,:) = min(P(p,:), ub);                       %
              Bounding the violating variables to their upper
              bound
47
48          f(p) = funct(P(p,:),i);                          %
              Determining the fitness of new solution
49
50          if f(p) < f_pbest(p)
51              f_pbest(p) = f(p);                          %
                  Updating the fitness value of the personal
                  best value
52              pbest(p,:) = P(p,:);                        %
                  Updating the personal best position value
53
54              if f_pbest(p) < f_gbest
55                  f_gbest = f_pbest(p);                   %
                      Updating the global best fitness value
                      value
56                  gbest = pbest(p,:);                     %
                      Updating the global best position
57              end
58
59          end
60
61      end
```

```
disp(i);
disp(f_gbest);
end


%% Objective (or) Fitness Function
function y=funct(x,k)
%Non-Stationary, Adaptive Exterior Penalty Function%
        [P_c,Th,~,~,~,dt,tb,~] = solid_prop(round(x(1)),x(2)
            ,x(3),x(4),x(5),x(6));
        c1 = max(P_c)/(95*10^5) - 1;
        c2 = tb/150-1;
        obj_func = -(sum(1*Th.*dt))/(10^12);
        q_x = [max(0,c1), max(0,c2)];
        y = 0;
        for i =1:length(q_x)
            if q_x(i) <= 0.001
            y = y + k*sqrt(k)*10*(q_x(i))^(1);
            elseif q_x(i) <= 0.1
            y = y + k*sqrt(k)*20*(q_x(i))^(1);
            elseif q_x(i) <= 1
            y = y + k*sqrt(k)*100*(q_x(i))^(1);
            else
            y = y + k*sqrt(k)*300*(q_x(i))^(2);
            end
        end
        y = y + 1*obj_func;
end
```

A.4 Solid Propulsion Optimization using fmincon

```
%% -------- Sequential Quadratic Programming -------- %%

%Problem Settings
fun = @func;
```

```
5  nonlcon=@cons;
6
7  %% Boundaries of Design Variables from Literature
8  lb=[3,0.2,0.3,0.01,0.01,4.3146];                    %Lower
       Bound
9  ub=[15,0.9,0.7,0.15,0.5,5];                         %Upper Bound
10
11
12 %% Initial Guesses from Particle Swarm Optimization
13 %x0
14
15 %% Sequential Quadratic Programming using fmincon using the
       above initial guesses
16 opts = optimoptions(@fmincon,'Algorithm','sqp','Display','
       iter','OptimalityTolerance',1e-3,'StepTolerance',1e-20,'
       PlotFcn','optimplotfval','MaxFunctionEvaluations',1500);
17 [x,fval] = fmincon(fun,x0,[],[],[],[],lb,ub,nonlcon,opts);
18
19
20 %% Objective Function and Constraints
21 function y=func(x)
22         %Re = 6378165.857;
23         [~,Th,~,~,dt,~] = solid_prop(round(x(1)),x(2),x(3),x
               (4),x(5),x(6));
24         y = -sum(Th.*dt)/(10^10);
25 end
26
27 function [c,ceq]=cons(x)
28         [P_c,~,~,~,~,~] = solid_prop(round(x(1)),x(2),x(3),x
               (4),x(5),x(6));
29         c1 = max(P_c)/(95*10^5) - 1;
30         c=c1;
31         ceq = [];
32 end
```

A.5 Two Dimensional Trajectory Analysis

```
function [ALT,FPA,VEL,t,Qmax,inc,lat_F,long_F,t_index,steer]
    = Trajectory_2d(x)      %Uncomment and use this line when
    using this function for getting output

%function [ob,c1,c2,c3,c4,c5] = Trajectory_2d(x)
                            %Uncomment and use this line when
   using PSO and DE
%function [ob,c,ceq] = Trajectory_2d(x)
                                    %Uncomment and use this
   line when using SQP
%Desired Orbit

%For MDF
%function [ob,c1,c2,c3,c4,c5] = Trajectory_2d(x,t_s1,
   m_dot_s1,Th_vac_s1,tb,lbyd) %Uncomment this line when
   using MDf method


ro=[500 500]*10^3;
                                                          %
   Circular Orbit of radius 500 km
AzL = 180;  %For Polar Orbit
                                                  %Launch
   Azimuth(degrees)
%AzL = 140;  %For Dog-Leg Maneuver
 %Vehicle Parameters
% Vehicle parameters
m_pl = 500;

   %Payload mass in kg
Isp = [284.7784;284.7784;310];
                                                       %Isp
```

```
    of stages in sec
eps = [0.12;0.12;0.15];%Structural factors of stages
T_by_W = [1.4;1.4;0.6];%Thrust-to-weight ratio of all stages
dia = 2;                %Diameter of Launch Vehicle in m
area = pi*(dia/2)^2;    %Cross sectional area of Launch
    Vehicle in m2
n=3;

%Initial attitude angles
roll = 0;
yaw = 0;
pitch = 0;

% Launch Pad parameters (Kulasekharapattinam)
lat_L = 8.3943;
long_L = 78.0516;
h0 = 26.40;

% Earth Parameters
g0 = 9.80665;

    %[m/s^2] g at Earth sea level
Re = 6378165.857;

    %Equatorial Radius of earth
Rp = 6356783.832;

    %Polar Radius of earth
omega_earth = [0 0 7.2921159*(10^-5)];
                                        %Angular Velocity of
    earth
mu = 3.986004418*(10^14);
                                                %gravity
     parameter
j2 = 1.0823*(10^-3);
```

```
                                                                    %
    gravity parameter
f = (Re-Rp)/Re;

    %Oblateness/Flattening

% Derived quantities
C = g0.*Isp;
apogee = max(ro);
perigee = min(ro);
k_e = (Re/Rp)^2;
PhiL = atand((1-f)^2*tand(lat_L));
Rs = Re/sqrt(1+(k_e-1)*(sind(PhiL))^2);
Ri = Rs+h0;
ri = Ri*[cosd(lat_L)*cosd(long_L) cosd(lat_L)*sind(long_L)
    sind(lat_L)];
vi = cross(omega_earth,ri);
AzL=180+AzL;

%inclination
incl = zeros(7000,1);
incl(1)=acosd(sind(AzL)*cosd(lat_L));

%Vel_req
ve_req=10387.92;
%Launch Vehicle Sizing optimization
[mi, ms, mp, ~, T_m] = lv_staging_opt(eps,Isp,T_by_W,3);
%For Stages 2 and 3
mdot = T_m./(C);
T = T_m';

% Comment this block when using this code in MDF analyisis
%Loading Propulsion Data for Stage 1
load('stage_1_dataset7.mat')
%Finding Burn Time
```

```
71 Th_temp_vac = Th_vac_s1(200:length(t_s1));
72 t_temp_vac = t_s1(200:length(t_s1));
73 [~,ids2] = max(Th_temp_vac);
74 tb = t_temp_vac(ids2);
75
76
77 %Loading cd vs mach data
78 load('cd_vs_machdata')
79
80
81
82
83 %Transformation Matrices
84 %1) Inertial Frame to Launch Frame
85 IL(1,:) = [cosd(lat_L)*cosd(long_L) cosd(lat_L)*sind(long_L)
       sind(lat_L)];
86 IL(2,:) = [sind(lat_L)*cosd(long_L)*sind(AzL)-cosd(AzL)*sind
    (long_L) cosd(AzL)*cosd(long_L)+sind(AzL)*sind(lat_L)*
    sind(long_L) -sind(AzL)*cosd(lat_L)];
87 IL(3,:) = [-sind(AzL)*sind(long_L)-cosd(AzL)*sind(lat_L)*
    cosd(long_L) sind(AzL)*cosd(long_L)-cosd(AzL)*sind(lat_L)
    *sind(long_L) cosd(AzL)*cosd(lat_L)];
88
89 %2) Launch Frame to Body Frame
90 LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(pitch
    )+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*cosd(pitch)
    -cosd(roll)*sind(pitch)];
91 LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(yaw)
    ];
92 LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(pitch
    )-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*sind(pitch)
    +cosd(roll)*cosd(pitch)];
93 LB = [LB1;LB2;LB3];
94 IB = LB*IL;
95
```

```
96  [t,m,vm,vrm,am,Q,~,mprop,mach,h,gammarel,azi,theta,lat_F,
        long_F,aoa,gamma,thetarot,theta_alt]=deal(zeros(7000,1));
97  [a,v,vr,vrbf,r,g,steer]=deal(zeros(7000,3));
98  vrg = zeros(3,7000);
99
100 % Base conditions (@t=0)
101 v(1,:) = [vi(1) vi(2) vi(3)];
102 r(1,:) = [ri(1) ri(2) ri(3)];
103 h(1) = norm(r(1,:))-Rs;
104 %Lat_flight(1) = lat_L;
105 %Long_flight(1) = long_L;
106 t(1) = 0;
107 m(1) = mi(1)+7622.19;
108 mprop(1) = mp(1)+7622.19;
109 gammarel(1) = 90;
110 lat_F(1) = lat_L;
111 long_F(1) = long_L;
112
113
114 iter = 2;
115 delt=0.1;
116
117 %% Vertical takeoff phase
118 while h(iter-1)<50
119     if h(iter-1)<1.782853443358382e+05
120         [~, a_amb, P_amb, rho_amb]=atmoscoesa(h(iter-1),'
              None');
121     elseif h(iter-1)<1.882853443358382e+05
122         [~, a_amb,P_amb, rho_amb]=atmoscoesa
              (1.782853443358382e+05,'None');
123     else
124         a_amb = 0;
125         rho_amb = 0;
126     end
127
```

```
128     t(iter) = t(iter-1) + delt;
129     m(iter) = m(iter-1)-interp1(t_s1,m_dot_s1,t(iter))*delt;
130     mprop(iter) = mprop(iter-1)-interp1(t_s1,m_dot_s1,t(iter
            ))*delt;
131     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
132     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
            ,area,cd_data,mach_data);
133     atb = ([(interp1(t_s1,Th_vac_s1,t(iter))-P_amb*area) 0
            0]-D)./m(iter,:);
134     a(iter,:) = (IB\atb')'+g(iter,:);
135     am(iter,:) = norm(a(iter,:));
136     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
137     vr(iter,:) = v(iter,:)-v(1,:);
138     vm(iter) = norm(v(iter,:));
139     vrm(iter) = norm(vr(iter,:));
140     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
141     h(iter) = norm(r(iter,:))- Rs;
142
143     vrbf(iter,:) = IB*vr(iter,:)';
144     aoa(iter)=atan2d(vrbf(iter,3),vrbf(iter,1));
145     pc=asin(r(iter,3)/norm(r(iter,:))); %Geocentric latitude
146     ti=atan(r(iter,2)/r(iter,1)); %Inertial longitude
147     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
148           -sin(ti),cos(ti),0;
149           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
150     vrg(:,iter) = (IG*vr(iter,:)');
151     gamma(iter)=90;
152     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
            ,:))*vrm(iter)));
153     pitch=0;
154     theta(iter) = aoa(iter)+gamma(iter);
155     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
156     steer(iter,:) = [roll yaw pitch];
157     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
            ,2)^2));
```

```
158     long_F(iter) = atan2d(r(iter,2),r(iter,1));
159     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
            -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
            -1,2)/1000,vr(iter-1,3)/1000);
160
161     iter = iter+1;
162 end
163
164 id1 = find(t>0,1,'last');
165 %% Constant turnover phase
166
167 while t(iter-1)<12.5
168     pitch = pitch+(x(21)*delt);
169     yaw = yaw+(x(22)*delt);
170     t(iter) = t(iter-1)+delt;
171     if h(iter-1)<1.782853443358382e+05
172         [~, a_amb, P_amb, rho_amb]=atmoscoesa(h(iter-1),'
                None');
173     elseif h(iter-1)<1.882853443358382e+05
174         [~, a_amb,P_amb, rho_amb]=atmoscoesa
                (1.782853443358382e+05,'None');
175     else
176         a_amb = 0;
177         rho_amb = 0;
178     end
179     m(iter) = m(iter-1)-interp1(t_s1,m_dot_s1,t(iter))*delt;
180     mprop(iter) = mprop(iter-1)-interp1(t_s1,m_dot_s1,t(iter
            ))*delt;
181     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
182     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
            ,area,cd_data,mach_data);
183     atb = ([(interp1(t_s1,Th_vac_s1,t(iter))-P_amb*area) 0
            0]-D)/m(iter,:);
184     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
            pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
```

```
    cosd(pitch)-cosd(roll)*sind(pitch)];
185 LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
    yaw)];
186 LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
    pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
    sind(pitch)+cosd(roll)*cosd(pitch)];
187 LB = [LB1;LB2;LB3];
188 IB = LB*IL;
189 logi = isnan(IB);
190 if ismember(1,logi)
191     ob = Inf;
192     c1 = Inf;
193     c2 = Inf;
194     c3 = Inf;
195     c4 = Inf;
196     c5 = Inf;
197     return
198 end
199 a(iter,:) = (IB\atb')'+g(iter,:);
200 am(iter,:) = norm(a(iter,:));
201 v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
202 vr(iter,:) = v(iter,:)-v(1,:);
203 vm(iter) = norm(v(iter,:));
204 vrm(iter) = norm(vr(iter,:));
205 r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
206
207 vrbf(iter,:) = IB*vr(iter,:)';
208 aoa(iter)=atan2d(vrbf(iter,3),vrbf(iter,1));
209
210 pc=asin(r(iter,3)/norm(r(iter,:))); %Geocentric latitude
211 ti=atan(r(iter,2)/r(iter,1)); %Inertial longitude
212 IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
213       -sin(ti),cos(ti),0;
214       -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
215 vrg(:,iter) = (IG*vr(iter,:)');
```

```
216     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
217     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
            ,:))*vrm(iter)));
218     thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
            norm(ri))));
219
220     theta(iter)=aoa(iter)+gamma(iter)+thetarot(iter);
221     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
222     h(iter) = norm(r(iter,:))-Rs;
223     steer(iter,:) = [roll yaw pitch];
224     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
            ,2)^2));
225     long_F(iter) = atan2d(r(iter,2),r(iter,1));
226     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
            -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
            -1,2)/1000,vr(iter-1,3)/1000);
227
228
229     iter=iter+1;
230 end
231
232 id2 = find(t>0,1,'last');
233
234 %% Gravity turn phase
235 while t(iter-1) < tb
236     t(iter) = t(iter-1)+delt;
237     if h(iter-1)<1.782853443358382e+05
238         [~, a_amb,P_amb, rho_amb]=atmoscoesa(h(iter-1),'None
                ');
239     elseif h(iter-1)<1.882853443358382e+05
240         [~, a_amb,P_amb, rho_amb]=atmoscoesa
                (1.782853443358382e+05,'None');
241     else
242         a_amb = 0;
243         rho_amb = 0;
```

```
244     end
245     m(iter) = m(iter-1)-interp1(t_s1,m_dot_s1,t(iter))*delt;
246     mprop(iter) = mprop(iter-1)-interp1(t_s1,m_dot_s1,t(iter
            ))*delt;
247     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
248     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
            ,area,cd_data,mach_data);
249     atb = ([(interp1(t_s1,Th_vac_s1,t(iter))-P_amb*area) 0
            0]-D)/m(iter,:);
250     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
            pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
            cosd(pitch)-cosd(roll)*sind(pitch)];
251     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
            yaw)];
252     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
            pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
            sind(pitch)+cosd(roll)*cosd(pitch)];
253     LB = [LB1;LB2;LB3];
254     IB = LB*IL;
255     logi = isnan(IB);
256     if ismember(1,logi)
257         ob = Inf;
258         c1 = Inf;
259         c2 = Inf;
260         c3 = Inf;
261         c4 = Inf;
262         c5 = Inf;
263         return
264     end
265     a(iter,:) = (IB\atb')'+g(iter,:);
266     am(iter,:) = norm(a(iter,:));
267     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
268     vr(iter,:) = v(iter,:)-v(1,:);
269     vm(iter) = norm(v(iter,:));
270     vrm(iter) = norm(vr(iter,:));
```

```
271     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
272
273     aoa(iter)=0;
274     pc=asin(r(iter,3)/norm(r(iter,:))); %Geocentric latitude
275     ti=atan(r(iter,2)/r(iter,1)); %Inertial longitude
276
277     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
278           -sin(ti),cos(ti),0;
279           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
280     vrg(:,iter) = (IG*vr(iter,:)');
281     lam=atan2d(vrg(2,iter),vrg(1,iter));
282     gam=asind(r(iter,:)*vrg(:,iter)/(norm(r(iter,:))*norm(
          vrg(:,iter))));
283
284     %gamma(iter)=gravity_turn(gam,lam,g(iter,:),atb,delt,v
          (1,:),IG,r(iter-1,:),v(iter-1,:));
285     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
          ,:))*vrm(iter)));
286
287     thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
          norm(ri))));
288     %theta(iter)=aoa(iter)+gamma(iter)+thetarot(iter);
289     theta_alt(iter)=pitch+(sind(gammarel(iter))*(norm(g(iter
          ,:))/vrm(iter)))*delt;
290     pitch=theta_alt(iter);
291     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
292     h(iter) = norm(r(iter,:))-Rs;
293     steer(iter,:) = [roll yaw pitch];
294     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
          ,2)^2));
295     long_F(iter) = atan2d(r(iter,2),r(iter,1));
296     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
          -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
          -1,2)/1000,vr(iter-1,3)/1000);
297
```

```
298
299     t(iter) = t(iter-1)+delt;
300     iter=iter+1;
301 end
302
303 id3 = find(t>0,1,'last');
304
305 %% First Stage Separation
306 while t(iter-1)<=t(id3)+8
307     t(iter) = t(iter-1)+delt;
308     if h(iter-1)<1.782853443358382e+05
309         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
310     elseif h(iter-1)<1.882853443358382e+05
311         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
              +05,'None');
312     else
313         a_amb = 0;
314         rho_amb = 0;
315     end
316     m(iter) = mi(1,1)-mp(1,1)-ms(1,1);
317     mprop(iter) = mp(2,1);
318     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
319     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
          ,area,cd_data,mach_data);
320     atb = -D/m(iter,:);
321     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
          pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
          cosd(pitch)-cosd(roll)*sind(pitch)];
322     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
          yaw)];
323     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
          pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
          sind(pitch)+cosd(roll)*cosd(pitch)];
324     LB = [LB1;LB2;LB3];
325     IB = LB*IL;
```

```
326     logi = isnan(IB);
327     if ismember(1,logi)
328         ob = Inf;
329         c1 = Inf;
330         c2 = Inf;
331         c3 = Inf;
332         c4 = Inf;
333         c5 = Inf;
334         return
335     end
336     a(iter,:) = (IB\atb')'+g(iter,:);
337     am(iter,:) = norm(a(iter,:));
338     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
339     vr(iter,:) = v(iter,:)-v(1,:);
340     vm(iter) = norm(v(iter,:));
341     vrm(iter) = norm(vr(iter,:));
342     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
343
344     aoa(iter)=0;
345     pc=asin(r(iter,3)/norm(r(iter,:))); %Geocentric latitude
346     ti=atan(r(iter,2)/r(iter,1)); %Inertial longitude
347     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
348           -sin(ti),cos(ti),0;
349           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
350     vrg(:,iter) = (IG*vr(iter,:)');
351     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
352     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
        ,:))*vrm(iter)));
353     thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
        norm(ri))));
354     theta(iter)=aoa(iter)+gammarel(iter)+thetarot(iter);
355     pitch=90-theta(iter);
356
357     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
358     h(iter) = norm(r(iter,:))-Rs;
```

```
359       steer(iter,:) = [roll yaw pitch];
360       lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
              ,2)^2));
361       long_F(iter) = atan2d(r(iter,2),r(iter,1));
362       [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
              -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
              -1,2)/1000,vr(iter-1,3)/1000);
363
364
365       t(iter) = t(iter-1)+delt;
366       iter=iter+1;
367   end
368
369   id4 = find(t>0,1,'last');
370
371
372
373
374   id5=id4;
375   k1=iter;
376   %% Stage-2 Optimal Trajectory
377   time = fix((mprop(iter-1)-mp(2)*0.002)/mdot(2));
378
379   %Finding pitch rates at each time point
380   n_pitch=10;

      %Number of segments
381   PRval= x(1:10);
382   PRates = find_rates(time,n_pitch,delt,PRval);
383
384   %Finding Yaw rate at each time point
385   %n_yaw = 5;

      %No.of Segmets
386   %YRval = x(23:27);
```

```
387 %YRates = find_rates(time,n_yaw,delt,YRval);
388
389 k2=1;
390
391 while iter<k1+size(PRates,2)
392     t(iter) = t(iter-1)+delt;
393
394     %% For IIP constraint
395     %if long_F_impact(iter-1)<=83 %&& lat_F_impact <= 9.51
396     %    yaw=yaw + (YRates(k2)*delt);
397     %end
398     %% For Dog-Leg Maneuver
399     %if steer(iter-1,2)-steer(id5,2)<=45
400     %    yaw=yaw + (YRates(k2)*delt);
401     %end
402
403     %if abs(steer(iter-1,2)-steer(id5,2))<=45
404     %    yaw = yaw + (Yawrates3fin(1,k3)*delt);
405     %end
406
407
408     %if (steer(id5,2) - steer(iter-1,2)) <= 70 && lat_F(iter
        -1) <= 12
409     %yaw = yaw + (YRates(k2)*delt);
410     %end
411
412     if h(iter-1)<1.782853443358382e+05
413         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
414     elseif h(iter-1)<1.882853443358382e+05
415         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
            +05,'None');
416     else
417         a_amb = 0;
418         rho_amb = 0;
419     end
```

```
420     m(iter) = m(iter-1)-mdot(2)*delt;
421     mprop(iter) = mprop(iter-1)-mdot(2)*delt;
422     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
423     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
            ,area,cd_data,mach_data);
424     atb = ([T(1,2) 0 0]-D)/m(iter,:);
425     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
            pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
            cosd(pitch)-cosd(roll)*sind(pitch)];
426     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
            yaw)];
427     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
            pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
            sind(pitch)+cosd(roll)*cosd(pitch)];
428     LB = [LB1;LB2;LB3];
429     IB = LB*IL;
430     logi = isnan(IB);
431     if ismember(1,logi)
432         ob = Inf;
433         c1 = Inf;
434         c2 = Inf;
435         c3 = Inf;
436         c4 = Inf;
437         c5 = Inf;
438         return
439     end
440     a(iter,:) = (IB\atb')'+g(iter,:);
441     am(iter,:) = norm(a(iter,:));
442     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
443     vr(iter,:) = v(iter,:)-v(1,:);
444     vm(iter) = norm(v(iter,:));
445     vrm(iter) = norm(vr(iter,:));
446     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
447
448
```

```
449
450     pc=asin(r(iter,3)/norm(r(iter,:)));
                                                %Geocentric
        latitude
451     ti=atan(r(iter,2)/r(iter,1));
                                                    %Inertial
        longitude
452     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
453           -sin(ti),cos(ti),0;
454           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
455     vrg(:,iter) = (IG*vr(iter,:)');
456     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
457     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
        ,:))*vrm(iter)));
458     theta(iter) = pitch+PRates(k2)*delt;
459     pitch=theta(iter);
460     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
461    %aoa(iter)=theta(iter)-gamma(iter);
462
463     vrbf(iter,:) = IB*vr(iter,:)';
464     aoa(iter)=atan2d(vrbf(iter,3),vrbf(iter,1));
465
466
467     h(iter) = norm(r(iter,:))-Rs;
468     steer(iter,:) = [roll yaw pitch];
469     k2=k2+1;
470     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
        ,2)^2));
471     long_F(iter) = atand(r(iter,2)/r(iter,1));
472    %[~,long_F_impact(iter),lat_F_impact(iter)] =
        impact_point(r(iter,:),vr(iter,:));
473     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
        -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
        -1,2)/1000,vr(iter-1,3)/1000);
474
```

```
475
476     iter=iter+1;
477 end
478 id6 = find(t>0,1,'last');
479
480  %% Second Stage Separation
481 while t(iter-1)<=t(id6)+x(23)
482     t(iter) = t(iter-1)+delt;
483     if h(iter-1)<1.782853443358382e+05
484         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
485     elseif h(iter-1)<1.882853443358382e+05
486         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
            +05,'None');
487     else
488         a_amb = 0;
489         rho_amb = 0;
490     end
491     m(iter) = mi(1)-mp(1)-ms(1)-mp(2)-ms(2);
492     mprop(iter) = mp(3);
493     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
494     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
        ,area,cd_data,mach_data);
495     atb = -D/m(iter,:);
496     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
        pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
        cosd(pitch)-cosd(roll)*sind(pitch)];
497     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
        yaw)];
498     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
        pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
        sind(pitch)+cosd(roll)*cosd(pitch)];
499     LB = [LB1;LB2;LB3];
500     IB = LB*IL;
501     logi = isnan(IB);
502     if ismember(1,logi)
```

```
503             ob = Inf;
504             c1 = Inf;
505             c2 = Inf;
506             c3 = Inf;
507             c4 = Inf;
508             c5 = Inf;
509             return
510         end
511         a(iter,:) = (IB\atb')'+g(iter,:);
512         am(iter,:) = norm(a(iter,:));
513         v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
514         vr(iter,:) = v(iter,:)-v(1,:);
515         vm(iter) = norm(v(iter,:));
516         vrm(iter) = norm(vr(iter,:));
517         r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
518
519         aoa(iter) = aoa(iter-1);
520         pc=asin(r(iter,3)/norm(r(iter,:)));
                                                %Geocentric
            latitude
521         ti=atan(r(iter,2)/r(iter,1));
                                                    %Inertial
            longitude
522         IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
523               -sin(ti),cos(ti),0;
524               -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
525         vrg(:,iter) = (IG*vr(iter,:)');
526
527         gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
528         gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
            ,:))*vrm(iter)));
529         thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
            norm(ri))));
530         theta(iter)=aoa(iter)+gamma(iter)+thetarot(iter);
531         pitch=theta(iter);
```

```
532       azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
533       h(iter) = norm(r(iter,:))-Rs;
534       steer(iter,:) = [roll yaw pitch];
535       lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
              ,2)^2));
536       long_F(iter) = atan2d(r(iter,2),r(iter,1));
537       [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
              -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
              -1,2)/1000,vr(iter-1,3)/1000);
538
539
540       t(iter) = t(iter-1)+delt;
541       iter=iter+1;
542   end
543
544   id7 = find(t>0,1,'last');
545
546   %[~,long_F_impact(iter-1),~] = impact_point(r(id4,:),vr(id4
          ,:));
547
548   k2=iter;
549
550   %% STAGE 3 Optimal trajectory
551   m(iter-1)=mi(3);
552   mprop(iter-1)=mp(3);
553   time3 = fix((mprop(iter-1)-mp(3)*0.002)/mdot(3));
554
555   n3=10;

        %Number of segments
556   PRval3= x(11:20);
557   PRates3 = find_rates(time3,n3,delt,PRval3);
558
559   %Uncomment when using IIP as a constraint, comment when
          using Dog-Leg Maneuver
```

```
560 %For optimizing yaw-rates
561 %ndash3=5;
562 %Yawrates3 = x(28:32);
                                                                    %
    Number of segments
563 %Yawrates3fin = find_rates(time3,ndash3,delt,Yawrates3);
564
565 k3=1;
566 while iter<k2+size(PRates3,2)
567     t(iter) = t(iter-1)+delt;
568
569     %Uncomment when using IIP as a constraint, comment when
        using Dog-Leg Maneuver
570     %if long_F_impact(iter-1) >= long_L %&& lat_F_impact(
        iter-1)<=7.53
571     %if  long_F_impact(iter-1) <= 83 && long_F_impact(iter
        -1) >= 82
572     %    yaw = yaw + (Yawrates3fin(1,k3)*delt);
573     %end
574
575     %if long_F_impact(iter-1) <= long_L && (steer(id7,2) -
        steer(iter-1,2)) <= 45
576     %if (steer(id7,2) - steer(iter-1,2)) <= 20 %&& iter <
        4680
577     %yaw = yaw + (Yawrates3fin(1,k3)*delt);
578     %end
579
580
581     if h(iter-1)<1.782853443358382e+05
582         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
583     elseif h(iter-1)<1.882853443358382e+05
584         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
            +05,'None');
585     else
586         a_amb = 0;
```

```
587             rho_amb = 0;
588         end
589         m(iter) = m(iter-1)-mdot(3)*delt;
590         mprop(iter) = mprop(iter-1)-mdot(3)*delt;
591         g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
592         [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
                ,area,cd_data,mach_data);
593         atb = ([T(1,3) 0 0]-D)/m(iter,:);
594         LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
                pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
                cosd(pitch)-cosd(roll)*sind(pitch)];
595         LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
                yaw)];
596         LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
                pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
                sind(pitch)+cosd(roll)*cosd(pitch)];
597         LB = [LB1;LB2;LB3];
598         IB = LB*IL;
599         logi = isnan(IB);
600         if ismember(1,logi)
601             ob = Inf;
602             c1 = Inf;
603             c2 = Inf;
604             c3 = Inf;
605             c4 = Inf;
606             c5 = Inf;
607             return
608         end
609         a(iter,:) = (IB\atb')'+g(iter,:);
610         am(iter,:) = norm(a(iter,:));
611         v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
612         vr(iter,:) = v(iter,:)-v(1,:);
613         vm(iter) = norm(v(iter,:));
614         vrm(iter) = norm(vr(iter,:));
615         r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
```

```
616
617     pc=asin(r(iter,3)/norm(r(iter,:)));
                                              %Geocentric
        latitude
618     ti=atan(r(iter,2)/r(iter,1));
                                                  %Inertial
        longitude
619     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
620           -sin(ti),cos(ti),0;
621           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
622     vrg(:,iter) = (IG*vr(iter,:)');
623     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
624     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
        ,:))*vrm(iter)));
625     theta(iter) = pitch+PRates3(k3)*delt;
626     pitch=theta(iter);
627
628     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
629     aoa(iter)=theta(iter)-gamma(iter);
630
631     vrbf(iter,:) = IB*vr(iter,:)';
632     aoa(iter)=atan2d(vrbf(iter,3),vrbf(iter,1));
633
634     h(iter) = norm(r(iter,:))-Rs;
635     steer(iter,:) = [roll yaw pitch];
636     k3=k3+1;
637     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
        ,2)^2));
638     long_F(iter) = atan2d(r(iter,2),r(iter,1));
639     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
        -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
        -1,2)/1000,vr(iter-1,3)/1000);
640
641
642    %[~,long_F_impact(iter),~] = impact_point(r(iter,:),vr(
```

```
        iter,:));
643
644     iter=iter+1;
645 end
646 id8 = find(t>0,1,'last');
647
648 %% Stage 3 Separation
649 while t(iter-1)<=t(id8)+8
650     t(iter) = t(iter-1)+delt;
651     if h(iter-1)<1.782853443358382e+05
652         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
653     elseif h(iter-1)<1.882853443358382e+05
654         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
            +05,'None');
655     else
656         a_amb = 0;
657         rho_amb = 0;
658     end
659     m(iter) = mi(1,1)-mp(1)-ms(1)-mp(2)-ms(2)-mp(3)-ms(3);
660     mprop(iter) = 0;
661     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
662     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
        ,area,cd_data,mach_data);
663     atb = -D/m(iter,:);
664     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
        pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
        cosd(pitch)-cosd(roll)*sind(pitch)];
665     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
        yaw)];
666     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
        pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
        sind(pitch)+cosd(roll)*cosd(pitch)];
667     LB = [LB1;LB2;LB3];
668     IB = LB*IL;
669     logi = isnan(IB);
```

```
670     if ismember(1,logi)
671         ob = Inf;
672         c1 = Inf;
673         c2 = Inf;
674         c3 = Inf;
675         c4 = Inf;
676         c5 = Inf;
677         return
678     end
679     a(iter,:) = (IB\atb')'+g(iter,:);
680     am(iter,:) = norm(a(iter,:));
681     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
682     vr(iter,:) = v(iter,:)-v(1,:);
683     vm(iter) = norm(v(iter,:));
684     vrm(iter) = norm(vr(iter,:));
685     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
686
687     aoa(iter) = aoa(iter-1);
688     pc=asin(r(iter,3)/norm(r(iter,:)));
                                              %Geocentric
        latitude
689     ti=atan(r(iter,2)/r(iter,1));
                                                  %Inertial
        longitude
690     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
691           -sin(ti),cos(ti),0;
692           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
693     vrg(:,iter) = (IG*vr(iter,:)');
694     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
695     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
        ,:))*vrm(iter)));
696     thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
        norm(ri))));
697     theta(iter)=aoa(iter)+gamma(iter)+thetarot(iter);
698     pitch=theta(iter);
```

```
699       azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
700       h(iter) = norm(r(iter,:))-Rs;
701       steer(iter,:) = [roll yaw pitch];
702       lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
              ,2)^2));
703       long_F(iter) = atan2d(r(iter,2),r(iter,1));
704       [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
              -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
              -1,2)/1000,vr(iter-1,3)/1000);
705
706
707       t(iter) = t(iter-1)+delt;
708       iter=iter+1;
709   end
710   id9 = find(t>0,1,'last');
711
712
713   %% For Output
714   %(Uncomment this block and comment the below blocks for
          running this
715   %script as function for getting outputs)
716
717   %The outputs returned by the function
718   [~,~,inc,~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter-1,2)/1000,r(
          iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter-1,2)/1000,vr(
          iter-1,3)/1000);
719   ALT=h;
720   Qmax=Q;
721   FPA=gammarel;
722   VEL=vm;
723   t_index=[id1,id2,id3,id4,id5,id6,id7,id8,id9];
724
725
726   %% 0) For Optimization
727   %(Uncomment this block and comment the above and below
```

```
    blocks for running this
%script as function for getting constraints and objective
    functions for PSO,DE)

% Comment the above block and uncomment the below blocks for
    Optimization
%[~,~,inc,~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter-1,2)/1000,r
    (iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter-1,2)/1000,vr(
    iter-1,3)/1000);
%ALT=h(iter-1);
%Qmax=max(Q);
%FPA=gammarel(iter-1);
%VEL=vm(iter-1);

%% 1) For Particle swarm optimization, Differential
    Evolution
%ob = -ALT/(500*10^3);
%ob = -VEL/(1000*sqrt(398600/(ALT/1000+Re/1000)));
%c1 = abs(VEL/(1000*sqrt(398600/(ALT/1000+Re/1000))) - 1) -
    10/(1000*sqrt(398600/(ALT/1000+Re/1000)));
%c1 = -(VEL/(1000*sqrt(398600/(ALT/1000+Re/1000))) - 1);
%c2 = abs(FPA/90) - (2/90);
%c3 = Qmax/100000 - 0.55;
%c4 = -(inc/90 - 1);
%c5 = -1*(ALT/(500*10^3)-1);
%c5 = -(steer(id5,2)/90 - steer(id9,2)/90) + 1/8;
%c5 = (long_F_impact(iter-1)/90 - long_L/90);
%c5 = steer(id9,2)/90 + 90/90;
%c6 = (long_F(iter-1)/90 - long_L/90);
%c4 = -inc/98 + 1;
%c6 = (- long_F_impact(id5)/90 + 82/90)^2; %+ (lat_F_impact(
    id5)/90 - 7.53/90)^2 - 1/(90^2);
%c6 = -long_F_impact(id5)/90 + 82/90;
%c6 = 0;
%for i = id5:id9
```

```
%    if lat_F_impact(i) >=5.55 && lat_F_impact(i) <= 9.51
%     c6 = c6 + (1 - (long_F_impact(i)-80.47)^2/132^2 - (
    lat_F_impact(i) - 7.53)^2/167^2);
%    elseif lat_F_impact >9.51
%    c6 = c6 + (long_F_impact(i)/90 - long_L/90)^2;
%    end
%end
%emp = find(lat_F_impact-6<0,1,'first');
%c6 = -long_F_impact(emp)/90 + 82/90;

%% 2) For Sequential Quagratic Programming
%(Uncomment this block and the above two below blocks for
    running this
%script as function for getting constraints and objective
    functions for fmincon)
%c=[c1,c2, c3,c5];
%ceq = [c4];
```

A.6 Auxiliary Functions used in Two Dimensional Trajectroy

A.6.1 Drag Function

```
function [D, Q, mach]=drag(v, a, rho, area,cd_data,mach_data
    )
mach=norm(v)/a;

%Calculated/ANSYS Simulated cd vs mach data
if mach <=0.6
    c_d = 0.3126;
elseif mach<max(mach_data)
    c_d = interp1(mach_data,cd_data,mach);       % import Cd
        as function of Mach
else
    c_d = 0;
```

```
11 end
12
13 %Averaged cd vs mach data of Saturn Launch Vehicle
14 %if mach <=0.6
15 %     c_d = 0.2083333*mach^2 - 0.25*mach + 0.46;
16 %elseif (mach > 6) && (mach <= 0.8)
17 %     c_d = 1.25*(mach)^3 - 2.125*(mach)^2 + 1.2*mach + 0.16;
18 %elseif (mach > 8) && (mach<=0.95)
19 %     c_d = 10.37037*(mach)^3 - 22.88889*(mach)^2 + 16.91111*
    mach - 3.7863;
20 %elseif (mach > 0.95 ) && (mach <= 1.05 )
21 %     c_d = -30*(mach)^3 + 88.5*(mach)^2 - 85.425*(mach) +
    27.51375;
22 %elseif (mach > 1.05) && (mach <= 1.15)
23 %     c_d = -20*(mach)^3 + 60*(mach)^2 -58.065*mach + 19.245;
24 %elseif (mach > 1.15 ) && (mach <= 1.30)
25 %     c_d = 11.85185*(mach)^2 - 44.88889*(mach)^2 +
    56.22222*(mach) - 22.58519;
26 %elseif (mach > 1.3) && (mach <= 2)
27 %     c_d = -0.04373178*(mach)^3 + 0.3236152*(mach)^2 -
    1.019679*mach + 1.544752;
28 %elseif (mach > 2) && (mach <= 3.25)
29 %     c_d = 0.01024*(mach)^3 - 0.00864*(mach)^2 - 0.33832*
    mach + 1.08928;
30 %elseif (mach > 3.25) && (mach <= 4.5)
31 %     c_d = -0.01408*(mach)^3 + 0.19168*(mach) - 0.86976*mach
     + 1.53544;
32 %else
33 %     c_d = 0.22;
34 %end
35
36 Q=0.5*rho*norm(v)^2;
37 D=[c_d*area*Q 0 0];
38
39 end
```

A.6.2 Oblate Spheroid Earth Gravity Model

```
function [g] = gravity(r,j2,Re,mu)
    x = r(1,1);
    y = r(1,2);
    z = r(1,3);
    r = sqrt(x^2 + y^2 + z^2);
    R = Re/r;
    Z = z/r;
    J = (3/2)*j2;
    p = 1+J*(R^2)*(1-5*Z^2);
    gx = -1*mu*x*p/(r^3);
    gy = -1*mu*y*p/(r^3);
    gz = -1*(mu/r^3)*(1+J*R^2*(3-5*Z^2))*z;
    g = [gx,gy,gz];
end
```

A.6.3 Gravity Turn

```
function gamma_gt = gravity_turn(gam,lam,g,atb,delt,v1,IG,
   rprev,vprev)
%alpha, beta and signma values are zeroes in gravity turn
   phase
a=0;b=0;c=0;
%AB matrix can be known from the above values
AB=[cosd(a)*cosd(b) -cosd(a)*sind(b)*cosd(c)+sind(a)*sind(c)
    -cosd(a)*sind(b)*sind(c)-sind(a)*sind(c);
     sind(b) cosd(b)*cosd(c) cosd(b)*sind(c);
     sind(a)*cosd(b) -sind(a)*sind(b)*cosd(c)-cosd(a)*sind(c)
         -sind(a)*sind(b)*sind(c)+cosd(a)*cosd(c)];
%GA matrix can be known from gam, lam
GA=[cosd(gam)*cosd(lam) cosd(gam)*sind(lam) -sind(gam);
     -sind(lam) cosd(lam) 0;
     sind(gam)*cosd(lam) sind(gam)*sind(lam) cosd(gam)];
%Using second definition for calculating IB matrix
IBnew=(IG*GA)*AB;
```

```
a = (IBnew\atb')'+g;
v = vprev+a.*delt;
vr = v-v1;
r = rprev+v.*delt;

pc=asin(r(1,3)/norm(r(1,:))); %Geocentric latitude
ti=atan2(r(1,2),r(1,1)); %Inertial longitude

IGnew = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
            -sin(ti),cos(ti),0;
            -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
%Relative velocity transformed into geographic frame
vrg = (IGnew*vr');
%Now Flight path angle in geographic frame
gamma_gt=(atand(vrg(3,1)/vrg(1,1)));
end
```

A.6.4 function for finding the pitch rates at all instants

```
%Function to find rates at each time point in each segment
   in the trajectory
function rates_at_each_point = find_rates(time, n, delt,
   PRval)
no_of_points = fix((time/n)/delt);
a = zeros(1, no_of_points);
a = a + PRval(1,1);
b = zeros(1, no_of_points);
for i = 2:n
    for j = 1 : no_of_points
        b(j) = PRval(i);
    end
    a = [a b];
end
rates_at_each_point = a;
end
```

A.6.4.1 Function for converting orbital elements to state vector

```
% Orbital Elements to State Vector %
function [r, v] = oe_2_sv(a, e, i, raan, aop, nu)
mu=398600;%km^3/sec^2
%Position In Perifocal frame
xp=a*(1-(norm(e))^2)*cos(nu*pi/180)/(1 + norm(e)*cos(nu*pi
    /180));
yp=a*(1-(norm(e))^2)*sin(nu*pi/180)/(1 + norm(e)*cos(nu*pi
    /180));
zp=0;
rp = [xp yp zp];
%Velocity In Perifocal frame
xdp = -sqrt(mu/(a*(1-(norm(e))^2)))*sin(nu*pi/180);
ydp = sqrt(mu/(a*(1-(norm(e))^2)))*(norm(e) + cos(nu*pi/180)
    );
zdp=0;
vp = [xdp, ydp, zdp];

%Rotation Matrices
R_raan = [cos(raan*pi/180) -sin(raan*pi/180) 0;
          sin(raan*pi/180)  cos(raan*pi/180) 0;
          0 0 1];
R_aop = [cos(aop*pi/180) -sin(aop*pi/180) 0;
         sin(aop*pi/180)  cos(aop*pi/180) 0;
         0 0 1];
R_i = [1 0 0;
       0 cos(i*pi/180) -sin(i*pi/180);
       0 sin(i*pi/180)  cos(i*pi/180)];
R = R_raan*R_i*R_aop;

%Position, Velocity vector in Inertial Frame
r = R*rp';
v = R*vp';
end
```

A.6.4.2 Function for converting state vector to orbital elements

```
%%State vector to Orbital Elements%%
function [a, e, i, raan, aop, nu]=sv_2_oe(x,y,z,xd,yd,zd)%in
    km and km/sec
mu=398600 ;%km^3/sec^2
r=[x, y, z];
v=[xd, yd, zd];
rm = sqrt(x^2 + y^2 +z^2);
vm = sqrt(xd^2 + yd^2 +zd^2);
%Semi Major Axis
a=1/(2/rm - vm^2/mu);
%Eccentricity Vector
for i=1:3
    e(i)= (vm^2/mu - 1/rm)*r(i) - (1/mu)*(dot(r,v))*v(i);
end

I=[1 0 0];
J=[0 1 0];
K=[0 0 1];

%Inclination angle
magw=sqrt((((y*zd)-(z*yd))^2)+(((x*zd)-(z*xd))^2)+(((y*xd)-(
    x*yd))^2));
wzunit=((x*yd)-(y*xd))/magw;
wxunit=((y*zd)-(z*yd))/magw;
wyunit=((z*xd)-(x*zd))/magw;
W = [wxunit, wyunit, wzunit];
i = acos(dot(W, K));
i = i*180/pi;
%Right Ascession of Ascending Node
N = cross(K/norm(K), W/norm(W))/(norm(W)*norm(K));
cos_raan=dot(I, N);
sin_raan=dot(cross(I, N), K);
raan_0 = atan2(sin_raan, cos_raan) ;
```

```
if raan_0 < 0
    raan = 2*pi + raan_0;
else
    raan=raan_0;
end
raan = raan*180/pi;
%Argument of Perigee(AoP)
cos_aop = dot(N, e/norm(e));
sin_aop = dot(cross(N, e/norm(e)), W);
aop_0 = atan2(sin_aop, cos_aop);
if aop_0 < 0
    aop = 2*pi + aop_0;
else
    aop=aop_0;
end
aop = aop*180/pi;
%True Anomaly(nu)
cos_nu = dot(e/norm(e), r/norm(r));
sin_nu = dot(cross(e/norm(e), r/norm(r)), W);
nu_0 = atan2(sin_nu, cos_nu);
if nu_0 < 0
    nu = nu_0 + 2*pi;
else
    nu = nu_0;
end
nu = nu*180/pi;
end
```

A.7 Optimization of Two-Dimensional Trajectory using PSO

```
%% ---------- Particle Swarm Optimization ------------ %%

%problem settings
%% For 2D Trajectory Optimization
\newpage
```

```
lb
   =[-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,
0,0,8];%Lower Bound
ub=[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,15];%Upper
   Bound

fun = @func;          %Fitness Function

%ALgorithm Parameters
w_max=1.2;
w_min=0.1;            %Linearly Decreasing Inertia Weight
c1=2;                 %self acceleration coefficient
c2=2;                 %social acceleration coefficient
Np=100;               %Size of population
T=100;                %Number of iterations

%% Algorithm %%
%Initialization
%Creating random population
f = zeros(Np,1);  %Array to store fitness values of
   population
D = length(lb);   %No.of variables in each particle of
   population

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);%
   position of Initial Random Population

v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); %
   velocity of Initial Random Population

for p = 1:Np
    f(p) = func(P(p,:),1);%Evaluating fitness function for
       the initial population
end
```

```
33
34 pbest = P;       %Initialize the personal best solutions
35 f_pbest = f;     %Initialize the fitness of the personal best
       solutions
36
37 [f_gbest, ind] = min(f_pbest); %Determining the fitness
       function for global best solution
38 gbest = P(ind,:); %Determining the global best solution
39 %f_plot =zeros(T,1);
40
41 %Iteration Loop
42 for i=1:T
43     w=w_max - (w_max - w_min)*i/T; %Linearly Varying
          Inertial Weight
44     for p=1:Np
45         v(p,:) = 0.73*(w*v(p,:) + c1*rand(1,D).*(pbest(p,:)
              - P(p,:)) + c2*rand(1, D).*(gbest - P(p,:))); %
              Calculate the Velocity
46         P(p,:) = P(p,:) + v(p,:);   %Update the position
47
48         P(p,:) = max(P(p,:), lb); %Bounding the violating
              variables to their lower bound
49         P(p,:) = min(P(p,:), ub); %Bounding the violating
              variables to their upper bound
50
51         f(p) = func(P(p,:),i); %Determining the fitness of
              new solution
52
53         if f(p) < f_pbest(p)
54             f_pbest(p) = f(p); %Updating the fitness value
                  of the personal best value
55             pbest(p,:) = P(p,:);%Updating the personal best
                  position value
56
57             if f_pbest(p) < f_gbest
```

```
58                  f_gbest = f_pbest(p); %Updating the global
                        best fitness value value
59                  gbest = pbest(p,:); %Updating the global
                        best position
60              end
61
62          end
63
64      end
65  disp(i);
66  disp(f_gbest);
67  end
68
69
70  %% Objective (or) Fitness Function
71  function y=func(x,k)
72  %Non-Stationary, Adaptive Exterior Penalty Function%
73          %Re = 6378165.857;
74          [obj_func,c1_v,c2_g,c3_Qm,c4_i,c5_alt] =
                Trajectory_2d(x)  ;
75          q_x = [max(0,c1_v), max(0,c2_g), max(0,c3_Qm), max
                (0,c4_i),max(0,c5_alt)];
76          y = 0;
77          for i =1:length(q_x)
78              if q_x(i) <= 0.001
79              y = y + k*sqrt(k)*10*(q_x(i))^(1);
80              elseif q_x(i) <= 0.1
81              y = y + k*sqrt(k)*20*(q_x(i))^(1);
82              elseif q_x(i) <= 1
83              y = y + k*sqrt(k)*100*(q_x(i))^(1);
84              else
85              y = y + k*sqrt(k)*300*(q_x(i))^(2);
86              end
87          end
88          y = y + 1*obj_func;
```

```
89 end
```

A.8 Optimization of Two-Dimensional Trajectory using SQP

```
1  %% ----- Sequential Quadratic Programming -----%%
2  %Problem Settings
3  fun = @func;
4  nonlcon=@cons;
5
6  %% For 2D Trajectory Optimization
7  lb
       =[-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,0,0,8];
                                        %Lower Bound for pitch and
       yaw rates
8  ub=[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,30];
                                                          %Upper
       Bound for pitch and yaw rates
9
10
11 %% Initial Guesses from Particle Swarm Optimization
12 %x0
13
14 %% Sequential Quadratic Programming using fmincon using the
       above initial guess
15 opts = optimoptions(@fmincon,'Algorithm','sqp','Display','
       iter','OptimalityTolerance',1e-3,'StepTolerance',1e-20,'
       PlotFcn','optimplotfval','MaxFunctionEvaluations',1500);
16 [x,fval] = fmincon(fun,x0,[],[],[],[],lb,ub,nonlcon,opts);
17
18
19 %% Objective Function and Constraints
20 function y=func(x)
21          [obj,~,~] = Trajectory_2d(x);
22          y = 1*obj;
23 end
```

```
24
25 function [c,ceq]=cons(x)
26         [~,in_eq,eq] = Trajectory_2d(x);
27         c=in_eq;
28         ceq = eq;
29 end
```

A.9 Function for Multi Disciplinary Feasible Analysis

```
1  function [ob,c1,c2,c3,c4,c5,c6,c7] = MDF(x)%For SQP and PSO
2
3  x_trajec = x(1:23);
4  x_solid = x(24:29);
5
6  % Launch vehicle Sizing Module
7  eps = [0.12;0.12;0.15];
8  T_by_W = [1.4;1.4;0.6];
9  I_sp = [284.7784;284.7784;310];
10 n=3;
11
12 %Launch Vehicle Sizing Module
13 [~,~, m_prop,~, T_m] = lv_staging_opt(eps,I_sp,T_by_W,n);
14
15
16 % Launch Vehicle Propulsion Module
17 [P_c,Th_vac,~,~,t,dt,tb_vac,~,m_dot] = solid_prop(round(
       x_solid(1)),x_solid(2),x_solid(3),x_solid(4),x_solid(5),
       x_solid(6));
18
19 %Launch Vehicle Trajectory Module
20 %if (sum(Th_vac.*dt)/1*10^(8))>=1 && min(Th_vac)/(1*10^(6))
       >=1
21 %if (sum(Th_vac.*dt)/1*10^(8))>=1 && min(Th_vac)
       /(0.93*10^(6))>=1
22 %if (sum(Th_vac.*dt))/(1.3*10^(8))>=1
```

```
23 if (tb_vac/(sum(Th_vac.*dt))) >=(75/(1.3*10^(8))) %&& (
   length(t) == length(m_dot))
24       [ob,c1,c2,c3,c4,c5] = Trajectory_2d(x_trajec,t,m_dot,
            Th_vac,tb_vac,x_solid(6));
25 else
26 ob = Inf;
27 c1 = Inf;
28 c2 = Inf;
29 c3 = Inf;
30 c4 = Inf;
31 c5 = Inf;
32 % Constraints for solid propellent
33 c6 = max(P_c)/(95*10^5) - 1;
34 c7 = tb_vac/150 - 1;
35 return
36 end
37
38 % Constraints for solid propellent
39 c6 = max(P_c)/(95*10^5) - 1;
40 c7 = tb_vac/150 - 1;
41
42 % For Sequential Quadratic Programming
43 %c = [c1, c2, c3, c5, c6, c7];
44 %ceq = [c4];
45
46 end
```

A.10 Optimization of MDF uisng PSO

```
1 %% -------- Particle Swarm Optimization -------- %%
2
3 %problem settings
4 %% For 2D Trajectory Optimization
5 lb
   =[-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,0,0,8
```

```
6  ,3,0.10000000000000,0.3,0.0100000000000000,0.0100000000000000
7  ,4.314604551295555];                                    %Lower Bound
8
9  ub=[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,15,15,
10 0.30000000000000,0.7,0.1500000000000000,0.500000000000000,5];
                                                           %Upper
   Bound
11
12
13 fun = @func;          %Fitness Function
14 %ALgorithm Parameters
15 w_max=1.2;
16 w_min=0.1;  %Linearly Decreasing Inertia Weight
17 c1=2;        %self acceleration coefficient
18 c2=2;        %social acceleration coefficient
19 Np=1000;     %Size of population
20 T=100;       %Number of iterations
21
22 %% Algorithm %%
23 %Initialization
24 %Creating random population
25 f = zeros(Np,1); %Array to store fitness values of
   population
26 D = length(lb); %No.of variables in each particle of
   population
27
28 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);%
   position of Initial Random Population
29 %P = gpuArray(P);
30 v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); %
   velocity of Initial Random Population
31 %v = gpuArray(v);
32
```

```
33 for p = 1:Np
34     f(p) = func(P(p,:),1);%Evaluating fitness function for
           the initial population
35 end
36
37 pbest = P;%Initialize the personal best solutions
38 f_pbest = f;%Initialize the fitness of the personal best
       solutions
39
40 [f_gbest, ind] = min(f_pbest); %Determining the fitness
       function for global best solution
41 gbest = P(ind,:); %Determining the global best solution
42 %f_plot =zeros(T,1);
43
44 %Iteration Loop
45 for i=1:T
46     w=w_max - (w_max - w_min)*i/T; %Linearly Varying
           Inertial Weight
47     for p=1:Np
48         v(p,:) = 0.73*(w*v(p,:) + c1*rand(1,D).*(pbest(p,:)
               - P(p,:)) + c2*rand(1, D).*(gbest - P(p,:))); %
               Calculate the Velocity
49         P(p,:) = P(p,:) + v(p,:);  %Update the position
50
51         P(p,:) = max(P(p,:), lb); %Bounding the violating
               variables to their lower bound
52         P(p,:) = min(P(p,:), ub); %Bounding the violating
               variables to their upper bound
53
54         f(p) = func(P(p,:),i); %Determining the fitness of
               new solution
55
56         if f(p) < f_pbest(p)
57             f_pbest(p) = f(p);%Updating the fitness value of
                   the personal best value
```

```
58              pbest(p,:) = P(p,:);%Updating the personal best
                   position value
59
60              if f_pbest(p) < f_gbest
61                  f_gbest = f_pbest(p); %Updating the global
                       best fitness value value
62                  gbest = pbest(p,:); %Updating the global
                       best position
63              end
64
65          end
66
67      end
68 %f_plot(i) = f_plot(i) + f_gbest;
69 disp(i);
70 disp(f_gbest);
71 end
72
73
74 %% Objective (or) Fitness Function for Multi-Disciplinary
       Feasible Method
75 function y=func(x,k)
76 %Non-Stationary, Adaptive Exterior Penalty Function%
77          %Re = 6378165.85
78          [obj_func,c1,c2,c3,c4,c5,c6,c7] = MDF(x)   ;
79          q_x = [max(0,c1), max(0,c2), max(0,c3), max(0,c4),
                max(0,c5),max(0,c6),max(0,c7)];%,max(0,c8)];
80          y = 0;
81          for i =1:length(q_x)
82              if q_x(i) <= 0.001
83              y = y + k*sqrt(k)*10*(q_x(i))^(1);
84              elseif q_x(i) <= 0.1
85              y = y + k*sqrt(k)*20*(q_x(i))^(1);
86              elseif q_x(i) <= 1
87              y = y + k*sqrt(k)*100*(q_x(i))^(1);
```

```
                else
                y = y + k*sqrt(k)*300*(q_x(i))^(2);
                end
            end
            y = y + 1*obj_func;
end
```

A.11 Optimization of MDF uisng SQP

```
%% --------- Sequential Quadratic Programming ---------- %%

%Problem Settings
fun = @func;
nonlcon=@cons;

%% For 2D Trajectory Optimization
lb
   =[-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,0,0,8,...

3,0.2,0.3,0.01,0.01,4.31460455129555]; %Lower Bound
ub
   =[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,15,15,0.9,...

 ,0.7,0.15,0.5,5];                    %Upper Bound

%% Initial Guesses from Particle Swarm Optimization
%x0

%% Sequential Quadratic Programming using fmincon using the
   above initial guesses
opts = optimoptions(@fmincon,'Algorithm','sqp','Display','
   iter','OptimalityTolerance',1e-3,'StepTolerance',1e-20,'
   PlotFcn','optimplotfval','MaxFunctionEvaluations',250);
[x,fval] = fmincon(fun,x0,[],[],[],[],lb,ub,nonlcon,opts);

```

```
20
21 %% Objective Function and Constraints
22 function y=func(x)
23
24         [obj,~,~,~,~,~,~,~] = MDF(x);
25
26         y = 1*obj;
27 end
28
29 function [c,ceq]=cons(x)
30         [~,c1,c2,c3,c4,c5,c6,c7] = MDF(x);
31         c=[c1,c2,c3,c5,c6,c7];
32         ceq =[c4];
33 end
```

A.12 Three Dimensional Trajectory Analysis

```
1
2 function [ALT,FPA,VEL,t,Qmax,inc,long_F_impact,lat_F,long_F,
    t_index,steer] = threed_simulation(x)      %Uncomment and
    use this line when using this function for getting output
3 %function [ob,c1,c2,c3,c4,c5] = threed_simulation(x)
                          %Uncomment and use this line when
    using PSO and DE
4 %function [ob,c,ceq] = threed_simulation(x)
                                  %Uncomment and use this
    line when using SQP
5 %Desired Orbit
6 ro=[500 500]*10^3;
                                                          %
    Circular Orbit of radius 500 km
7 %AzL = 180;  %For Polar Orbit
                                                  %Launch
    Azimuth(degrees)
8 AzL = 140;  %For Dog-Leg Maneuver
```

```
9
10
11 %Vehicle Parameters
12 % Vehicle parameters
13 m_pl = 500;

    %Payload mass in kg
14 Isp = [260 260 310];
                                                        %
    Isp of stages in sec
15 sf = [0.12 0.12 0.15];
                                                      %
    Structural factors of stages
16 tw_ratio = [1.4 1.4 0.6];
                                                    %Thrust-
    to-weight ratio of all stages
17 dia = 2;

    %Diameter of Launch Vehicle in m
18 area = pi*(dia/2)^2;
                                                        %
    Cross sectional area of Launch Vehicle in m2
19
20 %Initial attitude angles
21 roll = 0;
22 yaw = 0;
23 pitch = 0;
24
25 % Launch Pad parameters (SHAR)
26 lat_L = 13.7259;
27 long_L = 80.2266;
28 h0 = 26.40;
29
30 % Earth Parameters
31 g0 = 9.80665;
```

```
    %[m/s^2] g at Earth sea level
Re = 6378165.857;

    %Equatorial Radius of earth
Rp = 6356783.832;

    %Polar Radius of earth
omega_earth = [0 0 7.2921159*(10^-5)];
                                        %Angular Velocity of
    earth
mu = 3.986004418*(10^14);
                                                %gravity
     parameter
j2 = 1.0823*(10^-3);
                                                %
    gravity parameter
f = (Re-Rp)/Re;

    %Oblateness/Flattening

% Derived quantities
C = g0.*Isp;
apogee = max(ro);
perigee = min(ro);
k_e = (Re/Rp)^2;
PhiL = atand((1-f)^2*tand(lat_L));
Rs = Re/sqrt(1+(k_e-1)*(sind(PhiL))^2);
Ri = Rs+h0;
ri = Ri*[cosd(lat_L)*cosd(long_L) cosd(lat_L)*sind(long_L)
    sind(lat_L)];
vi = cross(omega_earth,ri);
AzL=180+AzL;

%inclination
```

```
52 incl = zeros(7000,1);
53 incl(1)=acosd(sind(AzL)*cosd(lat_L));
54
55 %Vel_req
56 ve_req=10387.92;
57 %Launch Vehicle Sizing optimization
58 [mi, ms, mp, ~, T_m] = lv_staging_opt(sf,Isp,T_by_W,3);
59 mdot = T_m'./(C);
60 T = T_m';
61
62 %Transformation Matrices
63 %1) Inertial Frame to Launch Frame
64 IL(1,:) = [cosd(lat_L)*cosd(long_L) cosd(lat_L)*sind(long_L)
       sind(lat_L)];
65 IL(2,:) = [sind(lat_L)*cosd(long_L)*sind(AzL)-cosd(AzL)*sind
    (long_L) cosd(AzL)*cosd(long_L)+sind(AzL)*sind(lat_L)*
    sind(long_L) -sind(AzL)*cosd(lat_L)];
66 IL(3,:) = [-sind(AzL)*sind(long_L)-cosd(AzL)*sind(lat_L)*
    cosd(long_L) sind(AzL)*cosd(long_L)-cosd(AzL)*sind(lat_L)
    *sind(long_L) cosd(AzL)*cosd(lat_L)];
67
68 %2) Launch Frame to Body Frame
69 LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(pitch
    )+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*cosd(pitch)
    -cosd(roll)*sind(pitch)];
70 LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(yaw)
    ];
71 LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(pitch
    )-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*sind(pitch)
    +cosd(roll)*cosd(pitch)];
72 LB = [LB1;LB2;LB3];
73 IB = LB*IL;
74
75 [t,m,vm,vrm,am,Q,~,mprop,mach,h,gammarel,azi,theta,lat_F,
    long_F,aoa,gamma,thetarot,theta_alt]=deal(zeros(7000,1));
```

```
76 [a,v,vr,vrbf,r,g,steer]=deal(zeros(7000,3));
77 vrg = zeros(3,7000);
78
79 % Base conditions (@t=0)
80 v(1,:) = [vi(1) vi(2) vi(3)];
81 r(1,:) = [ri(1) ri(2) ri(3)];
82 h(1) = norm(r(1,:))-Rs;
83 %Lat_flight(1) = lat_L;
84 %Long_flight(1) = long_L;
85 t(1) = 0;
86 m(1) = mi(1);
87 mprop(1) = mp(1);
88 gammarel(1) = 90;
89 lat_F(1) = lat_L;
90 long_F(1) = long_L;
91
92
93 iter = 2;
94 delt=0.1;

    %Time step for integration

95
96
97 %% Vertical takeoff phase
98 while h(iter-1)<100
99     if h(iter-1)<1.782853443358382e+05
100         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
101     elseif h(iter-1)<1.882853443358382e+05
102         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
              +05,'None');
103     else
104         a_amb = 0;
105         rho_amb = 0;
106     end
107
```

```
108     t(iter) = t(iter-1) + delt;
109     m(iter) = m(iter-1)-mdot(1,1)*delt;
110     mprop(iter) = mprop(iter-1)-mdot(1,1)*delt;
111     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
112     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
            ,area);
113     atb = ([T(1,1) 0 0]-D)./m(iter,:);
114     a(iter,:) = (IB\atb')'+g(iter,:);
115     am(iter,:) = norm(a(iter,:));
116     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
117     vr(iter,:) = v(iter,:)-v(1,:);
118     vm(iter) = norm(v(iter,:));
119     vrm(iter) = norm(vr(iter,:));
120     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
121     h(iter) = norm(r(iter,:))- Rs;
122
123     vrbf(iter,:) = IB*vr(iter,:)';
124     aoa(iter)=atan2d(vrbf(iter,3),vrbf(iter,1));
125     pc=asin(r(iter,3)/norm(r(iter,:))); %Geocentric latitude
126     ti=atan(r(iter,2)/r(iter,1)); %Inertial longitude
127     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
128           -sin(ti),cos(ti),0;
129           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
130     vrg(:,iter) = (IG*vr(iter,:)');
131     gamma(iter)=90;
132     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
            ,:))*vrm(iter)));
133     pitch=0;
134     theta(iter) = aoa(iter)+gamma(iter);
135     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
136     steer(iter,:) = [roll yaw pitch];
137     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
            ,2)^2));
138     long_F(iter) = atan2d(r(iter,2),r(iter,1));
139     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
```

```
        -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
        -1,2)/1000,vr(iter-1,3)/1000);

    iter = iter+1;
end

id1 = find(t>0,1,'last');
%% Constant turnover phase
%x=-1*ones(22,1);
while t(iter-1)<16.5
    pitch = pitch+(x(21)*delt);
    yaw = yaw+(0.5*x(22)*delt);
    t(iter) = t(iter-1)+delt;
    if h(iter-1)<1.782853443358382e+05
        [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
    elseif h(iter-1)<1.882853443358382e+05
        [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
            +05,'None');
    else
        a_amb = 0;
        rho_amb = 0;
    end
    m(iter) = m(iter-1)-mdot(1,1)*delt;
    mprop(iter) = mprop(iter-1)-mdot(1,1)*delt;
    g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
    [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
        ,area);
    atb = ([T(1,1) 0 0]-D)/m(iter,:);
    LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
        pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
        cosd(pitch)-cosd(roll)*sind(pitch)];
    LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
        yaw)];
    LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
        pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
```

```
    sind(pitch)+cosd(roll)*cosd(pitch)];
LB = [LB1;LB2;LB3];
IB = LB*IL;
a(iter,:) = (IB\atb')'+g(iter,:);
am(iter,:) = norm(a(iter,:));
v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
vr(iter,:) = v(iter,:)-v(1,:);
vm(iter) = norm(v(iter,:));
vrm(iter) = norm(vr(iter,:));
r(iter,:) = r(iter-1,:)+v(iter,:).*delt;

vrbf(iter,:) = IB*vr(iter,:)';
aoa(iter)=atan2d(vrbf(iter,3),vrbf(iter,1));

pc=asin(r(iter,3)/norm(r(iter,:))); %Geocentric latitude
ti=atan(r(iter,2)/r(iter,1)); %Inertial longitude
IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
      -sin(ti),cos(ti),0;
      -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
vrg(:,iter) = (IG*vr(iter,:)');
gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
    ,:))*vrm(iter)));
thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
    norm(ri))));

theta(iter)=aoa(iter)+gamma(iter)+thetarot(iter);
azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
h(iter) = norm(r(iter,:))-Rs;
steer(iter,:) = [roll yaw pitch];
lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
    ,2)^2));
long_F(iter) = atan2d(r(iter,2),r(iter,1));
[~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
    -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
```

```
        -1,2)/1000,vr(iter-1,3)/1000);
197
198
199     iter=iter+1;
200 end
201
202 id2 = find(t>0,1,'last');
203
204 %% Gravity turn phase
205 while mprop(iter-1)>mp(1)*0.002
206     t(iter) = t(iter-1)+delt;
207     if h(iter-1)<1.782853443358382e+05
208         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
209     elseif h(iter-1)<1.882853443358382e+05
210         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
            +05,'None');
211     else
212         a_amb = 0;
213         rho_amb = 0;
214     end
215     m(iter) = m(iter-1)-mdot(1,1)*delt;
216     mprop(iter) = mprop(iter-1)-mdot(1,1)*delt;
217     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
218     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
        ,area);
219     atb = ([T(1,1) 0 0]-D)/m(iter,:);
220     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
        pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
        cosd(pitch)-cosd(roll)*sind(pitch)];
221     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
        yaw)];
222     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
        pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
        sind(pitch)+cosd(roll)*cosd(pitch)];
223     LB = [LB1;LB2;LB3];
```

```
224     IB = LB*IL;
225     a(iter,:) = (IB\atb')'+g(iter,:);
226     am(iter,:) = norm(a(iter,:));
227     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
228     vr(iter,:) = v(iter,:)-v(1,:);
229     vm(iter) = norm(v(iter,:));
230     vrm(iter) = norm(vr(iter,:));
231     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
232
233     aoa(iter)=0;
234     pc=asin(r(iter,3)/norm(r(iter,:))); %Geocentric latitude
235     ti=atan(r(iter,2)/r(iter,1)); %Inertial longitude
236
237     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
238           -sin(ti),cos(ti),0;
239           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
240     vrg(:,iter) = (IG*vr(iter,:)');
241     lam=atan2d(vrg(2,iter),vrg(1,iter));
242     gam=asind(r(iter,:)*vrg(:,iter)/(norm(r(iter,:))*norm(
          vrg(:,iter))));
243
244     gamma(iter)=gravity_turn(gam,lam,g(iter,:),atb,delt,v
          (1,:),IG,r(iter-1,:),v(iter-1,:));
245     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
          ,:))*vrm(iter)));
246
247     thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
          norm(ri))));
248     theta(iter)=aoa(iter)+gamma(iter)+thetarot(iter);
249     theta_alt(iter)=pitch+(sind(gammarel(iter))*(norm(g(iter
          ,:))/vrm(iter)))*delt;
250     pitch=theta_alt(iter);
251     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
252     h(iter) = norm(r(iter,:))-Rs;
253     steer(iter,:) = [roll yaw pitch];
```

```
254     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
            ,2)^2));
255     long_F(iter) = atan2d(r(iter,2),r(iter,1));
256     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
            -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
            -1,2)/1000,vr(iter-1,3)/1000);
257
258
259     t(iter) = t(iter-1)+delt;
260     iter=iter+1;
261 end
262
263 id3 = find(t>0,1,'last');
264
265 %% First Stage Separation
266 while t(iter-1)<=t(id3)+8
267     t(iter) = t(iter-1)+delt;
268     if h(iter-1)<1.782853443358382e+05
269         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
270     elseif h(iter-1)<1.882853443358382e+05
271         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
                +05,'None');
272     else
273         a_amb = 0;
274         rho_amb = 0;
275     end
276     m(iter) = mi(1,1)-mp(1,1)-ms(1,1);
277     mprop(iter) = mp(2,1);
278     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
279     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
            ,area);
280     atb = -D/m(iter,:);
281     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
            pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
            cosd(pitch)-cosd(roll)*sind(pitch)];
```

```
282     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
            yaw)];
283     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
            pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
            sind(pitch)+cosd(roll)*cosd(pitch)];
284     LB = [LB1;LB2;LB3];
285     IB = LB*IL;
286     a(iter,:) = (IB\atb')'+g(iter,:);
287     am(iter,:) = norm(a(iter,:));
288     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
289     vr(iter,:) = v(iter,:)-v(1,:);
290     vm(iter) = norm(v(iter,:));
291     vrm(iter) = norm(vr(iter,:));
292     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
293
294     aoa(iter)=0;
295     pc=asin(r(iter,3)/norm(r(iter,:))); %Geocentric latitude
296     ti=atan(r(iter,2)/r(iter,1)); %Inertial longitude
297     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
298           -sin(ti),cos(ti),0;
299           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
300     vrg(:,iter) = (IG*vr(iter,:)');
301     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
302     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
            ,:))*vrm(iter)));
303     thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
            norm(ri))));
304     theta(iter)=aoa(iter)+gammarel(iter)+thetarot(iter);
305     pitch=90-theta(iter);
306
307     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
308     h(iter) = norm(r(iter,:))-Rs;
309     steer(iter,:) = [roll yaw pitch];
310     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
            ,2)^2));
```

```
311       long_F(iter) = atan2d(r(iter,2),r(iter,1));
312       [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
              -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
              -1,2)/1000,vr(iter-1,3)/1000);
313
314
315       t(iter) = t(iter-1)+delt;
316       iter=iter+1;
317   end
318
319   id4 = find(t>0,1,'last');
320   %[a, e, i, raan, aop, nu] = sv_2_oe(r(id4,1)/1000,r(id4,2)
          /1000,r(id4,3)/1000,vr(id4,1)/1000,vr(id4,2)/1000,vr(id4
          ,3)/1000)
321   %[a1, e1, i1, raan1, aop1, nu1] = convert(r(id4,1)/1000,r(
          id4,2)/1000,r(id4,3)/1000,vr(id4,1)/1000,vr(id4,2)/1000,
          vr(id4,3)/1000)
322   %[r_test, v_test] = oe_2_sv(a,e,i,raan,aop,nu)
323   %[r_test1, v_test2] = convertback(a1,e1,i1,raan1,aop1,nu1)
324   %display(r(id4,:))
325   %display(vr(id4,:))
326   %[rootbisect,long_F_impact,lat_F_impact] = bisectimpact(r(
          id4,:),vr(id4,:))
327   %[rootbisect1,long_F_impact1,lat_F_impact1] = impact_point(r
          (id4,:),vr(id4,:))
328
329   long_F_impact = zeros(7000,1);
330   lat_F_impact = zeros(7000,1);
331   [~,long_F_impact(iter-1),lat_F_impact(iter-1)] =
          impact_point(r(id4,:),vr(id4,:));
332
333   id5=id4;
334   k1=iter;
335   %% Stage-2 Optimal Trajectory
336   time = fix((mprop(iter-1)-mp(2)*0.002)/mdot(2));
```

```
337
338 %Finding pitch rates at each time point
339 n_pitch=10;

    %Number of segments
340 PRval= x(1:10);
341 PRates = find_rates(time,n_pitch,delt,PRval);
342
343 %Finding Yaw rate at each time point
344 n_yaw = 5;

    %No.of Segmets
345 YRval = x(23:27);
346 YRates = find_rates(time,n_yaw,delt,YRval);
347
348 k2=1;
349
350 while iter<k1+size(PRates,2)
351     t(iter) = t(iter-1)+delt;
352
353     %% For IIP constraint
354     %if long_F_impact(iter-1)<=83 %&& lat_F_impact <= 9.51
355     %   yaw=yaw + (YRates(k2)*delt);
356     %end
357     %% For Dog-Leg Maneuver
358     %if steer(iter-1,2)-steer(id5,2)<=45
359     %   yaw=yaw + (YRates(k2)*delt);
360     %end
361
362     %if abs(steer(iter-1,2)-steer(id5,2))<=45
363     %   yaw = yaw + (Yawrates3fin(1,k3)*delt);
364     %end
365
366
367     if (steer(id5,2) - steer(iter-1,2)) <= 70 && lat_F(iter
```

```
    -1) <= 12
368     yaw = yaw + (YRates(k2)*delt);
369     end
370
371     if h(iter-1)<1.782853443358382e+05
372         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
373     elseif h(iter-1)<1.882853443358382e+05
374         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
            +05,'None');
375     else
376         a_amb = 0;
377         rho_amb = 0;
378     end
379     m(iter) = m(iter-1)-mdot(1,2)*delt;
380     mprop(iter) = mprop(iter-1)-mdot(1,2)*delt;
381     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
382     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
        ,area);
383     atb = ([T(1,2) 0 0]-D)/m(iter,:);
384     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
        pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
        cosd(pitch)-cosd(roll)*sind(pitch)];
385     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
        yaw)];
386     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
        pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
        sind(pitch)+cosd(roll)*cosd(pitch)];
387     LB = [LB1;LB2;LB3];
388     IB = LB*IL;
389     a(iter,:) = (IB\atb')'+g(iter,:);
390     am(iter,:) = norm(a(iter,:));
391     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
392     vr(iter,:) = v(iter,:)-v(1,:);
393     vm(iter) = norm(v(iter,:));
394     vrm(iter) = norm(vr(iter,:));
```

```
395     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
396
397
398
399     pc=asin(r(iter,3)/norm(r(iter,:)));
                                                %Geocentric
        latitude
400     ti=atan(r(iter,2)/r(iter,1));
                                                    %Inertial
        longitude
401     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
402           -sin(ti),cos(ti),0;
403           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
404     vrg(:,iter) = (IG*vr(iter,:)');
405     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
406     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
        ,:))*vrm(iter)));
407     theta(iter) = pitch+PRates(k2)*delt;
408     pitch=theta(iter);
409     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
410     %aoa(iter)=theta(iter)-gamma(iter);
411
412
413     vrbf(iter,:) = IB*vr(iter,:)';
414     aoa(iter)=atan2d(vrbf(iter,3),vrbf(iter,1));
415
416
417     h(iter) = norm(r(iter,:))-Rs;
418     steer(iter,:) = [roll yaw pitch];
419     k2=k2+1;
420     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
        ,2)^2));
421     long_F(iter) = atand(r(iter,2)/r(iter,1));
422     [~,long_F_impact(iter),lat_F_impact(iter)] =
        impact_point(r(iter,:),vr(iter,:));
```

```
423     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
            -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
            -1,2)/1000,vr(iter-1,3)/1000);
424
425
426     iter=iter+1;
427 end
428 id6 = find(t>0,1,'last');
429
430  %% Second Stage Separation
431 while t(iter-1)<=t(id6)+8
432     t(iter) = t(iter-1)+delt;
433     if h(iter-1)<1.782853443358382e+05
434         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
435     elseif h(iter-1)<1.882853443358382e+05
436         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
                +05,'None');
437     else
438         a_amb = 0;
439         rho_amb = 0;
440     end
441     m(iter) = mi(1)-mp(1)-ms(1)-mp(2)-ms(2);
442     mprop(iter) = mp(3);
443     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
444     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
            ,area);
445     atb = -D/m(iter,:);
446     LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
            pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
            cosd(pitch)-cosd(roll)*sind(pitch)];
447     LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
            yaw)];
448     LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
            pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
            sind(pitch)+cosd(roll)*cosd(pitch)];
```

```
449     LB = [LB1;LB2;LB3];
450     IB = LB*IL;
451     a(iter,:) = (IB\atb')'+g(iter,:);
452     am(iter,:) = norm(a(iter,:));
453     v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
454     vr(iter,:) = v(iter,:)-v(1,:);
455     vm(iter) = norm(v(iter,:));
456     vrm(iter) = norm(vr(iter,:));
457     r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
458
459     aoa(iter) = aoa(iter-1);
460     pc=asin(r(iter,3)/norm(r(iter,:)));
                                          %Geocentric
        latitude
461     ti=atan(r(iter,2)/r(iter,1));
                                                %Inertial
        longitude
462     IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
463           -sin(ti),cos(ti),0;
464           -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
465     vrg(:,iter) = (IG*vr(iter,:)');
466
467     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
468     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
        ,:))*vrm(iter)));
469     thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
        norm(ri))));
470     theta(iter)=aoa(iter)+gamma(iter)+thetarot(iter);
471     pitch=theta(iter);
472     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
473     h(iter) = norm(r(iter,:))-Rs;
474     steer(iter,:) = [roll yaw pitch];
475     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
        ,2)^2));
476     long_F(iter) = atan2d(r(iter,2),r(iter,1));
```

```
477       [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
              -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
              -1,2)/1000,vr(iter-1,3)/1000);
478
479
480       t(iter) = t(iter-1)+delt;
481       iter=iter+1;
482   end
483
484   id7 = find(t>0,1,'last');
485
486   [~,long_F_impact(iter-1),~] = impact_point(r(id4,:),vr(id4
          ,:));
487
488   k2=iter;
489
490   %% STAGE 3 Optimal trajectory
491   m(iter-1)=mi(3);
492   mprop(iter-1)=mp(3);
493   time3 = fix((mprop(iter-1)-mp(3)*0.002)/mdot(3));
494
495   n3=10;

        %Number of segments
496   PRval3= x(11:20);
497   PRates3 = find_rates(time3,n3,delt,PRval3);
498
499   %Uncomment when using IIP as a constraint, comment when
          using Dog-Leg Maneuver
500   %For optimizing yaw-rates
501   ndash3=5;
502   Yawrates3 = x(28:32);
                                                                %
        Number of segments
503   Yawrates3fin = find_rates(time3,ndash3,delt,Yawrates3);
```

```
504
505 k3=1;
506 while iter<k2+size(PRates3,2)
507     t(iter) = t(iter-1)+delt;
508
509     %Uncomment when using IIP as a constraint, comment when
          using Dog-Leg Maneuver
510     %if long_F_impact(iter-1) >= long_L %&& lat_F_impact(
          iter-1)<=7.53
511     %if  long_F_impact(iter-1) <= 83 && long_F_impact(iter
          -1) >= 82
512     %    yaw = yaw + (Yawrates3fin(1,k3)*delt);
513     %end
514
515    %if long_F_impact(iter-1) <= long_L && (steer(id7,2) -
         steer(iter-1,2)) <= 45
516     if (steer(id7,2) - steer(iter-1,2)) <= 20 %&& iter <
         4680
517     yaw = yaw + (Yawrates3fin(1,k3)*delt);
518     end
519
520
521     if h(iter-1)<1.782853443358382e+05
522         [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
523     elseif h(iter-1)<1.882853443358382e+05
524         [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
              +05,'None');
525     else
526         a_amb = 0;
527         rho_amb = 0;
528     end
529     m(iter) = m(iter-1)-mdot(1,3)*delt;
530     mprop(iter) = mprop(iter-1)-mdot(1,3)*delt;
531     g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
532     [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
```

```
    ,area);
533 atb = ([T(1,3) 0 0]-D)/m(iter,:);
534 LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
    pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
    cosd(pitch)-cosd(roll)*sind(pitch)];
535 LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
    yaw)];
536 LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
    pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
    sind(pitch)+cosd(roll)*cosd(pitch)];
537 LB = [LB1;LB2;LB3];
538 IB = LB*IL;
539 a(iter,:) = (IB\atb')'+g(iter,:);
540 am(iter,:) = norm(a(iter,:));
541 v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
542 vr(iter,:) = v(iter,:)-v(1,:);
543 vm(iter) = norm(v(iter,:));
544 vrm(iter) = norm(vr(iter,:));
545 r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
546
547 pc=asin(r(iter,3)/norm(r(iter,:)));
                                        %Geocentric
    latitude
548 ti=atan(r(iter,2)/r(iter,1));
                                              %Inertial
    longitude
549 IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
550       -sin(ti),cos(ti),0;
551       -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
552 vrg(:,iter) = (IG*vr(iter,:)');
553 gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
554 gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
    ,:))*vrm(iter)));
555 theta(iter) = pitch+PRates3(k3)*delt;
556 pitch=theta(iter);
```

```
557
558       azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
559       aoa(iter)=theta(iter)-gamma(iter);
560
561       vrbf(iter,:) = IB*vr(iter,:)';
562       aoa(iter)=atan2d(vrbf(iter,3),vrbf(iter,1));
563
564       h(iter) = norm(r(iter,:))-Rs;
565       steer(iter,:) = [roll yaw pitch];
566       k3=k3+1;
567       lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
              ,2)^2));
568       long_F(iter) = atan2d(r(iter,2),r(iter,1));
569       [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
              -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
              -1,2)/1000,vr(iter-1,3)/1000);
570
571
572       [~,long_F_impact(iter),~] = impact_point(r(iter,:),vr(
              iter,:));
573
574       iter=iter+1;
575   end
576   id8 = find(t>0,1,'last');
577
578   %% Stage 3 Separation
579   while t(iter-1)<=t(id8)+8
580       t(iter) = t(iter-1)+delt;
581       if h(iter-1)<1.782853443358382e+05
582           [~, a_amb, ~, rho_amb]=atmoscoesa(h(iter-1),'None');
583       elseif h(iter-1)<1.882853443358382e+05
584           [~, a_amb,~, rho_amb]=atmoscoesa(1.782853443358382e
                  +05,'None');
585       else
586           a_amb = 0;
```

```
587             rho_amb = 0;
588         end
589         m(iter) = mi(1,1)-mp(1)-ms(1)-mp(2)-ms(2)-mp(3)-ms(3);
590         mprop(iter) = 0;
591         g(iter,:) = gravity(r(iter-1,:),j2,Re,mu);
592         [D,Q(iter),mach(iter)] = drag(vr(iter-1,:),a_amb,rho_amb
                ,area);
593         atb = -D/m(iter,:);
594         LB1 = [cosd(yaw)*cosd(pitch) cosd(roll)*sind(yaw)*cosd(
                pitch)+sind(roll)*sind(pitch) sind(roll)*sind(yaw)*
                cosd(pitch)-cosd(roll)*sind(pitch)];
595         LB2 = [-sind(yaw) cosd(roll)*cosd(yaw) sind(roll)*cosd(
                yaw)];
596         LB3 = [cosd(yaw)*sind(pitch) cosd(roll)*sind(yaw)*sind(
                pitch)-sind(roll)*cosd(pitch) sind(roll)*sind(yaw)*
                sind(pitch)+cosd(roll)*cosd(pitch)];
597         LB = [LB1;LB2;LB3];
598         IB = LB*IL;
599         a(iter,:) = (IB\atb')'+g(iter,:);
600         am(iter,:) = norm(a(iter,:));
601         v(iter,:) = v(iter-1,:)+a(iter,:).*delt;
602         vr(iter,:) = v(iter,:)-v(1,:);
603         vm(iter) = norm(v(iter,:));
604         vrm(iter) = norm(vr(iter,:));
605         r(iter,:) = r(iter-1,:)+v(iter,:).*delt;
606
607         aoa(iter) = aoa(iter-1);
608         pc=asin(r(iter,3)/norm(r(iter,:)));
                                                %Geocentric
                latitude
609         ti=atan(r(iter,2)/r(iter,1));
                                                    %Inertial
                longitude
610         IG = [-sin(pc)*cos(ti),-sin(pc)*sin(ti),cos(pc);
611               -sin(ti),cos(ti),0;
```

```
612             -cos(pc)*cos(ti),-cos(pc)*sin(ti),-sin(pc)];
613     vrg(:,iter) = (IG*vr(iter,:)');
614     gamma(iter)=(atand(vrg(3,iter)/vrg(1,iter)));
615     gammarel(iter)=asind(r(iter,:)*vr(iter,:)'/(norm(r(iter
            ,:))*vrm(iter)));
616     thetarot(iter)=(acosd((r(iter,:)*ri')/(norm(r(iter,:))*
            norm(ri))));
617     theta(iter)=aoa(iter)+gamma(iter)+thetarot(iter);
618     pitch=theta(iter);
619     azi(iter) = atan2d(vrg(2,iter),vrg(1,iter));
620     h(iter) = norm(r(iter,:))-Rs;
621     steer(iter,:) = [roll yaw pitch];
622     lat_F(iter) = atan2d(r(iter,3),sqrt(r(iter,1)^2+r(iter
            ,2)^2));
623     long_F(iter) = atan2d(r(iter,2),r(iter,1));
624     [~,~,incl(iter),~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter
            -1,2)/1000,r(iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter
            -1,2)/1000,vr(iter-1,3)/1000);
625
626
627     t(iter) = t(iter-1)+delt;
628     iter=iter+1;
629 end
630 id9 = find(t>0,1,'last');
631
632
633 %% For Output
634 %(Uncomment this block and comment the below blocks for
        running this
635 %script as function for getting outputs)
636
637 %The outputs returned by the function
638 [~,~,inc,~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter-1,2)/1000,r(
        iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter-1,2)/1000,vr(
        iter-1,3)/1000);
```

```
639 ALT=h;
640 Qmax=Q;
641 FPA=gammarel;
642 VEL=vm;
643 t_index=[id1,id2,id3,id4,id5,id6,id7,id8,id9];
644
645
646 %% 0) For Optimization
647 %(Uncomment this block and comment the above and below
      blocks for running this
648 %script as function for getting constraints and objective
      functions for PSO,DE)
649
650 % Comment the above block and uncomment the below blocks for
       Optimization
651 %[~,~,inc,~,~,~]=sv_2_oe(r(iter-1,1)/1000,r(iter-1,2)/1000,r
      (iter-1,3)/1000,vr(iter-1,1)/1000,vr(iter-1,2)/1000,vr(
      iter-1,3)/1000);
652 %ALT=h(iter-1);
653 %Qmax=max(Q);
654 %FPA=gammarel(iter-1);
655 %VEL=vm(iter-1);
656
657 %% 1) For Particle swarm optimization, Differential
      Evolution
658 %ob = -ALT/(Re);
659 %c1 = abs(VEL/sqrt(Re*g0) - 1000*sqrt(398600/((ALT/1000)+Re
      /1000))/(sqrt(Re*g0))) - 10/sqrt(Re*g0);
660 %c2 = abs(FPA/90) - (2/90);
661 %c3 = Qmax/100000 - 0.45;
662 %c4 = (inc/90 - 1);
663 %c5 = -(steer(id5,2)/90 - steer(id9,2)/90) + 1/8;
664 %c5 = (long_F_impact(iter-1)/90 - long_L/90);
665 %c5 = steer(id9,2)/90 + 90/90;
666 %c6 = (long_F(iter-1)/90 - long_L/90);
```

```
667 %c4 = -inc/90 + 1;
668 %c6 = (- long_F_impact(id5)/90 + 82/90)^2; %+ (lat_F_impact(
    id5)/90 - 7.53/90)^2 - 1/(90^2);
669 %c6 = -long_F_impact(id5)/90 + 82/90;
670 %c6 = 0;
671 %for i = id5:id9
672 %    if lat_F_impact(i) >=5.55 && lat_F_impact(i) <= 9.51
673 %     c6 = c6 + (1 - (long_F_impact(i)-80.47)^2/132^2 - (
    lat_F_impact(i) - 7.53)^2/167^2);
674 %    elseif lat_F_impact >9.51
675 %    c6 = c6 + (long_F_impact(i)/90 - long_L/90)^2;
676 %    end
677 %end
678 %emp = find(lat_F_impact-6<0,1,'first');
679 %c6 = -long_F_impact(emp)/90 + 82/90;
680
681 %% 2) For Sequential Quagratic Programming
682 %(Uncomment this block and the above two below blocks for
    running this
683 %script as function for getting constraints and objective
    functions for fmincon)
684 %c=[c1, c2, c3];
685 %ceq = [c4, c5];
```