POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Laurea Magistrale in Ingegneria Meccanica

DEVELOPMENT OF AN OPTIMIZATION FRAMEWORK INTEGRATING GASDYN FOR TUNING INTERNAL COMBUSTION ENGINE PARAMETERS TO ENHANCE PERFORMANCE

Thesis supervisor: Prof. Gianluca D'Errico

Thesis advisor: Ing. Andrea Marinoni

Candidate:

Naveenkumar Murugavel

ID code 10918836

Academic Year 2024 - 2025

Abstract

The increasing demand for fuel efficiency and environmental protection presents notable challenges in the design of Internal Combustion Engines(ICEs). ICEs are very complex systems, and developing them involves more experimental testing, which is time-consuming and costly. To accelerate this process, various programs have been developed that apply statistical methods such as the Design of Experiments (DoE) to reduce the number of physical tests necessary for effective tuning.

This description includes this present thesis, which introduces an optimization framework *engine_optimizer*, written in Python language that integrates the fluid dynamics simulator *Gasdyn* and aims to identify the optimal engine parameters using the Mesh Adaptive Direct Search(MADS) algorithm that enhances the performance while minimizing the need for experimental validation. This tool was then used for the design of a single-cylinder engine of a motorcycle focusing on Intake Duct Sizing Optimisation which analyzed and optimized the intake duct length to enhance airflow and engine efficiency and another for Variable Valve Timing (VVT) optimization where the valve timing at various operating points were optimized to enhance combustion efficiency and reduced emissions. By simulating the engine with the optimized intake duct dimensions and VVT settings, significant improvements were observed in performance metrics with slightly considerable technical trade-offs.

These operations underscore the value of using simulation-based optimization tools in engine design as they not only accelerate the design process but also reduce costs by minimizing the dependence on physical prototypes and experimental testing.

Keywords: *engine_optimizer,* Objective, Optimization, Mesh Adaptive Direct Search(MADS), Internal Combustion Engine

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the complexity of the design of internal combustion engines (ICEs) being an elaborate system has been increasing due to meticulous requirements of fuel efficiency and emission reductions. Traditional engine tuning methods rely heavily on detailed bench testing where the operator starts from the basic configuration and varies every engine parameter and tests it, which is both time-consuming and costly to effectively design a modern Internal Combustion Engine.

To address these challenges, computational tools have been developed to simulate operations and foresee performance metrics precisely. However, a physically rigorous description of all phenomena happening inside the engine requires a very expensive fluid dynamic study, and for this reason, the simulators used are one-dimensional (1D), which provides accurate results while avoiding high calculation times. One such tool is *Gasdyn*, a one-dimensional thermo-fluid dynamic simulator written in FORTRAN language developed by the Department of Energy at the Politecnico di Milano. *Gasdyn* facilitates the simulation of unsteady reacting flows within the whole engine system and facilitates the rapid calculations of parameters such as volumetric efficiency, torque, power output, fuel consumption, pollutant emissions, and tailpipe noise.

Despite the capabilities of simulators like *Gasdyn*, the large number of parameters, variables, and constraints involved in engine design make manual optimization impractical; in fact, the simulator alone is not adequate to find the optimal one for the engine designer among the numerous configurations available. To navigate this complexity, optimization programs have been developed to interface with the simulators, which find the optimal configurations that meet the specific design objectives.

This thesis introduces *engine_optimizer*, an optimization framework written in Python language that integrates seamlessly with *Gasdyn*. Utilizing the Mesh Adaptive Direct Search (MADS) algorithm - a derivative-free optimization method effective for constrained optimization problems, *engine_optimizer* effectively identifies the optimal engine parameters while maintaining design constraints. The optimizer offers four optimizations that can be performed, which are Duct Length Optimization, Duct diameter Optimization, Valve timing Optimization & Air fuel ratio Optimization. The research conducted in this thesis involves two main parts: the first part provides a detailed explanation of the optimizer's functions, features, and optimization types, along with guidance on utilizing the framework, and the second part contains the demonstration and validation of this program by optimizing the intake duct sizing and Variable Valve Timing (VVT) of a single cylinder motorcycle engine.

Chapter 2

1D Engine Modelling

2.1 Introduction

In the ever-evolving landscape of internal combustion engine (ICE) development, engineers face the challenge of designing engines that are not only powerful and efficient but also environmentally friendly and cost-effective. Achieving this balance requires tools that can accurately predict engine behavior under various conditions, allowing for informed design decisions early in the development process. One such invaluable tool is one-dimensional (1D) engine modeling. Unlike three-dimensional (3D) computational fluid dynamics (CFD) simulations, which provide detailed insights but are computationally intensive, 1D modeling simplifies the engine's complex geometry into a series of interconnected components along a single dimension. This approach enables faster simulations while still capturing essential engine dynamics, making it particularly useful during the preliminary design and optimization phases.

Gasdyn, in a similar fashion, is a one-dimensional flow solver for entire internal combustion engine systems. In this thesis, all the simulated results are obtained through *Gasdyn*, which are discussed comprehensively in the following chapters. In this chapter, we will delve into the fundamentals of 1D engine modeling, exploring its scope, key features, and the advantages it offers over traditional methods. We will also discuss how integrating optimization techniques with 1D models can further enhance engine performance, providing a comprehensive understanding of this powerful simulation approach.

2.2 ID Conservation equation

In an internal combustion (IC) engine, the flow of gases is complex. These gases move through the system in an unsteady and compressible manner, constantly changing due to the

engine's operation. The flow is also highly turbulent, and friction between the gas and the engine walls plays a crucial role. As the gases pass through different parts of the engine, they experience variations in cross-sectional area, temperature, and pressure, making the flow behavior complex and highly dynamic.

While a full 3D analysis would give the most accurate picture of what's happening inside the engine, it requires a lot of computational power and time. To make things more efficient, the *Gasdyn* code simplifies this by working in 1D, using certain assumptions to reduce complexity. This approach allows *Gasdyn* to predict the flow behavior quickly while still providing results that are very close to reality. That's the key benefit of using such a tool.

At the core of the simulation are the Navier-Stokes equations, which include:

- Mass conservation equation
- Momentum conservation equation
- Energy conservation equation

To solve these equations, we also need an equation of state (such as the ideal gas law) that relates the fluid's pressure, temperature, and density. *Figure 1* shows a basic one-dimensional control volume, a small slice of the engine flow path with a length of dx.

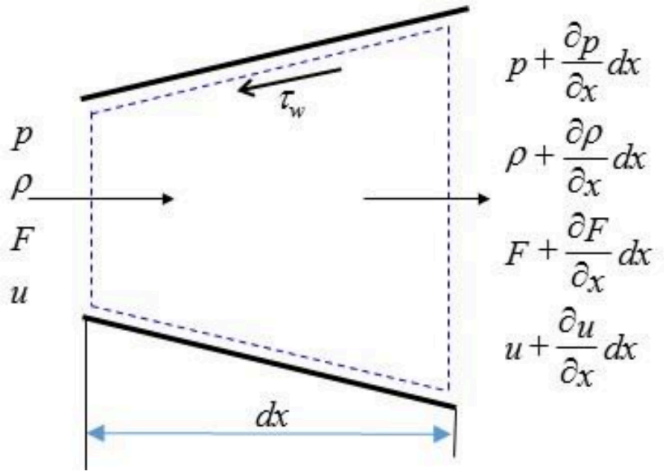

Figure 1 – Finite 1D control volume

Within this control volume, we track how properties like p, ρ, F, and u change as the gas

flows through. Here's what the key terms represent:

p: Pressure

ρ: Density

F: Cross-sectional area

u: Fluid velocity

τ_w: Wall shear stress

2.2.1 Mass conservation equation

The mass conservation equation is based on the principle that the rate at which mass flows into a control volume minus the rate at which mass flows out of the control volume must equal the rate of change of mass within the control volume itself. This can be mathematically expressed as:

$$(\rho + \frac{\partial \rho}{\partial x}dx)\left(u + \frac{\partial u}{\partial x}dx\right)\left(F + \frac{\partial F}{\partial x}dx\right) - \rho u F = -\frac{\partial}{\partial t}(\rho F dx) \tag{2.1}$$

Where:

- ρ: Fluid density
- u: Fluid velocity
- F: Cross-sectional area
- t: Time

This equation represents the balance between the mass flowing in and out of the control volume and the rate at which mass is accumulating within it. To simplify the equation and eliminate second-order infinitesimal terms (which become negligibly small as dx approaches zero), we can rearrange the terms:

$$\frac{\partial \rho}{\partial t} + \rho\frac{\partial u}{\partial x} + u\frac{\partial \rho}{\partial x} + \frac{\rho u}{F}\frac{\partial F}{\partial x} = 0 \tag{2.2}$$

In this form, the mass conservation equation expresses the change in density (ρ), velocity (u), and area (F) along the flow direction x, as well as the temporal variation of mass density within the control volume.

2.2.2 Momentum conservation equation

The momentum conservation equation expresses Newton's second law applied to a control volume, stating that the sum of external forces acting on the control surface equals the rate of change of momentum within the control volume.

Pressure Forces:

$$pF - \left(p + \frac{\partial p}{\partial x}dx\right)\left(F + \frac{\partial F}{\partial x}dx\right) + p\frac{dF}{dx}dx = -\frac{\partial(\rho F)}{\partial x}dx + p\frac{dF}{dx}dx \tag{2.3}$$

Shear Stress Forces:

The wall shear stress τ_w is related to the friction factor f by

$$\tau_w = f \cdot \frac{1}{2}\rho u^2 \tag{2.4}$$

The force due to wall friction is then:

$$F_{\text{friction}} = -f \cdot \frac{1}{2}\rho u^2 \cdot (\pi D dx) \tag{2.5}$$

Rate of Change of Momentum in the Control Volume:

$$\frac{\partial}{\partial t}(\rho u F dx) \tag{2.6}$$

Net Efflux of Momentum from the Control Surface:

$$\left(\rho + \frac{\partial \rho}{\partial x}dx\right)\left(u + \frac{\partial u}{\partial x}dx\right)^2\left(F + \frac{\partial F}{\partial x}dx\right) - \rho u^2 F \approx \frac{\partial}{\partial x}(\rho u^2 F)dx \tag{2.7}$$

Final Momentum Equation:

Combining the above terms and simplifying, we obtain the momentum conservation equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial \rho}{\partial x} + \frac{1}{\rho}\frac{\partial \rho}{\partial x} + G = 0 \tag{2.8}$$

Where:

$$G = f \cdot \left(\frac{u^2}{2}\right) \cdot \left(\frac{u}{|u|}\right) \cdot \left(\frac{4}{D}\right) \tag{2.9}$$

Here, G accounts for the viscous dissipation due to friction

2.2.3 Energy Conservation Equation

Applying the First Law of Thermodynamics to a control volume, the energy conservation equation is derived. This law states that the rate of change of energy within the control volume equals the net rate of energy transfer into the volume through heat and work interactions.

The general form of the energy conservation equation is:

$$\dot{Q} - \dot{L} = \frac{\partial E_0}{\partial t} + \frac{\partial H_0}{\partial x}\, dx \tag{2.10}$$

To obtain the final expression of the energy conservation equation, it is necessary to better specify the single terms. The specifications of the Terms are as follows.

- Heat Transfer Rate ($\dot{Q}$):

  The heat entering the control volume through the pipe walls is given by:

$$\dot{Q} = q \cdot \rho F\, dx + \Delta H_{\text{reaction}} \cdot F\, dx \tag{2.11}$$

Where:

- q: Heat exchanged per unit mass and time.
- ρ: Density of the fluid.
- F: Cross-sectional area of the control volume.
- $\Delta H_{\text{reaction}}$: Heat released per unit time and volume by chemical reactions occurring in the gas flow.

- Time Rate of Change of Total Stagnation Internal Energy ($\frac{\partial E_0}{\partial t}$):

The variation in time of the total stagnation internal energy is given by:

$$\frac{\partial E_0}{\partial t} = \frac{\partial}{\partial t} \left(e_0 \cdot \rho F \, dx \right) \tag{2.12}$$

Where:

- e_0: Specific stagnation energy.

- The specific stagnation energy is defined as:

$$e_0 = e + \frac{u^2}{2} \tag{2.13}$$

Where:

- e: Specific internal energy.
- u: Fluid velocity

- Net Flux of Stagnation Enthalpy Across the Control Surface ($\frac{\partial H_0}{\partial x}$):

The net flux of stagnation enthalpy through the control surface is given by:

$$\frac{\partial H_0}{\partial x} = \frac{\partial}{\partial x} \left(h_0 \cdot \rho F u \right) \, dx \tag{2.14}$$

Where:

- h_0: Specific stagnation enthalpy.

- The specific stagnation enthalpy is defined as:

$$h_0 = e_0 + \frac{p}{\rho} = c_v T + \frac{p}{\rho} + \frac{u^2}{2} \tag{2.15}$$

Considering the assumptions of a perfect gas and incorporating the speed of sound a, the energy conservation equation becomes:

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} - a^2\left(\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x}\right) - (\gamma - 1)\rho\,(q + u \cdot G) = 0 \tag{2.16}$$

Where:

- a: Speed of sound.
- γ: Ratio of specific heats ($\gamma = \frac{c_p}{c_v}$).
- c_p: Specific heat at constant pressure.
- c_v: Specific heat at constant volume.
- G: Viscous dissipation term.

The speed of sound a in a perfect gas is given by:

$$a = \sqrt{\gamma R T} \tag{2.17}$$

Where:

- R: Specific gas constant.
- T: Temperature

2.2.4 Conservative form of conservation equations

The system of conservation equations represents a hyperbolic problem, expressed in its non-conservative form.

$$\begin{cases} \frac{\partial \rho F}{\partial t} + \frac{\partial (\rho u F)}{\partial x} = 0 \\ \frac{\partial (\rho u F)}{\partial t} + \frac{\partial ((\rho u^2 + p) F)}{\partial x} - p\frac{\partial F}{\partial x} + \rho G F = 0 \\ \frac{\partial (\rho e_0 F)}{\partial t} + \frac{\partial (\rho u h_0 F)}{\partial x} - \rho q F = 0 \end{cases} \tag{2.18}$$

To simplify the resolution of the system, the equations can be reformulated in conservative form and subsequently expressed in matrix form. In this conservative form, the classical momentum equation is replaced by the impulse equation, which is derived as a linear combination of the mass and momentum conservation equations. This reformulation allows the grouping of terms into three distinct vectors, facilitating a more structured and efficient numerical implementation.

- Vector of conserved variables

$$\bar{W}(x,t) = \begin{bmatrix} \rho F \\ \rho F \\ \rho e_0 F \end{bmatrix} \tag{2.19}$$

  These three variables are independent; they vary with x and t, and their fluxes are conserved along the shockwave.

- Vector of conserved variable fluxes

$$\bar{F}(\bar{W}) = \begin{bmatrix} \rho u F \\ (\rho u^2 + p) F \\ \rho u h_0 F \end{bmatrix} \tag{2.20}$$

  Mass, impulse, and stagnation enthalpy fluxes are conserved through the boundaries.

- Vector of source terms

$$\bar{C}(\bar{W}) = \begin{bmatrix} 0 \\ -p\frac{dF}{dx} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \rho G F \\ -\rho q F \end{bmatrix} \tag{2.21}$$

  The source term is composed of contributions from two distinct vectors. The first vector accounts for source terms arising from variations in the cross-sectional area along the duct. The second vector represents source terms resulting from friction between the fluid and the duct walls, as well as heat exchange between the fluid and its surroundings.

Finally, the conservation equations can be written in the compact form:

$$\frac{\partial \bar{W}(x,t)}{\partial t} + \frac{\partial \bar{F}(\bar{W})}{\partial x} + \bar{C}(\bar{W}) = 0 \tag{2.22}$$

At this stage, the system consists of three equations with four unknowns: pressure (p), velocity (u), internal energy (e), and density (ρ). Therefore, a fourth equation is required to close the system and enable its solution. This additional equation is derived from the thermodynamic properties of the fluid. Typically, one may adopt either the ideal gas model with constant specific heats or a mixture of ideal gases approach, depending on the application.

In the case of gases in engine manifolds and cylinders, the ideal gas model is usually adopted, and the system is closed by the state equation of an ideal gas:

$$\frac{p}{\rho} = RT \tag{2.23}$$

Introducing the ideal gas law adds temperature (T) as a new variable to the system. However, for an ideal gas, both internal energy (e) and enthalpy (h) are functions of temperature. Therefore, additional equations must be incorporated into the system to relate these thermodynamic properties:

$$e = e(T) \tag{2.24}$$

$$h = e(T) + RT \tag{2.25}$$

In the case of a perfect gas, where specific heats at constant volume (c_v) and constant pressure (c_p) are assumed to be constant, these relationships simplify to:

$$e = c_v T \tag{2.26}$$

$$h = c_p T \tag{2.27}$$

By including these expressions for internal energy and enthalpy, the system becomes fully determined, consisting of six independent equations for six unknowns:

$$(p,\ u,\ e,\ \rho,\ h,\ T) \tag{2.28}$$

With appropriate boundary conditions, this closed set of equations enables the determination of the flow properties at any section along the duct.

2.3 Numerical methods

In general, it's not possible to find exact (analytical) solutions to the set of partial differential equations described above. Instead, numerical methods are used to approximate the solution. To do this, the continuous flow of fluid is divided into a finite number of small elements, creating a grid in both space and time. The governing equations are then converted into algebraic form, which computers can solve. This process is known as discretization. The accuracy and stability of the solution depend on the numerical method used. For many years, the Method of Characteristics was widely used. It provides first-order accuracy in space and time, but it struggles with discontinuities like shock waves.

To overcome this, it was later replaced by shock-capturing methods, which are better at handling abrupt changes in flow properties.

2.3.1 Shock-capturing numerical method

Gasdyn solves the conservation equations in conservative form using a shock-capturing method at each internal node of the pipes. As the fluid flows through the pipes, it experiences variations in temperature, pressure, and chemical composition and may encounter shock waves. These shock waves create contact discontinuities, which also appear in the numerical solution. The shock-capturing method is specifically designed to handle these sharp changes and still provide an accurate and stable solution. This method works by applying the same finite difference scheme at each node to approximate the spatial derivatives. It's based on the compact matrix form of the conservation equations and offers second-order accuracy in both space and time.

To apply this method, two key adjustments are made:

- The vector $\bar{W}(x,t)$ is replaced by W_i^n, meaning $W(i\Delta x, n\Delta t)$, where the indices i and n are referred to as space and time.
- The source vectors are deleted.

In this way, the equation X becomes:

$$\frac{\partial \bar{W}}{\partial t} + \frac{\partial \bar{F}(\bar{W})}{\partial x} = 0 \tag{2.29}$$

By integrating the vector X over a grid divided into Δx and Δt meshes, Y is obtained

$$\int_{t}^{t+\Delta t} \int_{x}^{x+\Delta x} \frac{\partial \bar{W}}{\partial t} + \frac{\partial \bar{F}(\bar{W})}{\partial x} dx dt = 0 \tag{2.30}$$

$$(\bar{W}_i^{n+1} - \bar{W}_i^n)\Delta x + \left(\bar{F}_{i+\frac{1}{2}}^{n+1} - \bar{F}_{i-\frac{1}{2}}^{n}\right)\Delta t = 0 \tag{2.31}$$

The space discretization for F at the calculation point "i" is done using central differencing, which is why a ½ weight is applied on each side. This approach helps avoid issues related to the derivability condition of the Euler Equations. By using the integral form, the method can capture contact discontinuities in the flow without needing to know their exact position.

By rearranging the expression, we get a new form that represents the discretized version of the continuous equation. This is the version used in the finite difference method during numerical implementation.

$$\frac{(\bar{W}_i^{n+1} - \bar{W}_i^n)}{\Delta t} + \frac{\left(\bar{F}_{i+\frac{1}{2}}^{n+1} - \bar{F}_{i-\frac{1}{2}}^{n}\right)}{\Delta x} \tag{2.32}$$

Summing the differences along x:

$$\Delta x \sum_{i,min}^{i,max} \bar{W}_i^{n+1} = \Delta x \sum_{i,min}^{i,max} \bar{W}_i^n + \Delta t \left(\bar{F}_{i+\frac{1}{2}}^{n+1} - \bar{F}_{i-\frac{1}{2}}^{n}\right) \tag{2.33}$$

The expression above means that the total mass, momentum, and energy at time step $n+1$ is equal to the amount at time step n, plus the net fluxes that have passed through the ends of the duct. This approach ensures that the conservation laws are respected, making it a conservative discretization scheme.

Specifically, *Gasdyn* uses Lax-Wendroff and MacCormack shock-capturing techniques to solve the equations. However, the detailed explanation of these methods goes beyond the scope of this thesis.

2.4 Scope and Main Features of 1D Engine Modelling

1D modelling simplifies the complex three-dimensional nature of engine flows by representing the system as interconnected components such as cylinders, ducts, valves, turbines, compressors, and after-treatment devices. Each component is solved along a single spatial dimension, typically the direction of flow, while still capturing unsteady thermodynamic and fluid dynamic interactions. This approach offers a good balance between simulation accuracy and computational speed, making it ideal for parametric studies, control development, and virtual calibration. *Gasdyn* allows users to construct, simulate, and analyze virtual engine models ranging from simple silencer systems to full engine configurations with turbochargers, EGR systems, and after-treatment components.

2.4.1 Key Scopes of 1D Engine Modelling Using *Gasdyn*

1D engine modelling with *Gasdyn* supports a wide range of applications, including but not limited to:

- Complete Engine System Simulation: From single-cylinder research engines to full multi-cylinder powertrains, users can simulate both naturally aspirated and forced induction systems, and it will compute the engine torque, power output, volumetric efficiency, and transient behavior across a range of operating conditions.

- Thermodynamic Simulation: Involves modelling combustion cycles for both spark-ignition (SI) and compression-ignition (CI) engines, accounting for heat transfer, gas exchange processes, and turbocharging systems. These simulations predict flow-related pressure and temperature fluctuations throughout the engine, which are crucial for performance optimization and thermal management.

- Exhaust and Intake Acoustic Analysis: Focuses on simulating unsteady pressure wave dynamics within intake and exhaust systems. This enables the design of efficient intake manifolds and silencers, and supports the prediction of acoustic transmission loss and noise generation, contributing to improved engine acoustics and compliance with noise regulations.

- Transient and Steady-State Analysis: Supports both steady-state and transient simulations, enabling the evaluation of engine performance under controlled

conditions (fixed-load, speed sweeps) and dynamic scenarios such as standardized driving cycles (e.g., WLTP, WMTC). This flexibility helps in understanding both average and time-dependent behaviors of engine systems.

- Control-Oriented Modelling: Facilitates integration with control systems to virtually test and calibrate engine management strategies, including throttle control, EGR (Exhaust Gas Recirculation), VGT (Variable Geometry Turbocharging), and others. *Gasdyn* can generate virtual sensor signals and dynamic responses, supporting applications such as model-based calibration, real-time simulation, and embedded control development.

- After-Treatment System Design: Simulates the performance of emission control components such as diesel particulate filters (DPF), catalytic converters, and selective catalytic reduction (SCR) systems. The model accounts for thermal loading and pollutant conversion efficiency while analyzing the formation and mitigation of emissions like NOx, CO, unburned hydrocarbons (HC), and soot.

- Coupled Vehicle Simulations: Enables co-simulation with external platforms like MATLAB/Simulink for full-vehicle system integration. This includes the influence of gear shifting strategies, vehicle mass, rolling resistance, and aerodynamics. *Gasdyn* supports coupling with longitudinal vehicle dynamics tools, allowing the assessment of drivability, transient response, and fuel consumption over real-world driving cycles such as WLTP, NEDC, and WMTC.

2.4.2 Features of *Gasdyn*: A Versatile 1D Engine Simulation Tool

Gasdyn is a high-fidelity 1D simulation environment designed for modelling complete engine systems with an emphasis on transient and wave dynamics. It combines ease of use with advanced technical depth, allowing users to build virtual engines and run simulations efficiently.

- Comprehensive Library of Engine Components

  Gasdyn provides an extensive set of engine components that can be graphically assembled within its interface. These include inlet and outlet boundary conditions, single and multi-cylinder engine models, intake and exhaust ducts, junctions, plenum

volumes, turbochargers (turbines and compressors), wastegates, and exhaust gas recirculation (EGR) valves. Users can also integrate throttle bodies, fuel injectors, catalytic converters, diesel particulate filters, and silencer elements such as expansion chambers and perforated pipes. Each component is customizable, allowing detailed definitions of geometry, material properties, thermodynamics, and flow behavior. This flexibility supports a wide range of engine architectures and configurations.

- Support for Both Steady-State and Transient Simulations

  Gasdyn supports multiple types of simulation modes to accommodate diverse analysis needs. In steady-state simulations, the software is used to generate full-load curves, develop engine performance maps, and support design optimizations under constant operating conditions. In contrast, engine transient simulations allow engineers to replicate real-world operating scenarios such as throttle response, gear shifting, and acceleration events. Furthermore, *Gasdyn* supports vehicle-level transient simulations through integration with standard driving cycles like WLTP and WMTC. This enables the modeling of engine behavior within complete vehicle systems, aiding in fuel economy and emissions assessments under realistic conditions.

- Advanced Combustion and Heat Transfer Models

  The software incorporates robust models for combustion and heat transfer, supporting both spark-ignition (SI) and compression-ignition (CI) engines. For SI engines, Wiebe-function-based combustion models simulate the rate of heat release, while CI engines benefit from detailed combustion modeling approaches. *Gasdyn* also integrates empirical and semi-empirical pollutant formation models, allowing the estimation of NOx, soot, and other key emissions. Additionally, users can analyze in-cylinder pressure traces using the Heat Release Rate (HRR) tool, often informed by experimental data. Heat transfer models account for wall heat fluxes, surface temperatures, and coolant interactions, providing insights into engine thermal management.

- Acoustic Simulation of Silencers and Exhausts

  Acoustic behavior in intake and exhaust systems is another key focus area of *Gasdyn*. Using both time-domain and frequency-domain approaches, the software predicts key acoustic parameters such as Transmission Loss (TL), Transfer Function (TF), and acoustic impedance. These simulations are critical for optimizing mufflers, silencers, and air intake systems, helping reduce engine noise while maintaining performance. Such capabilities are valuable in both motorsport and passenger vehicle applications, where noise, vibration, and harshness (NVH) characteristics are crucial.

- Turbocharging and Boost System Simulation

  Gasdyn includes advanced modelling tools for turbocharging systems. It supports the simulation of fixed geometry and variable geometry turbines (VGTs), along with associated compressors. The software offers pre-processing tools to convert raw manufacturer map data into formats suitable for simulation, including curve fitting for efficiency and performance. Additional components like intercoolers and charge-air cooling systems can be added to complete the boost circuit. *Gasdyn* also allows the modelling of actuator dynamics and closed-loop control systems using PID controllers, enhancing accuracy in boost pressure and EGR control simulations.

- After-Treatment Systems and Emissions Control

  The software provides detailed models for exhaust after-treatment systems, including catalytic converters, diesel particulate filters (DPFs), and EGR setups. These models consider chemical kinetics, substrate and washcoat characteristics, thermal responses of the exhaust system, and pressure losses across components. With these capabilities, *Gasdyn* can simulate cold-start behavior and assess the effectiveness of emissions reduction strategies under various operating conditions, supporting the development of compliant and efficient powertrains.

- Graphical Pre- and Post-Processing

  Gasdyn simplifies the simulation workflow with two dedicated graphical tools: *Gasdyn*Pre and *Gasdyn*Post. *Gasdyn*Pre offers a drag-and-drop interface for setting up the engine architecture, defining boundary conditions, and configuring component

parameters. This interface is intuitive and requires minimal training, making it accessible for users with varying experience levels. Post-processing is handled via *Gasdyn*Post, which allows users to visualize simulation results such as pressure, temperature, velocity, and emissions in both time and frequency domains. The tool supports comparative analysis between simulation cases, enabling efficient design evaluations and result interpretation.

- Co-Simulation with MATLAB/Simulink and Axisuite

  Gasdyn can be integrated into co-simulation environments for real-time and hardware-in-the-loop (HiL) applications. It supports communication with MATLAB Simulink via Co-Simulation Input/Output blocks, allowing engine models to be linked with vehicle dynamics, control algorithms, or real-time data acquisition systems. Additionally, *Gasdyn* is compatible with Axisuite for advanced emissions modeling, enabling the simulation of complex after-treatment systems and their interactions with the engine in dynamic conditions. This makes *Gasdyn* suitable not only for offline simulation but also for embedded systems development and validation.

- Multi-Scenario and Parallel Simulation Capability

  To enhance productivity, *Gasdyn* includes features for running batch simulations across a range of engine speeds, loads, and control parameters. These multi-scenario simulations are useful for generating engine performance maps, conducting parametric sensitivity analyses, and validating control strategies. Parallel processing further reduces simulation time, making it feasible to explore large design spaces or optimize system performance within shorter development cycles.

2.5 Literature review: Optimization Tools Coupled with 1D Engine Models

The integration of optimization tools with one-dimensional (1D) engine models has become a cornerstone in modern engine development, facilitating efficient exploration of design spaces and enabling the identification of optimal solutions that balance multiple performance objectives.

1D engine models have evolved from basic thermodynamic representations to sophisticated computational fluid dynamics (CFD) tools. These models simulate engine components as interconnected elements with one-dimensional flow, allowing for efficient system-level analysis while capturing essential flow dynamics and thermodynamic processes with reasonable computational costs. Their primary advantage lies in the ability to simulate complete engine systems with acceptable accuracy, requiring significantly less computational resources than 3D CFD approaches, making them particularly valuable for optimization studies involving numerous iterations.

Coupling optimization algorithms with 1D engine models enables engineers to systematically explore design spaces, address conflicting objectives, and reduce development time and costs. This integration allows for methodical exploration of multiple design variables simultaneously, helping to identify optimal compromises between competing goals such as performance, emissions, and fuel economy. Virtual optimization significantly reduces the need for physical prototyping and testing, accelerating the development cycle.

Several optimization methodologies have been successfully coupled with 1D engine models. Single-objective optimization focuses on optimizing a single performance metric, such as power output or fuel efficiency, and remains relevant for applications where a particular aspect of engine performance is prioritized. Multi-objective optimization addresses several performance criteria simultaneously, generating Pareto-optimal solutions that represent the best possible trade-offs between competing objectives. Design of Experiments (DoE) techniques help identify the most significant design parameters and their interactions, allowing engineers to focus optimization efforts more efficiently by reducing the dimensionality of the problem.

The applications of coupled 1D models and optimization tools are diverse. In engine performance optimization, researchers have enhanced various aspects such as intake and exhaust system design, combustion optimization, and turbocharging systems. For instance, optimizing manifold geometry, valve timing, and port design improves volumetric efficiency and reduces pumping losses. Tuning combustion parameters balances power output with emissions control, while optimizing turbocharger matching and control strategies enhances transient response and efficiency across operating ranges.

Emissions reduction is another critical application area, especially with increasingly stringent regulations. 1D models coupled with optimization tools assist in optimizing Exhaust Gas Recirculation (EGR) strategies to reduce NOx emissions without excessive fuel economy penalties and in designing aftertreatment systems by sizing and configuring catalytic converters and particulate filters for optimal emissions conversion efficiency.

Optimizing transient operation has gained importance due to modern driving cycles emphasizing real-world vehicle operation. This includes developing control strategies for improved transient response during acceleration and load changes and integrating hybrid powertrains to optimize engine operation within hybrid systems for overall efficiency.

Advanced coupled approaches have been developed to overcome limitations of pure 1D models while maintaining computational efficiency. Sequential 1D-3D coupling uses 1D models for system-level optimization, then validates and refines results with targeted 3D CFD analysis of complex flow domains. Co-simulation involves simultaneously running 1D models for system-level behavior with 3D models for specific components, where detailed flow analysis is crucial.

Recent advancements have seen the integration of machine learning techniques with 1D model-based optimization. Surrogate modeling uses machine learning to create computationally efficient models based on 1D simulation results, enabling faster optimization iterations. Adaptive optimization strategies employ machine learning to guide the optimization process based on previous simulation results for enhanced convergence and robustness.

Despite significant progress, challenges remain in the field of optimization with 1D engine models. While 1D models are more efficient than 3D CFD approaches, optimization still requires numerous iterations, necessitating the development of faster simulation techniques. Ensuring that 1D models accurately represent complex physical phenomena, particularly during transient operation, is an ongoing challenge. Developing optimization strategies that account for uncertainties in model parameters and operating conditions is crucial for real-world applicability in noisy experimental scenarios.

Emerging trends in the field include real-time optimization methods fast enough for real-time control optimization, multi-fidelity approaches that integrate models of varying fidelity at different stages of the design process, and digital twin integration that couples engine optimization with digital twin technology for continuous improvement throughout the engine lifecycle.

The integration of optimization tools with 1D engine models represents a powerful approach for modern engine development, enabling efficient exploration of complex design spaces and identification of optimal solutions balancing multiple competing objectives. As computational capabilities advance and optimization algorithms become more sophisticated, this integrated approach will continue to evolve, playing an increasingly important role in addressing the challenges of future internal combustion engine design.

Chapter 3

Engine Optimizer

3.1 Advantages of the Optimizer

The engineering design field has been transformed and benefited numerously by the adoption of fluid dynamic simulators. These advanced computational tools, supported by increasingly powerful computing resources, foster engineers to precisely predict and analyze fluid behavior within complex systems. This capability of the simulators enables virtual prototyping, which allows the evaluation and refining of designs in a digital environment before the physical prototypes are constructed. Utilizing these capabilities makes the design process efficient and reduces the time and costs associated with traditional experimental testing. The use of simulators such as *Gasdyn* has become of fundamental importance in the design of an engine. After developing and validating the engine's numerical model, which includes all the necessary geometric and thermodynamic data necessary for the simulation, it's essential to conduct a series of tests to find the optimal configuration.

Each cycle of simulation requires three steps to be carried out:

1. **Modifying Optimization Variables:** Update with the specific parameters identified for optimization intended for this specific cycle.

2. **Initiating the Simulation:** Launch the simulation with the modified parameters set to observe the engine's performance under these conditions

3. **Evaluating Results:** Analyze the output data to understand how these modified parameters influence the performance metrics and overall engine behavior.

By following the above steps, we will get the performance metrics for the combination for which the simulation is run. But to find the optimal configuration, it is necessary to test them with the range of variables, and it's evident that this requires a lot of time, especially when the test variables and the parameters are numerous. To understand how time-consuming this process can be, let's look at a normal engine optimization case. Consider an engine where we need to test how changing four different variables affects performance using 3 levels for each variable within the range of interest. In this case, the number of simulations to be run for the range of values is 4^3 = 64 simulation cycles and this count is only for testing at a single rpm of the engine consider if we want to test these variables for five different rpms means then the number of simulations will reach 64 * 5 = 320 simulations tests. These tests are just the starting point to characterize the engine behavior, but when more parameters come into play, it is not certain that the optimal configuration is among them. So it is necessary to perform additional simulations to find the ideal combination that truly meets our intended performance metrics. So, as the number of variables increases, the total number of tests needed grows exponentially.

Now it is easy to understand that relying on simulation software alone isn't enough to make the design process efficient and fast. To address this, the *engine_optimizer* optimization framework was developed and implemented, which is interfaced with the fluid dynamics simulator, speeding up the simulation process and the search for optimal configuration. However, this optimizer doesn’t reduce the role of the designer. Instead, it accelerates the testing process and helps quickly find the optimal values within the variable ranges, ultimately leading to the ideal configuration. Since the technical feasibility of the engine must also be considered for the optimal values we got from the optimizer, supervision is required. It's up to the designer to identify and understand which parameters have the most significant impact on engine performance. This approach helps avoid unnecessary tests, focusing only on variables that significantly impact performance. It also prevents testing configurations that are physically unfeasible, as they would violate the structural constraints and design limitations set from the beginning. Therefore, anyone using this optimizer should have a clear understanding of the problem's characteristics to guide the analysis in the right direction and achieve the best results in the shortest time.

3.2 The Interaction Between *Gasdyn* and *engine_optimizer*

The interaction between the *engine_optimizer* written in Python language and *Gasdyn* written in FORTRAN language requires a robust system that ensures accurate variables between these two programs. During each optimization step, *engine_optimizer* performs two key tasks:

- Updating Engine Variables: Before initiating a new simulation, *engine_optimizer* modifies the engine parameters inside the *Input_gen.dat* file.

- Reading simulation outputs: The optimizer reads the *Gadsyn's* numerical results after the simulation

The second task is straightforward since the output results from the *Gasdyn* are as *.csv (Comma-separated values)* files, which are structured in a tabular format allowing *engine_optimizer* to parse and analyze the data efficiently, and in fact, Python can easily read *.csv* files.

It is more complex to automatically transfer the new variable from the optimizer to *Gasdyn* for successive simulations. e*ngine_optimizer* must be adaptable to various engine models that can handle different positions and tabulations of variables within input files. Each engine model will have its unique configuration, leading to variations in the structure and content of the *Input_gen.dat* file. Due to this variability, *engine_optimizer* must be smart and adaptable, capable of identifying and updating the right parameters automatically, without needing manual input each time.

This exchange of variables is facilitated by the "INDATI" subroutine within *Gasdyn,* which recognizes the "PUT_VAR" line inserted by the *engine_optimizer* in the Input_gen.dat file. If these lines are not present, *Gasdyn* runs independently from *engine_optimizer* using the default parameters specified in the Input file, allowing for standalone operation when required.

The process of exchange of variables to be optimized between the *Gasdyn and engine_optimizer is* illustrated below:

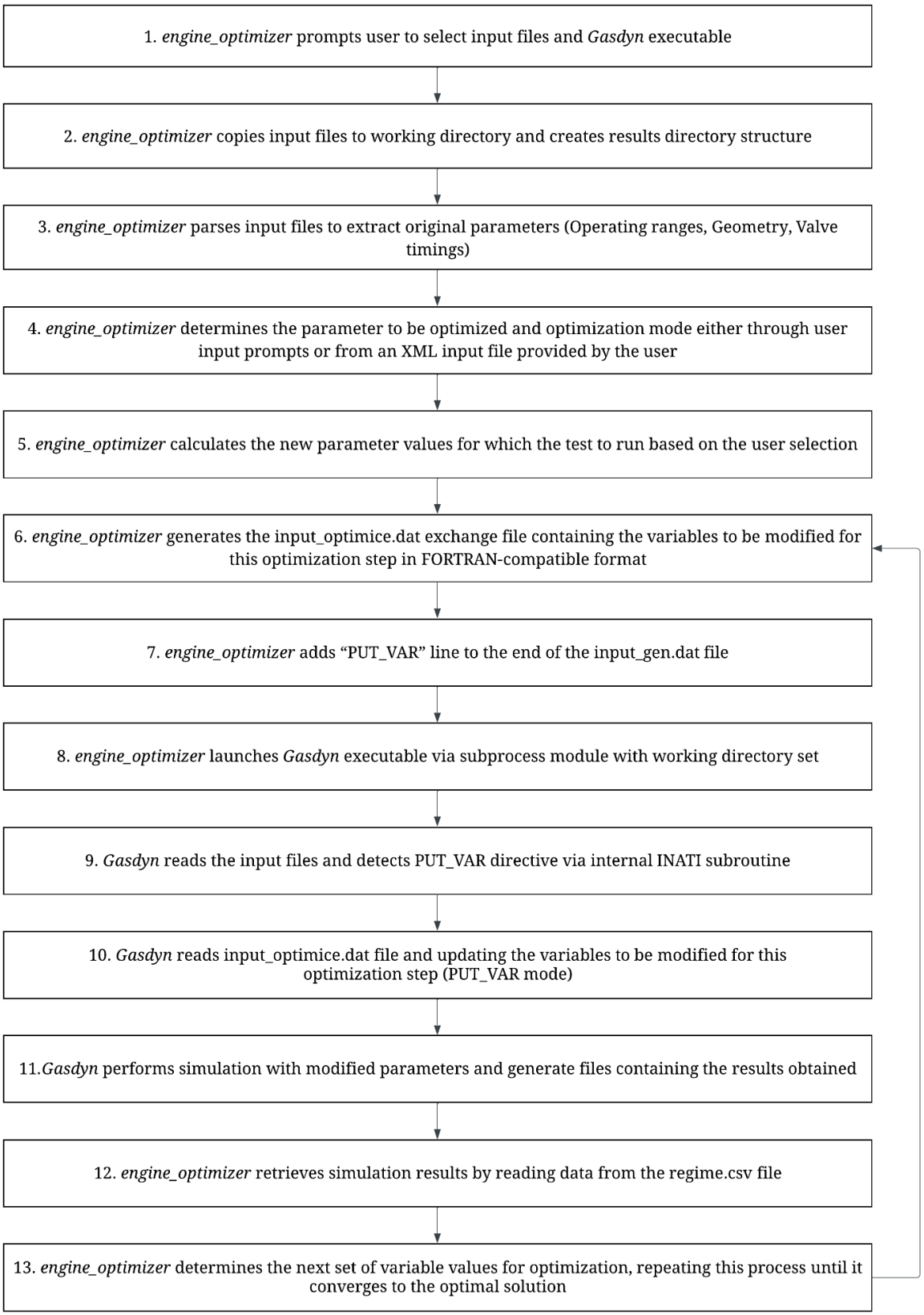

Figure 2 – Phases of the variable exchange process between *Gasdyn* and *engine_optimizer*: Steps 1 to 5 occur during the initialization of engine_optimizer, while Steps 6 to 13 are repeated for each simulation performed by the optimizer

3.3 Calculation accuracy and simulation storage

In the scope of engine optimization, it is essential to achieve a balance between computational accuracy and practical efficiency. The *engine_optimizer* framework addresses this by providing users the option to define the tolerance for each variable and objective within the simulation, which means the user can specify the number of decimal places to which each quantity should be rounded. This feature helps in aligning with real-world measurements, where in experimental settings, measuring devices can't capture tiny details beyond a certain point, so by setting the simulation to the same level of detail, the results will be more realistic and closer to the real-time measurement. Another important contribution of this feature is the reduction of computational errors because numerical simulations often involve discretization and rounding, which can introduce minor errors; therefore, controlling the precision helps avoid these errors and makes the results more reliable.

Moreover, *engine_optimizer* contains an archive where the previously executed simulations, along with their input configurations, are saved, and before initiating a new simulation, the optimizer checks the archive to know whether it is already been performed, and if a match is found, then it prevents the simulation. This control procedure requires a series of additional operations that offer time efficiency, especially when engine models get complex, simulations can become time-intensive, so by preventing repeated computations, the optimization is performed faster. These features in *engine_optimizer,* like controlling precision and intelligent simulation archiving, work together to make the optimization process smoother, ensuring that simulations stay both accurate and efficient.

3.4 Mesh Adaptive Direct Search (MADS)

When it comes to optimizing complex engineering systems like internal combustion engines, traditional optimization methods that rely on gradients often fall short. This is primarily because engine simulations are typically "black boxes"; we can observe the inputs and outputs, but the internal workings are opaque, and analytical derivatives are unavailable. Moreover, performance metrics such as volumetric efficiency, torque, and brake-specific fuel consumption (BSFC) can be non-smooth or discontinuous, making gradient-based methods unreliable. Additionally, these simulations are computationally expensive and susceptible to numerical noise, further complicating the optimization process.

To tackle these challenges, the Engine Optimizer employs the Mesh Adaptive Direct Search (MADS) algorithm. MADS is a derivative-free optimization method specifically designed for scenarios where the objective function is complex, lacks smoothness, or is computationally expensive to evaluate. It systematically explores the design space without relying on gradient information, making it well-suited for optimizing engine design parameters. Its adaptability allows it to handle various optimization scenarios, including duct length, valve timing, air-fuel ratio, and diameter optimization.

The choice of MADS is further supported by its proven effectiveness in engineering applications. For instance, studies have demonstrated its capability to efficiently handle multiobjective black-box optimization problems, where traditional methods struggle due to the lack of derivative information. Additionally, MADS has been successfully integrated with surrogate models and parallel computing techniques to enhance its performance in computationally intensive tasks. In summary, the Engine Optimizer's integration of the MADS algorithm provides a robust and efficient framework for optimizing engine design parameters, effectively navigating the complexities inherent in engine simulations.

The Optimizer implements the Mesh Adaptive Direct Search (MADS) algorithm through the following steps:

- Initialization
- Search step
- Poll step
- Mesh Management
- Simulation Integration
- Termination
- Result Processing

3.4.1 Initialization

The Mesh Adaptive Direct Search (MADS) algorithm begins its optimization process by discretizing the search space through the construction of a mesh. At the k -th iteration, this mesh, denoted as M_k , is defined by the equation:

$$M_k = \{x_k + \Delta_k^m D z \ : \ z \in \mathbb{N}^n\} \tag{3.1}$$

In this formulation, x_k represents the current iterate, Δ_k^m is the mesh size parameter, D is a matrix whose columns form a positive spanning set, and z is an integer vector. This mesh structure enables the algorithm to systematically explore the search space by evaluating points that are discrete combinations of the current iterate and scaled direction vectors.

From a practical standpoint, the initialization phase involves several critical steps to set up the optimization environment effectively. Initially, input files are read to extract essential parameters, including variable bounds and problem dimensions. Subsequently, from the Design of Experiments (DoE) results, techniques are employed to generate initial data points, which are instrumental in constructing preliminary interpolation models of the objective function. These models provide a foundational understanding of the search space's landscape, guiding the algorithm's exploratory moves.

The initial mesh size Δ_0^m is typically set in proportion to the ranges of the decision variables, ensuring that the initial search is neither too coarse nor too fine. This proportional setting facilitates a balanced exploration, allowing the algorithm to make meaningful progress in the early stages. Moreover, appropriate bounds and problem dimensions are configured to define the feasible region accurately, ensuring that all subsequent evaluations remain within acceptable limits.

By meticulously establishing these initial conditions, MADS is well-prepared to commence its iterative search process, systematically refining the mesh and honing in on optimal solutions through successive evaluations.

3.4.2 Search step

The search step is an optional yet flexible component that aims to identify improved solutions by evaluating trial points selected from the current mesh, M_k. These trial points can be chosen using various strategies, including heuristics or surrogate models.

One effective approach to selecting these trial points is through Latin Hypercube Sampling (LHS). LHS is a statistical method designed to generate a sample of plausible values from a

multidimensional distribution. It partitions each input distribution into intervals of equal probability and selects one sample from each interval, ensuring that the sample points are spread more evenly across the parameter space compared to pure random sampling.

In the context of MADS, LHS can be employed to explore promising regions of the parameter space by generating points both locally around the current best point and globally for broader exploration. This dual approach allows the algorithm to refine its search in areas known to be promising while also investigating new regions that might lead to better solutions. Moreover, specialized sampling strategies can be applied for each parameter type, such as duct length or valve timing, to account for their unique characteristics and constraints. By tailoring the sampling process to the specific nature of each parameter, the algorithm can more effectively navigate the search space.

To evaluate the generated points efficiently, interpolation models can be used. These models approximate the objective function, allowing for quicker evaluations and thus saving on computational resources. By integrating LHS with interpolation models, MADS can perform a more comprehensive and efficient search for optimal solutions.

3.4.3 Poll step

The poll step serves as a systematic approach to explore the immediate vicinity of the current solution, especially when the preceding search step fails to identify a better candidate. This step is crucial for ensuring local exploration in all directions, thereby enhancing the robustness of the optimization process. At iteration k, the poll set P_k is defined as:

$$P_k = \{x_k + \Delta_k^p d : d \in D_k\} \tag{3.2}$$

Here, x_k represents the current iterate, Δ_k^p is the poll size parameter, and D_k is a finite set of polling directions that form a positive spanning set. This configuration ensures that the algorithm evaluates points in all directions around x_k, facilitating a comprehensive local search.

In practical terms, implementing the poll step involves generating poll points systematically around the current best point. The direction vectors $d \in D_k$ are chosen to suit the specific

characteristics of each parameter type. For instance, step sizes are tailored to the physical meanings of the parameters:

Variables & Objectives	Accuracy
Engine speed [rpm]	1
Duct length [m]	0.01
Duct diameter [m]	0.001
IVO & EVO [°]	1
A/F Ratio	0.1
Torque [Nm]	0.1
Volumetric efficiency [%]	0.1
BSFC [g/kWh]	0.5

Table 1 – Tolerance Values for Variables & Objectives

These carefully selected step sizes ensure that the poll points are both meaningful and within feasible bounds, thereby enhancing the efficiency of the optimization process. By evaluating the objective function at these poll points, the algorithm can identify potential improvements. If a better solution is found, the current iteration is updated accordingly. If not, the algorithm may adjust the poll size parameter Δ_k^p to refine the search in subsequent iterations. Overall, the poll step in MADS is a structured and adaptive mechanism that ensures thorough local exploration, contributing significantly to the algorithm's ability to navigate complex optimization landscapes.

3.4.4 Mesh Management

Mesh management plays a pivotal role in balancing exploration and exploitation during the optimization process. The algorithm dynamically adjusts the mesh and poll sizes based on the success or failure of recent iterations, ensuring efficient navigation through the search space. Mathematically, the adjustment rules are as follows:

- **When an improved point is found**:

$$\Delta_{k+1}^{m} = \tau \Delta_{k}^{m}, \quad \Delta_{k+1}^{p} = \tau \Delta_{k}^{p} \tag{3.3}$$

- **When no improvement is found**:

$$\Delta_{k+1}^{m} = \frac{\Delta_{k}^{m}}{\tau}, \quad \Delta_{k+1}^{p} = \frac{\Delta_{k}^{p}}{\tau} \tag{3.4}$$

Here, Δ_k^m and Δ_k^p represent the mesh and poll sizes at iteration k, respectively, and $\tau > 1$ is a predefined scaling factor.

In optimizer, this means that after a successful iteration, where a better solution is identified, the algorithm expands the mesh and poll sizes by multiplying them with τ (commonly set to 2). This expansion allows the algorithm to explore the search space more broadly, potentially accelerating progress towards the optimal solution. Conversely, if an iteration does not yield an improvement, the algorithm contracts the mesh and poll sizes by dividing them by τ (commonly set to 0.5). This contraction focuses the search on a more localized region, refining the exploration around promising areas.

To prevent the algorithm from overshooting or becoming too conservative, practical implementations often impose limits on the maximum and minimum mesh sizes. This ensures that the search remains within feasible bounds and maintains a balance between global exploration and local exploitation.

The mesh refinement process continues iteratively, adapting to the landscape of the objective function, until predefined termination criteria are met. These criteria could include reaching a maximum number of iterations, achieving a satisfactory objective function value, or observing negligible improvements over successive iterations. Through this adaptive mesh management strategy, MADS effectively navigates complex optimization landscapes, balancing the need for global exploration with the precision of local search.

3.4.5 Simulation Integration and Caching

In simulation-based optimization, particularly when employing the Mesh Adaptive Direct Search (MADS) algorithm, integrating simulations efficiently is crucial due to the high computational cost of evaluating complex models. To address this, MADS incorporates surrogate models and caching mechanisms, which significantly reduce the computational burden associated with objective function evaluations.

Surrogate models serve as approximations of the true objective function, constructed from previously evaluated data points. These models, which can be based on techniques like regression, neural networks, or radial basis functions, provide quick estimates of the objective function's value for new candidate solutions. By using these approximations, the algorithm can efficiently explore the search space without the need for expensive simulations at every step. This approach is particularly beneficial in the search step of MADS, where numerous candidate points are evaluated to identify promising regions in the parameter space.

Caching is another vital component in simulation integration. By storing the results of previously evaluated points, the algorithm can avoid redundant simulations, thereby saving computational resources. When a candidate solution is proposed, the algorithm first checks the cache to see if the point has already been evaluated. If it has, the stored result is used; if not, the simulation is run, and the new result is added to the cache for future reference.

The integration of surrogate models and caching within MADS allows for a more efficient optimization process, enabling the algorithm to handle complex, expensive simulations effectively. The results are extracted from the simulation output files, and treating those as the initial points, the caching is applied, and the MADS is performed. This approach ensures that computational resources are utilized judiciously, focusing on evaluating the most promising candidate solutions and avoiding unnecessary simulations.

3.4.6 Termination

Determining when to terminate the optimization process is crucial for balancing computational efficiency with solution accuracy. The algorithm employs several termination criteria to decide when to stop iterating:

- **Mesh Size Threshold**: The algorithm monitors the mesh size parameter, denoted as Δ_k^m , at each iteration k. If this parameter becomes smaller than a predefined threshold $\Delta_{\min}$, it indicates that the search is sufficiently refined, and further iterations may yield negligible improvements. Mathematically, the termination condition is:

$$\Delta_k^m < \Delta_{\min} \tag{3.5}$$

- **Maximum Number of Iterations**: To prevent excessive computation, a maximum number of iterations $k_{\max}$ is set. Once the algorithm reaches this limit, it terminates regardless of the current solution's quality.

- **Objective Function Improvement Tolerance**: The algorithm tracks the improvement in the objective function value between successive iterations. If the improvement falls below a certain tolerance ϵ over a specified number of iterations, it suggests convergence to a local optimum, prompting termination. This can be expressed as:

$$|f(x_{k+1}) - f(x_k)| < \epsilon \tag{3.6}$$

  for a consecutive number of iterations.

- **Target Objective Value Achievement**: In target-based optimization scenarios, the algorithm stops when the objective function value reaches or surpasses a predefined target value f_{target}.

In the context of an engine optimization application, these termination criteria are implemented, and by this incorporation, the MADS algorithm ensures a balance between computational efficiency and the attainment of optimal or near-optimal solutions in complex optimization tasks.

3.4.7 Result Processing

The final phase involves processing and interpreting the results to extract meaningful insights. This step is crucial in translating the computational findings into actionable information, especially in applications like engine optimization.

Optimal Parameter Identification: The algorithm provides the best-found solution x^* , which represents the set of decision variables yielding the optimal objective function value within the defined constraints and mesh resolution. This solution is the culmination of iterative evaluations and adjustments, reflecting the most favorable configuration discovered during the optimization process.

Performance Metrics: To assess the quality and effectiveness of the optimal solution, various performance metrics are computed. These may include the final objective function value $f(x^*)$, convergence rate, computational time, and the number of function evaluations. Such metrics provide a quantitative measure of the optimization process's efficiency and the solution's robustness.

Visualization: Visual tools are employed to gain deeper insights into the optimization landscape and the algorithm's behavior, which will be explained in the later part. Response surfaces illustrate how changes in decision variables affect the objective function, while optimization trajectories depict the path taken by the algorithm through the search space.

Result Storage: The final results, including the optimal parameters, performance metrics, and visualizations, are systematically stored in structured formats such as files, which will be briefed in the following sections. This organization facilitates easy retrieval for reporting, further analysis, or future reference.

In optimizer, this comprehensive result processing ensures that the optimal engine parameters are not only identified but also thoroughly evaluated and documented, providing a solid foundation for informed decision-making and subsequent implementation. The results folder will consist of a file known as "*final_results.csv*" where the DoE results for each test variable will be saved, and these will be explained in the later part the MADS search point as explained in *Section 3.4.2* performed will also be printed by the optimizer and it is is illustrated in *Figure 3*.

```
MADS search,7500.0,0.23,MADS_104,98.0
MADS search,7500.0,0.55,MADS_105,94.6
MADS search,7500.0,0.38,MADS_106,96.5
MADS search,7500.0,0.54,MADS_107,94.8
MADS search,7500.0,0.08,MADS_108,99.5
MADS search,7500.0,0.39,MADS_109,96.4
MADS search,7500.0,0.23,MADS_110,98.0
MADS search,7500.0,0.34,MADS_111,97.0
MADS search,7500.0,1.16,MADS_112,102.1
MADS search,7500.0,1.18,MADS_113,102.9
MADS search,7500.0,0.72,MADS_114,88.7
MADS search,7500.0,0.97,MADS_115,94.7
MADS search,7500.0,0.55,MADS_116,94.5
```

Figure 3 – MADS search points printed in results file

To enhance user engagement and provide transparency during the optimization process, the current iteration number, along with the current point and current mesh size information of the MADS algorithm, is displayed in the terminal. This real-time feedback allows users to monitor the algorithm's progress, understand how many iterations have been completed, and gauge how close the process might be to convergence. Such prompts serve as valuable checkpoints, especially during long-running optimizations, by keeping users informed and involved throughout the computational journey. The prompt is shown in *Figure 4*.

```
Iteration 19:
Current point: 0.345
Current value: 96.969
Mesh size: 0.006

Iteration 20:
Current point: 0.188
Current value: 98.479
Mesh size: 0.012

Iteration 21:
Current point: 0.145
Current value: 98.88
Mesh size: 0.024
```

Figure 4 – Prompt displaying the current iteration performed by the MADS to the user

After completing the MADS optimization process, the final results are presented in a results file, which includes the optimal parameter values and their corresponding objective function value. Following this, the file provides search statistics detailing the total number of function

evaluations performed, caching information such as the number of cache hits and the hit rate as discussed in *Section 3.4.5*, and the evolution of the mesh size from its initial to final value. These details offer a comprehensive overview of the optimization's performance and efficiency, as illustrated in *Figure 5*.

```
# Optimal Solution
Best Duct Length (m): 1.18
Best VE: 102.9

# Search Statistics
total_evaluations: 226
cache_hits: 119
cache_hit_rate: 0.5265486725663717
initial_mesh_size: 0.252
final_mesh_size: 0.024
iterations: 25
lb: -0.08000000000000002
ub: 1.18
range: 1.26
dimension: 1
best_value: 100.98
```

Figure 5 – Optimal point and search statistics printed in the results file

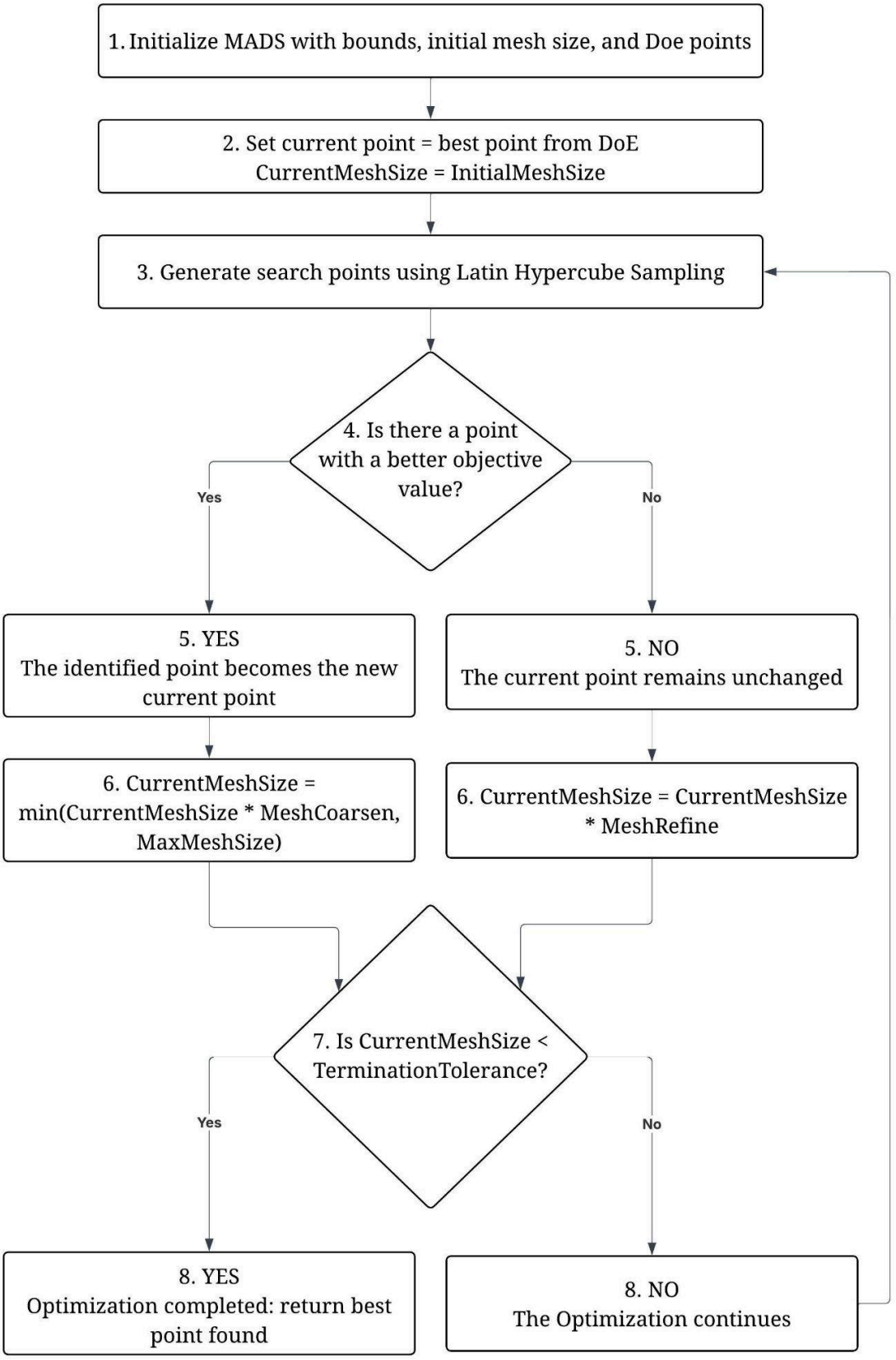

Figure 6 – Summary Flowchart of the optimization steps performed by the MADS algorithm

3.5 Features of *engine_optimizer*

3.5.1 Starting up *engine_optimizer*

Since the optimizer is developed in Python Language, we need to set an environment capable of executing Python scripts. I opted for Visual Studio Code due to its robust features and user-friendly interface, but the user is free to opt for any Python-compatible editor or integrated Development Environment (IDE) that suits their preferences. Once the *engine_optimizer* is launched, it asks the user to select the input files that define the engine's configuration, followed by the selection of the fluid dynamics simulator executable "*Gasdyn.exe*". Subsequently, the optimizer will ask you to input the specific optimization task to be performed, along with the concerned parameters associated with it. These inputs can be provided either manually or through a pre-defined XML file based on the user's preference, where the variables, objectives, and constraints can be chosen and inputted. The actions performed by the optimizer throughout this optimization process are printed in the command terminal, allowing to monitor each step of the process. *Figures 7* and *8* are the prompts that ask the user to select the input files and the executable file, respectively.

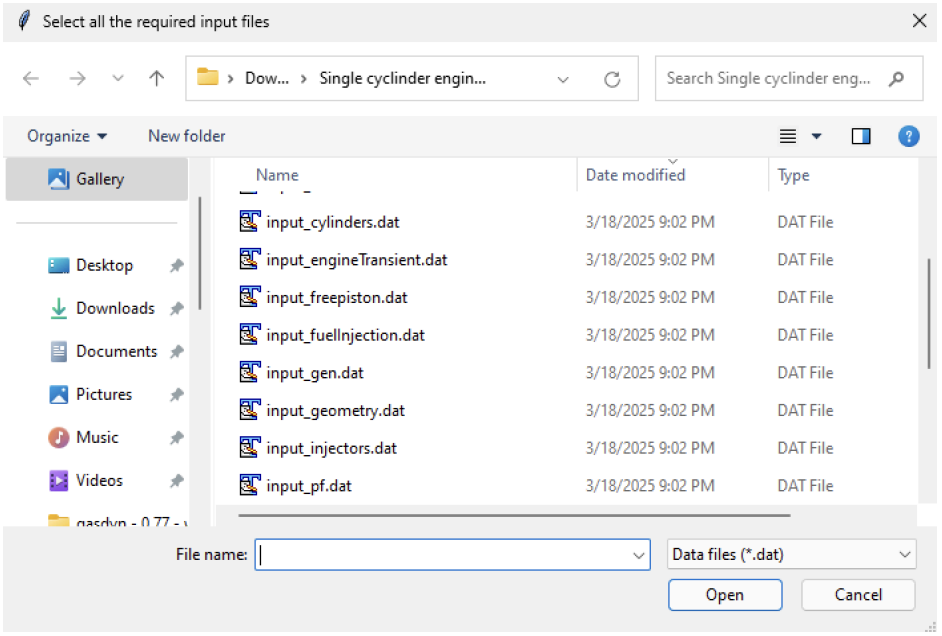

Figure 7 – Input files selection prompt

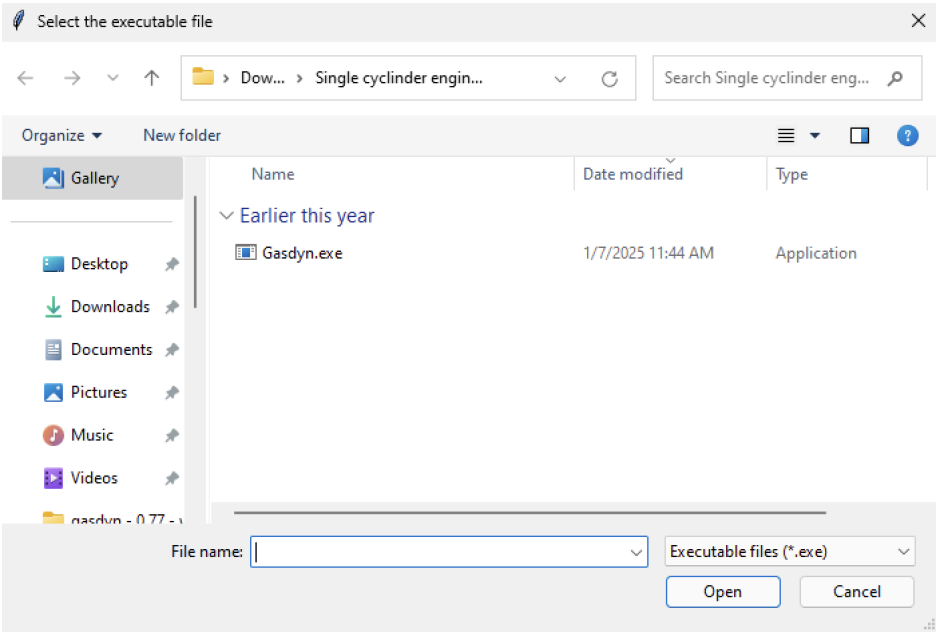

Figure 8 – Executable file selection prompt

Once the input files and the executable have been selected, the program then asks the user to choose whether they want to proceed by inputting the optimization parameters using an XML file or not. The user simply needs to type in their choice and press Enter to proceed. *Figure 9* shows the prompt where this choice is made.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_injectors.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_pf.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_SCmover.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_sensors.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_silencer.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_turbines.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_valves.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_vehicle.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_warmup.dat to acquisition directory.
Copied C:/Users/Naveen/Downloads/Single cyclinder engine inputs/input_wastegate.dat to acquisition directory.
Executable copied successfully.

Do you want to use an XML input file for optimization? (y/n):
```

Figure 9 – XML Inputting choice prompt

If the user enters “y” then the program starts with the XML input file the user provides; otherwise, it switches to the manual input method, where both methods will be explained in the following sections.

3.5.2 Variables:

The optimizer provides a set of key parameters that can be varied and simulated, and these are critical for fine-tuning engine performance. The variables available for optimization include:

- **Duct length:** The length of the intake and exhaust ducts that influences the volumetric efficiency and the engine breathing.
- **Duct diameter:** The cross-sectional area of the ducts affects the airflow resistance and the engine performance.
- **Valve timing (IVO, EXO, IVC, EVC):** The timing of valve events, Intake Valve Opening (IVO), Exhaust Valve Opening (EXO), Intake Valve Closing (IVC), and Exhaust Valve Closing (EVC), which are crucial for optimizing the engine breathing and the combustion efficiency.
- **Air-to-fuel ratio (A/F ratio):** The ratio of air to fuel entering the combustion chamber, which impacts the combustion efficiency and the emissions.

The RPM is not a variable, and this parameter defines the operating range of the engine, which remains unchanged. These RPMs are read by the program from the input files given. The optimizer allows users to select specific RPM values for which they wish to perform simulations and optimizations, depending on the engine's intended application. The user can choose a single RPM, multiple RPMs, or opt to run simulations across the entire available RPM. The available RPMs are prompted to the user, and they can choose among the above-mentioned choices and enter them simply as per the optimizer's instructions prompted. *Figure 10* illustrates the prompt where user input their RPM choices.

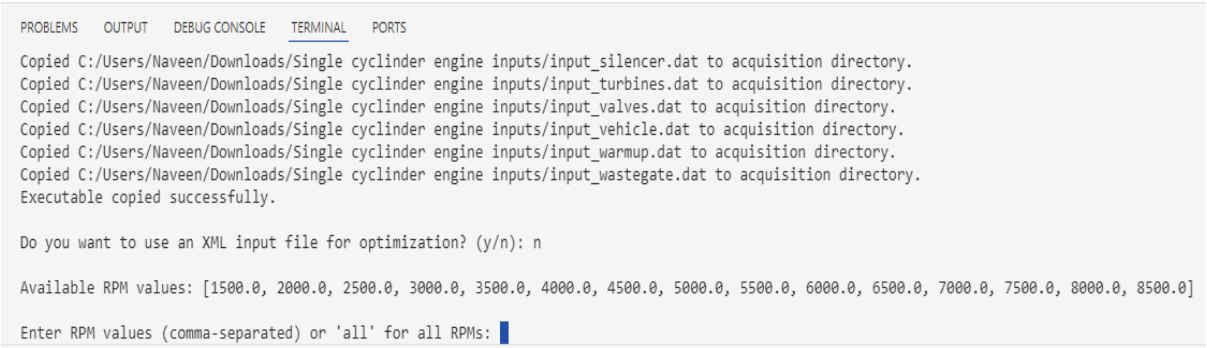

Figure 10 – RPM Inputting prompt

Each variable requires the user to specify a minimum and maximum value, setting the range

within which the parameter will be adjusted during the simulation tests.

3.5.3 Objectives and Constraints

In engine performance optimization, clearly defining objectives and constraints is essential to pave the simulation process toward desired outcomes. Objectives are the performance metrics that the optimization aims to enhance, while constraints are the boundaries within which the optimization must operate, ensuring feasible and practical solutions.

1. **Objectives:**

   The Objectives represent the key performance indicators that the optimization aims to improve. The objectives that the program handles are

   - Volumetric Efficiency
   - Torque
   - Brake Specific Fuel Consumption (BSFC)

   The program prompts the user to select the objective of their preference just by simply inputting as instructed, and *Figure 11* illustrates the objective selection prompt and instruction.

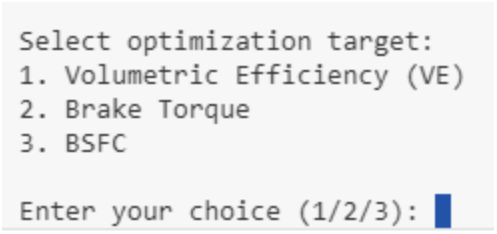

Figure 11 – Objective selection prompt

2. **Constraints:**

   The optimizer has two types of constraints that can be performed, and they are mentioned as optimization modes in the program. They are

   - **Optimize to maximum/minimum value:** Optimizing the variable to the maximum or minimum value of the chosen objective.
   - **Optimize to target value:** Optimizing the variable to the target objective value inputted by the user.

Figure 12 illustrates the optimization mode selection prompt. If option 1 is selected, the simulation begins immediately. However, if option 2 is chosen, an additional prompt will appear, asking the user to enter the target value for the objective.

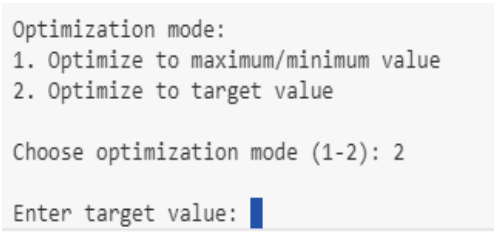

Figure 12 – Constraint selection prompt

3.5.4 Grouping of ducts

One of the key features of the *engine_optimizer* is its ability to group the subelements of the duct for optimization, both in terms of length and diameter. In engine design, both intake and exhaust ducts are made of multiple interconnected subelements. Modifying the length or diameter of a single subelement will not make technical sense, which leads to a non-tangible configuration. Therefore, the grouping of subelements is mandatory to perform technically sound simulations and optimization. The geometry of the intake and exhaust ducts significantly affects the volumetric filling of the cylinder and impacts the engine's performance. This impact is due to the modification in the effects of the pressure waves that are generated by the opening and closing of the valves that travel through the ducts and enhance the flow of air-fuel mixtures into the cylinders. The proper tuning of duct geometries can harness the pressure waves to improve volumetric efficiency and overall engine performance. However, identifying which section of the duct is most sensitive to these effects is challenging, especially in complex engines with numerous subelements.

Trying to optimize each sub-element individually would result in performing an enormous number of simulations, which could cancel out all the advantages of the *engine_optimizer*. For instance, we can consider optimizing only five ducts, each consisting of three subelements, using a three-level factorial design will require $3^{15} = 1.4 * 10^7$ simulations approximately. Many of these configurations would be physically feasible or modify the engine structure too much. Grouping all 15 subelements into a single duct would have allowed only 3 DoE tests to be carried out with easier and faster characterization. To address

these challenges, *engine_optimizer* has the feature of selecting one or more subelements and grouping them, making modifications to the variables as a group, and studying how the duct length and diameter variation affect the performance of the engine. The optimizer reads the available subelements along with their required geometry data and index number from the input files of the engine. Depending on the user's selection of optimizing either duct length or diameter, the program displays a comprehensive list of available subelements. For duct length optimization, each subelement is listed with its original length and subelement index number. For duct diameter optimization, the original left and right diameters and subelement index number of each subelement are provided. This data is provided in a formatted manner in a prompt that allows users to efficiently choose which subelements to group and optimize. An example of this prompt, while performing duct length optimization and duct diameter optimization, is illustrated in *Figures 13* and *14,* respectively.

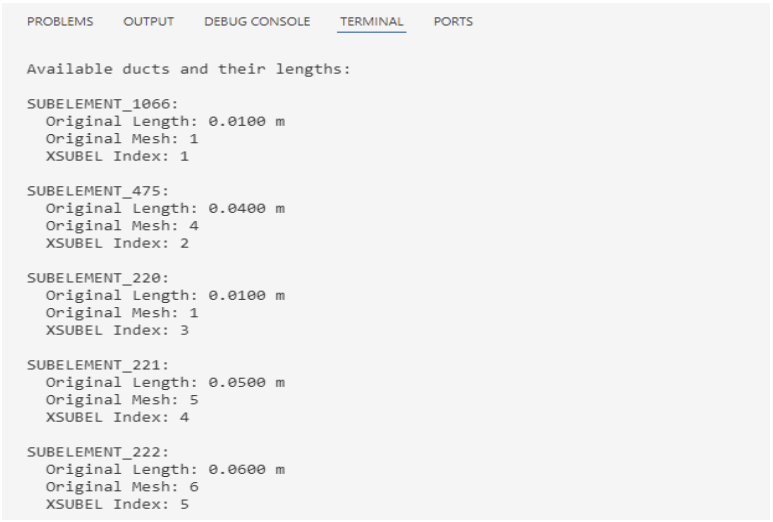

Figure 13 – Available ducts and their length data prompt

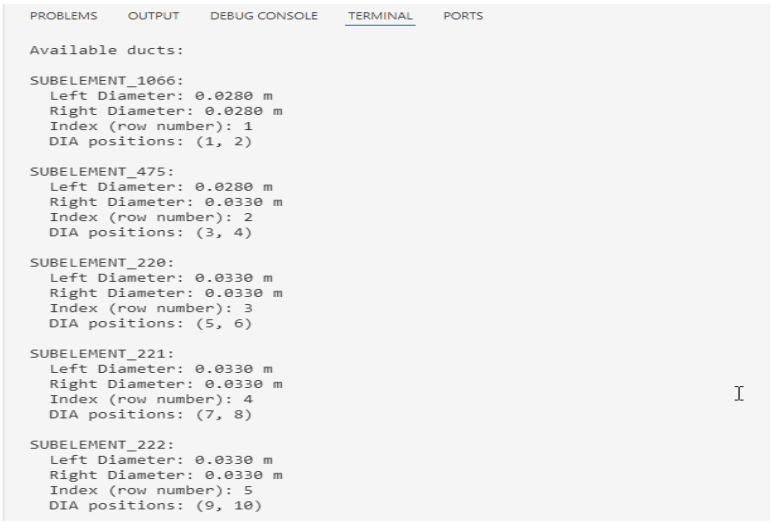

Figure 14 – Available ducts and their diameter data prompt

After showing the available ducts, it will prompt the user to enter the number of groups to simulate, followed by inputting the group's name, and then the subelement index to be grouped, and the index for each subelement is shown in the available ducts prompt. Before entering the indices of the sub-elements to be grouped, the optimizer displays a prompt containing the relevant duct data. These grouping steps are common for both duct length and diameter optimization. *Figure 15* shows the grouping prompt; however, for clarity, the displayed duct data has been simplified and edited to reduce visual clutter and make the illustration more readable.

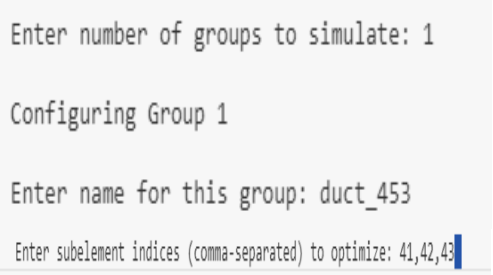

Figure 15 – Duct grouping prompt

Once the subelements to be grouped are selected, the user inputs the minimum and maximum values with the number of Design of Experiments (DoE) to be performed. The variable to be modified, such as duct length, is then varied in the group, and then it is simulated and optimized. For example, if we are optimizing duct length and the selected duct consists of four subelements, and we group it, its original length is 0.3m. Now we need to modify the total length of the duct to 0.5m. It would be incorrect to simply assign 0.5m to each subelement, as it sums to 2m, which is beyond our expected value. If we are implying that each subelement length should be 0.5/4 = 0.125m, it is only valid if all the subelements had identical original lengths, which is not a common case in practical engineering configurations.

To handle this, the optimizer implements proportional scaling based on the original lengths of each subelement. This method maintains the relative length differences while attaining the desired total length. Let me explain how the optimizer performs this scaling with an example. When a target total length is specified for a group of subelements, the optimizer calculates a scaling factor based on the ratio of the target length (L target) to the original total length (L total length).

$$\text{Scaling Factor} = \frac{L_{\text{target}}}{L_{\text{total orig}}} \tag{3.7}$$

Each subelement's new length is then calculated by multiplying its original length by the scaling factor.

$$L_{\text{new}}(i) = L_{\text{orig}}(i) \times \text{Scaling Factor} \tag{3.8}$$

Example case:

The duct consists of 7 subelements, which are as follows

- *SUBELEMENT_1* = 0.09 m
- *SUBELEMENT_2* = 0.041 m
- *SUBELEMENT_3* = 0.093 m
- *SUBELEMENT_4* = 0.098 m
- *SUBELEMENT_5* = 0.1 m
- *SUBELEMENT_6* = 0.025 m
- *SUBELEMENT_7* = 0.02 m

Total Original Length = 0.467 m

Target Length = 1 m

Scaling factor = 1 / 0.467 ≈ 2.141

Applying the scaling factor to each subelement

- *SUBELEMENT_1* = 0.09 * 2.141 ≈ 0.2 m
- *SUBELEMENT_2* = 0.041 * 2.141 ≈ 0.09 m
- *SUBELEMENT_3* = 0.093 * 2.141 ≈ 0.2 m
- *SUBELEMENT_4* = 0.098 * 2.141 ≈ 0.2 m
- *SUBELEMENT_5* = 0.1 * 2.141 ≈ 0.21 m
- *SUBELEMENT_6* = 0.025 * 2.141 ≈ 0.05 m

- *SUBELEMENT_7* = 0.02 * 2.141 ≈ 0.04 m

The sum of these modified lengths is equal to *1m,* which confirms the accuracy of the scaling process. Now we have the solution for the duct length cases, but the grouping is a feature implied in duct diameter optimization also and it has its own hurdles. In some cases, subelements within a duct group may have uniform diameters at both left and right ends, and in this case, the scaling is straightforward, whereas both ends' diameters are modified to the target diameter. However, the challenge arises when subelements have different left and right diameters, and in this case, a single target diameter is not feasible because it would disrupt the original geometric proportions. To tackle these issues, the optimizer facilitates the proportional scaling of duct diameters, which maintains the original ratio between the left and right diameters. The scaling process involves the following steps, and let me explain this with an example case:

- Calculate the ratio between the original right and left diameters for the subelements with asymmetric ends.

$$\text{Ratio} = \frac{\text{Right Diameter}}{\text{Left Diameter}} \tag{3.9}$$

- Now, the left diameter is modified according to the defined target value, and the right diameter is scaled using the ratio.

$$\text{New Right Diameter} = \text{New Left Diameter} \times \text{Ratio} \tag{3.10}$$

Example case:

The duct consists of *SUBELEMENT_1*, whose

- Left diameter = 0.048 m
- Right diameter = 0.06 m

Now we assume we want to test this duct by varying the diameter within a defined range from 0.01 m to 0.1 m using 3 levels of Design of Experiments (DoE). Therefore, the diameter values to be tested are:

- 0.01 m
- 0.055 m
- 0.1 m

Ratio = 0.06 / 0.048 = 1.25

Using the ratio of 1.25, the updated diameters for each test case are calculated as follows.

- Left diameter = 0.01 m
  Right diameter = 0.01 * 1.25 = 0.012 m
- Left diameter = 0.055 m
  Right diameter = 0.055 * 1.25 = 0.068 m
- Left diameter = 0.1 m
  Right diameter = 0.1 * 1.25 = 0.125 m

In the case where the left diameter is larger than the right diameter in the original conditions, then the ratio between the left and right diameters is calculated, and the new left diameter is calculated with the ratio. This ensures that the physical taper of the duct remains consistent while preserving its structural logic.

3.6 Optimization modes

The optimization modes here are based on the different variables to be optimized. The optimization modes available in the *engine_optimizer* are

- Duct length optimization
- Valve timing optimization
- Air-to-Fuel ratio (A/F) optimization
- Duct and Valve optimization
- Duct diameter optimization

Once the *engine_optimizer* is launched, it prompts the user to provide the required input files along with the executable of the fluid dynamic simulator. After that, it asks whether to proceed with the XML input method. If the user enters “no,” the manual input method is initiated. The user is then asked to enter the engine RPM, followed by a prompt to select one of the available optimization modes, and the user can simply enter the choice as instructed in the prompt. *Figure 16* shows the optimization mode selection prompt, and each optimization mode will be explained in the following sections.

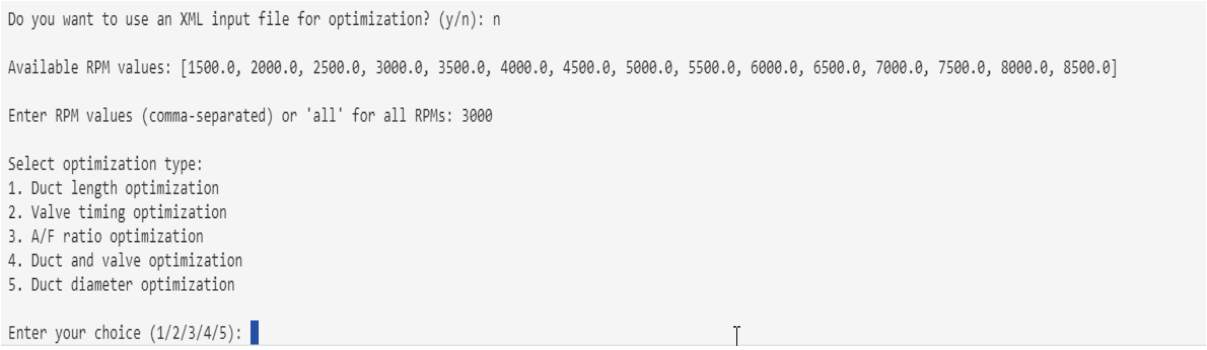

Figure 16 – Optimization mode selection prompt

3.6.1 Duct Length Optimization

As explained in the previous paragraph, the user will be asked the choose the optimization mode. Once "1" is entered, indicating duct length optimization, the available ducts will be displayed along with each subelement's original length, its mesh size, and the index number. *Figure 13* in *Section 3.5.4* will illustrate this prompt. After this, the user is asked to enter the number of groups, group names, and the indices of the subelements to be grouped for simulation and optimization, and *Figure 15* in *Section 3.5.4* illustrates this prompt. If the duct consists of only one subelement, then the user can simply enter that concerned single subelement index number. Once the subelement indices are entered, the program asks the user the enter the minimum and maximum length of the ducts within which the tests will be performed, followed by the number of intervals that define the levels of Design of Experiments (DoE). *Figure 17* will illustrate the entire process from the subelement indices entry to the definition of DoE.

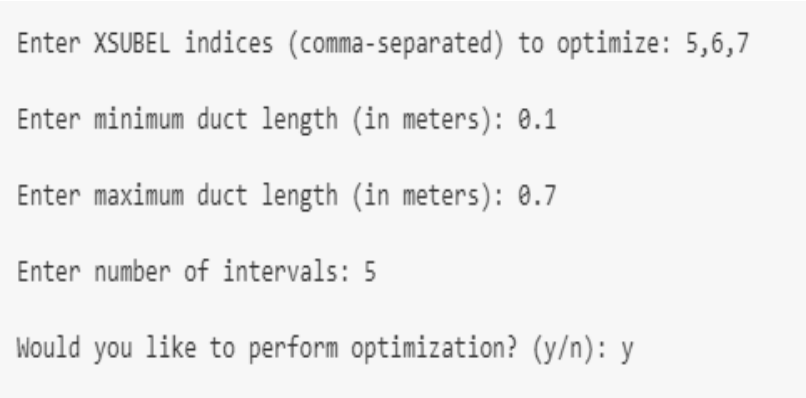

Figure 17 – Duct length optimization variable inputting prompt

After entering the number of intervals, the program prompts the user to confirm whether optimization should be performed. If the user enters "yes," the steps described in *section*

3.5.3 Objectives and Constraints will be followed, allowing the user to define suitable objectives and constraints based on their specific use case. If the user selects "no," only the simulations will be carried out without applying any optimization process.

Let me explain what is happening internally once the inputs are entered by the user. Since the duct's original length is read, the program applies the proportional scaling for the test variables to be performed as explained in *Section 3.5.4 Grouping of ducts*. Once the program determines the test variables for which the simulation is to be run it creates the folders for each Design of Experiments (DoE) level, naming them sequentially as "doe1", "doe2", and so on, and the number of doe folders created will be equal to the number of intervals specified by the user. These folder creations will be happening under the folder name "results" which will be created in the directory where the optimizer is executed and inside it, there will be a subfolder named under the RPM value for which the simulation is running for example if the RPM is 7500, the folder will be named "7500". All the individual DoE folders will be present inside this RPM folder. Inside each DoE folder, the *Input_gen.dat* file is copied, and a specific line containing the keyword "*PUT_VAR*" is added to this file to define the variable being tested. In addition to this, a new file named *Input_optimice*.dat is also generated within each folder. This file contains the corresponding input data passed to *Gasdyn* to run the simulation, formatted as shown in *Figure 18*.

input_optimice.dat

```
&PUT_VAR_1
RPM_INDEX = 4
IREVMIN = 3000.0
/
&PUT_VAR_2
XSUBEL(5) = 0.1826
OPTIMESH(5) = 18
XSUBEL(6) = 0.2739
OPTIMESH(6) = 27
XSUBEL(7) = 0.2435
OPTIMESH(7) = 24
/
```

Figure 18 – *Input_optimice.dat* file formatting for duct length optimization

In this file, each parameter is listed one after the other under the PUT_VAR section. The RPM is added with its index, followed by the test variable duct length in this case, along with the corresponding index and the newly calculated mesh size for each subelement, as shown in *Figure 18*. Each subelement has its own mesh size, which is read from the input files. To

ensure accurate simulation results, the mesh size should increase when the length increases and decrease when the length decreases. The new mesh size is calculated using the following formula:

$$N\_new = (N\_old / L\_old) \times L\_new \tag{3.11}$$

For example, if the original length (L_old) is 0.04 m and the original mesh size (N_old) is 8, and the test variable length (L_new) is 0.45 m, then:

$N\_new = (8 / 0.04) \times 0.45 = 90$

This approach ensures that the mesh density remains consistent relative to the geometry, maintaining the accuracy of the simulation results. But how these test variables are defined to Gasdyn by the *engine_optimizer*, and this process is explained in *Section 3.2. The Interaction Between Gasdyn and engine_optimizer*. Each DoE folder will contain the output files generated by Gasdyn for that specific test case.

3.6.2 Valve timing Optimization

Once the user selects option “2” in the optimization mode prompt, the *engine_optimizer* enters valve timing optimization mode. The program then prompts the engine's original intake and exhaust valve timings, which provide a clear reference to the user. Then the user is prompted to input the desired minimum and maximum values for both intake and exhaust valve timings, defining the range within which the tests will be conducted. Following this, the user specifies the number of intervals, which determines the levels for the Design of Experiments (DoE). This structured approach ensures that the optimization process is tailored to the user's specific requirements. *Figure 19* illustrates the entire procedure described above.

```
Enter your choice (1/2/3/4/5): 2

Original valve timings:
Intake valve(s): 306.0
Exhaust valve(s): 96.0

Enter minimum intake valve timing: 300

Enter maximum intake valve timing: 330

Enter minimum exhaust valve timing: 90

Enter maximum exhaust valve timing: 105

Enter number of intervals for valve timing: 3
```

Figure 19 – Valve timing Optimization prompt

After the user enters the number of intervals, the program asks whether they'd like to proceed with optimization. If the user selects "yes," the optimizer follows the steps explained earlier in *Section 3.5.3*, where they can set the objectives and constraints that best fit their specific case. On the other hand, if the user chooses "no," the tool will simply run the simulations without applying any optimization process.

Once the user has entered all the inputs, the optimizer calculates the test variables needed for the simulations. As described in *Section 3.6.1*, result folders are created, and within each DoE folder, a copy of the *Input_gen.dat* file is placed. A special line with the keyword "*PUT_VAR*" is added to indicate the variable being tested. Along with this, a new file called *Input_optimice.dat* is also created in each folder. This file holds the specific input data passed to *Gasdyn* to run the simulation, structured in the format shown in *Figure 20*.

input_optimice.dat

```
&PUT_VAR_1
RPM_INDEX = 7
IREVMIN = 7500.0
/
&PUT_VAR_2
AVO_VAR(7,1) = 315.0
AVO_VAR(7,2) = 315.0
EVO_VAR(7,1) = 97.5
EVO_VAR(7,2) = 97.5
/
```

Figure 20 – *Input_optimice.dat* formatting for valve timing optimization

In this file, each parameter is listed one after another under the PUT_VAR section. First, the RPM is added with its index, and then the test variable for valve timing is indexed along with the RPM index, as shown in the figure. The output files of each test variable are available in its corresponding DoE folder.

3.6.3 Air-to-Fuel ratio (A/F) Optimization

The Air-to-Fuel ratio (A/F) optimization is initiated when the user enters "3." It asks the user to input the minimum and maximum A/F ratio within which the tests will be performed, followed by inputting the number of intervals, which corresponds to the level of Design of Experiments (DoE). As mentioned in the previous optimization methods, similar steps of optimization preference and selection of objectives and constraints will be prompted and entered by the user. The entire process, from optimization mode selection to constraint inputting, is shown in *Figure 21*.

```
Enter RPM values (comma-separated) or 'all' for all RPMs: 7500

Select optimization type:
1. Duct length optimization
2. Valve timing optimization
3. A/F ratio optimization
4. Duct and valve optimization
5. Duct diameter optimization

Enter your choice (1/2/3/4/5): 3

Enter minimum A/F ratio: 13

Enter maximum A/F ratio: 15

Enter number of intervals: 5

Would you like to perform optimization? (y/n): y

Select optimization target:
1. Volumetric Efficiency (VE)
2. Brake Torque
3. BSFC

Enter your choice (1/2/3): 1

Optimization mode:
1. Optimize to maximum/minimum value
2. Optimize to target value

Choose optimization mode (1-2): 1
```

Figure 21 – Air-to-Fuel ratio (A/F) optimization prompt

Once these inputs are provided by the user, the optimizer begins calculating the test variables for which the simulations will be run. As described in *Section 3.6.1,* the results folder is created, and within it, the DoE folders are generated. Each DoE folder contains a copy of the *Input_gen.dat* file, where a line with the keyword "*PUT_VAR*" is added to specify the variable under test. Additionally, a file named *Input_optimice.dat* is created to facilitate

communication between *Gasdyn* and the *engine_optimizer*, as explained in *Section 3.2*. This file includes the input data required by *Gasdyn* to run the simulation, formatted as shown in *Figure 22*.

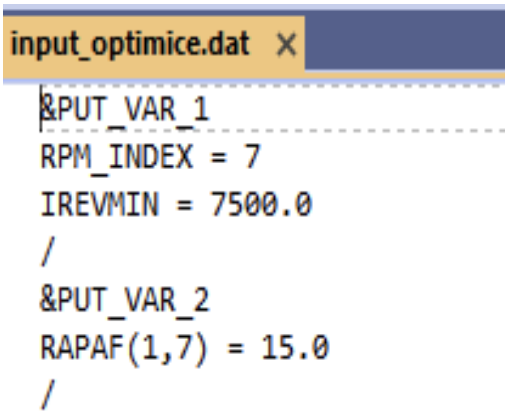

Figure 22 – *Input_optimice.dat* formatting for Air-to-Fuel ratio (A/F) optimization

The RPM indexing and formatting process is similar across all optimization modes. For each case, the test variable for (A/F) ratio optimization is formatted together with the RPM index, clearly indicating the parameter being evaluated, as shown in the figure. The simulation output files are then saved within their respective DoE folders.

3.6.4 Duct length and Valve Optimization

When the user selects option "4" in the optimization mode prompt, the *engine_optimizer* initiates a combined optimization process for both duct length and valve timing. This mode is designed to evaluate how simulations perform when multiple variables are optimized simultaneously. Optimizing multiple parameters at once will make the process faster.

After the mode is initiated the optimizer prompts the user the list of available ducts with its original length and mesh size data as shown in *Figure 8* and it will ask to user to input the subelement indices for which the simulation and optimization to be performed followed by inputting the maximum and minimum lengths of subelement within which the optimization is performed followed by inputting by the number of intervals that define the levels of Design of Experiments (DoE). After this, the optimizer will ask the user to enter the minimum and maximum values of the intake and exhaust duct, followed by the levels of DoE. Once all these values are entered, the program prompts the user to set their optimization preferences, including the objectives and constraints, as explained in *Section 3.5.3*. *Figure 23* will

illustrate the process explained above.

```
SUBELEMENT_31496:
  Original Length: 0.0100 m
  Original Mesh: 2
  XSUBEL Index: 36

SUBELEMENT_31490:
  Original Length: 0.0100 m
  Original Mesh: 2
  XSUBEL Index: 37

SUBELEMENT_31684:
  Original Length: 0.0120 m
  Original Mesh: 2
  XSUBEL Index: 38

Enter XSUBEL Index of the duct to optimize: 5

Enter minimum duct length (in meters): 0.1

Enter maximum duct length (in meters): 1

Enter number of intervals for duct length: 5

Enter minimum intake valve timing: 300

Enter maximum intake valve timing: 330

Enter minimum exhaust valve timing: 90

Enter maximum exhaust valve timing: 105

Enter number of intervals for valve timing: 3

Would you like to perform optimization? (y/n): y

Select optimization target:
1. Volumetric Efficiency (VE)
2. Brake Torque
3. BSFC

Enter your choice (1/2/3): 1

Optimization mode:
1. Optimize to maximum/minimum value
2. Optimize to target value

Choose optimization mode (1-2): 1
```

Figure 23 – Duct and Valve optimization prompt

Once the user has entered all the required inputs, the optimizer begins by calculating the test variables. For duct lengths, it uses proportional scaling as described in *Section 3.5.4*

Grouping of Ducts, while for valve timings, it computes the test values based on the user-defined range. After the test variables are generated, the program creates the necessary results folders following the structure outlined in *Section 3.6.1*. Inside each Design of Experiments (DoE) folder, a copy of the *Input_gen.dat* file is placed, and a special line containing the keyword "*PUT_VAR*" is added to specify the variable being tested, and the optimizer creates a new file named *Input_optimice.dat* in each folder. This file contains the exact input data that will be passed to *Gasdyn* to run the simulation, formatted in the structure shown in *Figure 24*.

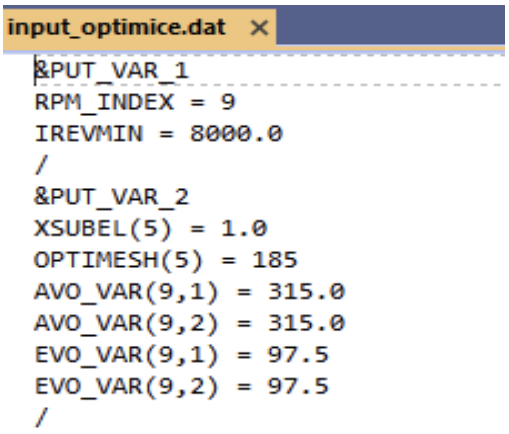

Figure 24 – *Input_optimice.dat* formatting for Duct and Valve optimization

Inside this file, the simulation parameters are listed one after the other under the PUT_VAR section. It begins with the RPM, which is added along with its index. Next, the duct length variable is included, which is written together with its corresponding duct index and the newly calculated mesh size. The calculation of the new mesh size is briefly explained in *Section 3.6.1*. After the duct information, the intake and exhaust valve timings are added, each indexed along with the RPM. These parameters are arranged sequentially, just as shown in the figure. All output files generated during the simulation for each set of test values are stored in their respective DoE folders. This setup is what enables the *engine_optimizer* to handle multi-variable optimization, combining changes to both duct lengths and valve timings in one run. And the best part is, the system is flexible. If we want to add more variables in the future, we just need to follow the same structured approach. This method forms a solid foundation for developing the optimizer's capabilities even further.

3.6.5 Duct Diameter Optimization

When the user selects option “5” in the optimization menu, the *engine_optimizer* enters the duct diameter optimization mode. In this mode, the optimizer displays a list of available ducts, showing their left and right diameters along with their index numbers. It also calculates and presents the positions of these ducts to provide a clear understanding of their layout, as illustrated in *Figure 14* of *Section 3.5.4*. The user is then prompted to define the number of groups for optimization, assign names to each group, and specify the indices of the subelements to be included in each group. If a duct consists of only one subelement, the user can simply enter that specific subelement's index. This grouping process is illustrated in *Figure 15* of *Section 3.5.4*. After defining the groups, the user specifies the minimum and maximum diameters for the ducts within each group, along with the number of intervals that determine the levels for the Design of Experiments (DoE). By following these steps, the optimizer systematically explores various duct diameter configurations, aiming to identify the optimal setup that meets the desired performance criteria. *Figure 25* will illustrate the entire process from the subelement indices entry to the definition of DoE.

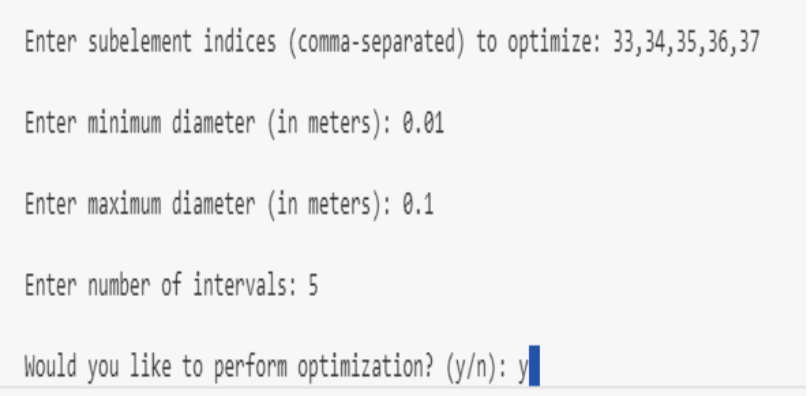

Figure 25 – Duct diameter optimization variable inputting prompt

Once the user enters the number of intervals, the program asks whether to go ahead with optimization. If the user chooses “yes,” the optimizer moves forward by following the process outlined in *Section 3.5.3*, where the user can define objectives and set any constraints based on their preference. On the other hand, if the user opts for “no,” the program will skip the optimization part and just run the simulations.

Since the optimizer reads the original diameter values, it will apply the proportionality scaling for the test variables to be performed, and this proportionality scaling is explained in S*ection 3.5.4 Grouping of ducts*. Once the test variables are calculated for which the simulation is run, the result folders are created as described in *Section 3.6.1*. Within each DoE folder, a copy of *Input_gen.dat* is placed, and a special line "*PUT_VAR*" is added to indicate the variable being tested. Along with this, a new file named *Input_optimice.dat* is created in each DoE folder. This file contains the specific input details that are sent to *Gasdyn* to carry out the simulation, and its format is shown in *Figure 26*.

input_optimice.dat

```
&PUT_VAR_1
RPM_INDEX = 13
IREVMIN = 9000.0
/
&PUT_VAR_2
DIA(65) = 0.055000
DIA(66) = 0.051292
DIA(67) = 0.055000
DIA(68) = 0.055000
DIA(69) = 0.055000
DIA(70) = 0.044611
DIA(71) = 0.055000
DIA(72) = 0.046712
DIA(73) = 0.055000
DIA(74) = 0.048790
/
```

Figure 26 – *Input_optimice.dat* file formatting for duct length optimization

In this file, each parameter is listed one after the other under the PUT_VAR section. The RPM is added with its index, followed by the test variable duct diameter in this case, along with the corresponding index, which is calculated by the optimizer. Since a duct has two sides and two different diameters in certain cases, we need to differentiate between them and pass it to the *Gasdyn* to make it understood. The diameter index is calculated as follows:

$$\text{Left section of subelement number } N = 2 * N - 1 \tag{3.12}$$

$$\text{Right section of subelement number } N = 2 * N \tag{3.13}$$

The index number (N) of the duct will be read by the optimizer from the input files. Let me explain this with an example,

Consider a duct with the index number (N) = 33

Left section of subelement number N = 2 * 33 - 1 = 65

Right section of subelement number N = 2 * 33 = 66

In *Figure 26*, we can see that these computed index numbers are used to mention the test variable calculated using the proportional scaling that ensures the modification interacts with *Gasdyn* correctly. The process of how these test variables are passed from the optimizer to *Gasdyn* is detailed in *Section 3.2*, *The Interaction Between Gasdyn and engine_optimizer*. Once the simulation runs for each test case, the output files generated by *Gasdyn* are saved in their corresponding DoE folders.

3.7 Multiprocessing

One of the key goals of this optimization framework is to make the simulation process time-efficient. Thus, *engine_optimizer* is equipped with the parallel processing feature through Python's *multiprocessing* module. Specifically, it uses the *pool* class that allows multiple simulations to be distributed and executed across different CPU cores simultaneously. This parallel execution strategy helps significantly reduce the total time required for optimization by making full use of available computing resources.

However, running a numerous number of simulations at the same time, there may not be enough CPU memory and access to shared resources. To avoid overloading the system, the optimizer doesn't run every simulation in a single go. Instead, it uses a batching method where it limits the number of simulations running in parallel to 9 at a time. If the total number of simulations is more than 9, it automatically breaks them into batches. This controlled execution helps manage system load and ensures stability during longer and more demanding optimization tasks.

For example, if we need to run 15 simulations, the optimizer splits them into two batches. The first 9 simulations run together in batch 1, and once it is complete, the last 6 simulations will run in batch 2. This step-by-step batching ensures that the system doesn't run out of memory or crash due to excessive parallel load. All the results from each batch are saved and organized in the corresponding results folders, as explained earlier in *Section 3.6.1*. *Figure 27*

shows batch 1 being executed with 9 simulations, and *Figure 28* illustrates batch 2 being executed with the remaining 6 simulations.

```
Processing RPM: 7500.0

Starting batch 1 (9 simulations, 1-9) out of 15 total simulations...
Batch progress: 0/9 simulations completed, 9 currently running
Batch progress: 0/9 simulations completed, 9 currently running
Batch progress: 0/9 simulations completed, 9 currently running
Copied input_axiCoupling.dat to doe1
Copied input_cat.dat to doe1
Copied input_compressors.dat to doe1
```

Figure 27 – Multiprocessing implemented: the first 9 simulations executed in batch 1 can be seen out of 15 simulations to be performed

```
Starting batch 2 (6 simulations, 10-15) out of 15 total simulations...
Batch progress: 0/6 simulations completed, 6 currently running
Batch progress: 0/6 simulations completed, 6 currently running
Batch progress: 0/6 simulations completed, 6 currently running
Batch progress: 0/6 simulations completed, 6 currently running
Batch progress: 0/6 simulations completed, 6 currently running
Copied input_axiCoupling.dat to doe10
```

Figure 28 – Multiprocessing implemented: the remaining 6 simulations executed in batch 2 can be seen out of 15 simulations to be performed

Before initiating the simulations, the optimizer first checks for any duplicates and eliminates them. By doing this, it ensures that the computational resources are not wasted. Then the simulations are launched asynchronously, and the optimizer provides real-time progress updates so users can track the optimization status. Each simulation is equipped with retry mechanisms. If something goes wrong, like a temporary file access issue or a memory hiccup, the optimizer doesn't just fail and stop. Instead, it tries again, with short, increasing delays between attempts. This approach helps handle minor, temporary problems without hindering the entire optimization run. Since multiple simulations can run at the same time, the system also uses thread-safe logging and file-locking mechanisms to avoid race conditions when different processes try to access the same files or data. By balancing smart parallelism with safe resource management, the optimizer delivers solid performance improvements while keeping everything stable and under control.

During the execution of the earlier example, a file access issue occurred, but thanks to the retry mechanism, the optimizer was able to recover and continue. This process is shown in *Figure 29*.

```
Copied executable to doe9
Running simulation for RPM: 7500.0, A/F Ratio: 13.00
Error executing simulation
Return code: 1
Error: 1

Batch progress: 0/9 simulations completed, 9 currently running
Running simulation for RPM: 7500.0, A/F Ratio: 13.29
Batch progress: 0/9 simulations completed, 9 currently running
File access error, retrying... (Attempt 1/3)
Batch progress: 0/9 simulations completed, 9 currently running
Batch progress: 0/9 simulations completed, 9 currently running
Batch progress: 0/9 simulations completed, 9 currently running
Batch progress: 0/9 simulations completed, 9 currently running
Batch progress: 0/9 simulations completed, 9 currently running
Batch progress: 0/9 simulations completed, 9 currently running
```

Figure 29 – Multiprocessing implemented: you can see the file access problem encountered and resolved by the retry mechanism

In the above figure, it is clear that the retry mechanism resolves the file access error, and the retry mechanism is limited to 3. Even after 3 retries, the error persists means the simulations will be terminated and shown to the user as it's not executed.

3.8 Inputting through an *XML* file

As explained in the previous sections, each step in the process is typically guided by prompts from the optimizer, allowing the user to manually input the required parameters based on their needs. To make this process more efficient and user-friendly, the optimizer also supports an XML-based input method. Think of the XML file as a pre-filled instruction sheet, it contains all the necessary actions and values based on the user's intended application. When this file is provided, the optimizer skips the usual step-by-step user prompts and instead directly runs the simulations and optimization as specified in the XML, streamlining the entire workflow.

When the optimizer is launched, the user is first prompted to select the input files, followed by the executable file. Once that's done, the optimizer asks whether to proceed using an XML file. If the user selects "No," the process continues in the standard manual mode. But if the user selects "Yes," the optimizer will prompt them to choose the appropriate XML file. This selection process is illustrated in *Figures 30 & 31*.

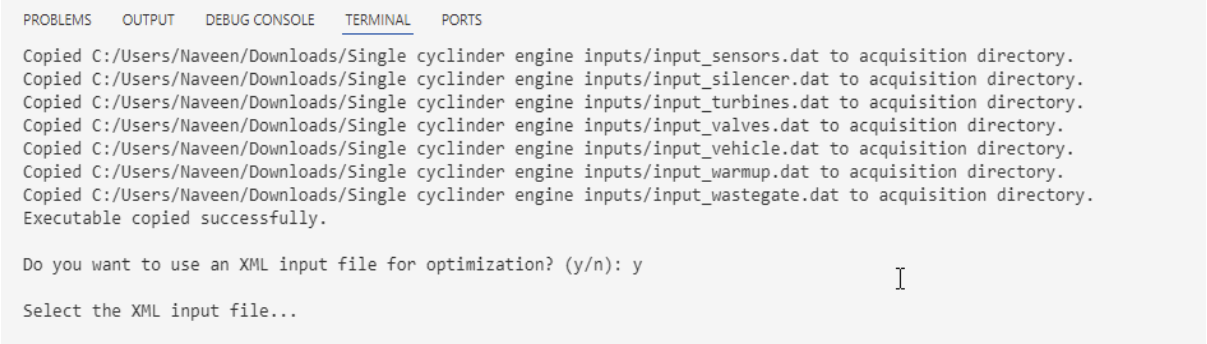

Figure 30 – Prompt asking for XML Input file preference

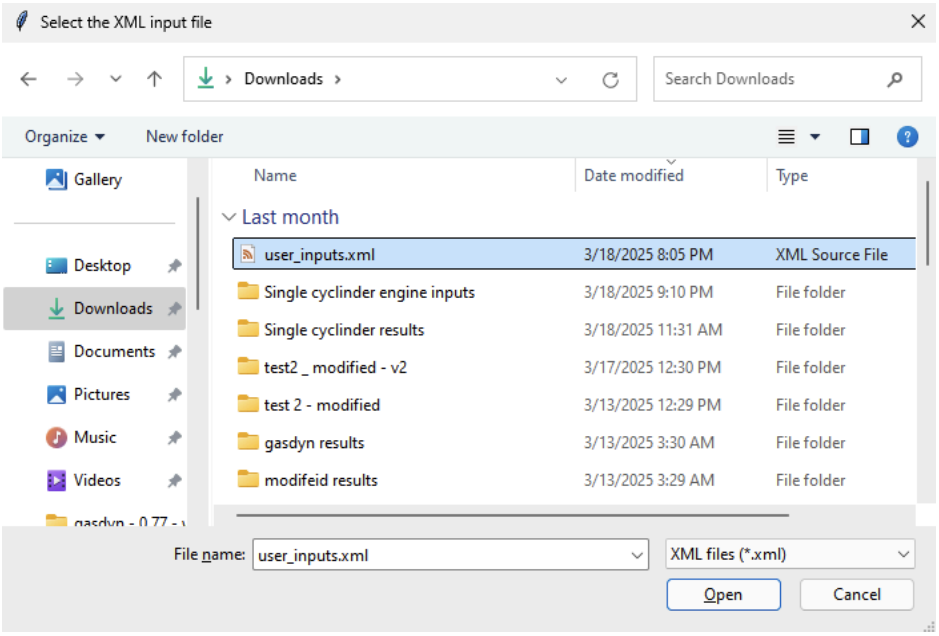

Figure 31 – XML file selection prompt

To make sure everything runs smoothly with the optimizer, the XML file needs to be named exactly *user_inputs.xml*. This name isn't just a suggestion, it's how the optimizer knows to pick it up and use it automatically, without asking for anything extra from the user. Once the file is in place with the right name, the optimizer will handle the rest on its own. Below, we can find a clear breakdown of what goes inside this XML file, so the user can set it up properly and make sure everything is configured just the way the user needs.

```
<?xml version="1.0" encoding="UTF-8" ?>
<OptimizationConfig>
 <!-- Common parameters for all optimization modes -->
 <Parameter Name="RPM" Type="Vector" Dimension="[n]" Unit="-">4000</Parameter>
  <!-- Optimization Type:
      1 = Duct length optimization
      2 = Valve timing optimization
      3 = A/F ratio optimization
      4 = Duct and valve optimization (combined)
      5 = Duct diameter optimization -->
 <Parameter Name="OptimizationType" Type="Integer" Unit="-">1</Parameter>
  <!-- MADS Optimization settings (common to all modes) -->
 <Parameter Name="MADS" Type="Text" Unit="">YES</Parameter>
 <Parameter Name="OptimizationObjective" Type="Integer" Unit="-">1</Parameter>
 <!-- OptimizationObjective:
      1 = Volumetric Efficiency (VE)
      2 = Brake Torque
      3 = BSFC -->

 <Parameter Name="OptimizationMode" Type="Integer" Unit="-">1</Parameter>
 <!-- OptimizationMode:
      1 = Optimize to maximum/minimum value
      2 = Optimize to target value -->

 <Parameter Name="TargetValue" Type="Float" Unit="-">253</Parameter>
 <!-- Only needed if OptimizationMode = 2 -->
  <!-- ============================================================ -->
 <!-- MODE 1: DUCT LENGTH OPTIMIZATION PARAMETERS                  -->
 <!-- ============================================================ -->
  <!-- Group-based parameters for duct length optimization -->
```

```
<Parameter Name="NumberOfDuctGroups" Type="Integer" Unit="-">1</Parameter>

 <!-- Group 1 -->

<Parameter Name="DuctGroup1Name" Type="Text" Unit="-">duct_473 (Exhaust) - VE</Parameter>

<!-- Use EITHER Names OR Indices - Names take precedence if both are specified -->

<Parameter Name="DuctGroup1Names" Type="Vector" Dimension="[n]" Unit="-">SUBELEMENT_1066,
SUBELEMENT_475, SUBELEMENT_220, SUBELEMENT_221, SUBELEMENT_222, SUBELEMENT_223,
SUBELEMENT_224, SUBELEMENT_225</Parameter>

<!-- <Parameter Name="DuctGroup1Indices" Type="Vector" Dimension="[n]" Unit="-">12,13,14</Parameter> -->

<Parameter Name="DuctGroup1MinLength" Type="Float" Unit="m">0.005</Parameter>

<Parameter Name="DuctGroup1MaxLength" Type="Float" Unit="m">1.0</Parameter>

<Parameter Name="DuctGroup1Intervals" Type="Integer" Unit="-">5</Parameter>

 <!-- Group 2 (add more as needed) -->

<Parameter Name="DuctGroup2Name" Type="Text" Unit="-">test2</Parameter>

<Parameter Name="DuctGroup2Names" Type="Vector" Dimension="[n]" Unit="-">SUBELEMENT_31570,
SUBELEMENT_31576, SUBELEMENT_31843, SUBELEMENT_31844, SUBELEMENT_31845,
SUBELEMENT_31846, SUBELEMENT_31847</Parameter>

<Parameter Name="DuctGroup2MinLength" Type="Float" Unit="m">0.02</Parameter>

<Parameter Name="DuctGroup2MaxLength" Type="Float" Unit="m">1</Parameter>

<Parameter Name="DuctGroup2Intervals" Type="Integer" Unit="-">5</Parameter>



 <!-- ============================================================= -->

<!-- MODE 2: VALVE TIMING OPTIMIZATION PARAMETERS                  -->

<!-- ============================================================= -->

 <!-- Valve timing parameters -->

<Parameter Name="IntakeMinTiming" Type="Float" Unit="degrees">300</Parameter>

<Parameter Name="IntakeMaxTiming" Type="Float" Unit="degrees">330</Parameter>

<Parameter Name="ExhaustMinTiming" Type="Float" Unit="degrees">90</Parameter>

<Parameter Name="ExhaustMaxTiming" Type="Float" Unit="degrees">105</Parameter>

<Parameter Name="ValveIntervals" Type="Integer" Unit="-">3</Parameter>

 <!-- ============================================================= -->

<!-- MODE 3: A/F RATIO OPTIMIZATION PARAMETERS                     -->

<!-- ============================================================= -->

 <!-- A/F Ratio parameters -->
```

```
<Parameter Name="AFRatioMin" Type="Float" Unit="-">13.0</Parameter>

<Parameter Name="AFRatioMax" Type="Float" Unit="-">15.0</Parameter>

<Parameter Name="AFRatioIntervals" Type="Integer" Unit="-">5</Parameter>

 <!-- ================================================================ -->

<!-- MODE 4: COMBINED DUCT & VALVE OPTIMIZATION PARAMETERS           -->

<!-- ================================================================ -->

 <!-- Combined duct and valve parameters -->

<!-- Duct parameters for combined mode -->

<Parameter Name="CombinedDuctName" Type="Text" Unit="-">NIL</Parameter>

<Parameter Name="CombinedDuctMinLength" Type="Float" Unit="m">NIL</Parameter>

<Parameter Name="CombinedDuctMaxLength" Type="Float" Unit="m">NIL</Parameter>

<Parameter Name="CombinedDuctIntervals" Type="Integer" Unit="-">NIL</Parameter>

 <!-- Valve parameters for combined mode (reuses valve timing ranges from Mode 2) -->

<Parameter Name="CombinedValveIntervals" Type="Integer" Unit="-">3</Parameter>

 <!-- ================================================================ -->

<!-- MODE 5: DUCT DIAMETER OPTIMIZATION PARAMETERS                   -->

<!-- ================================================================ -->

 <!-- Group-based parameters for diameter optimization -->

<Parameter Name="NumberOfDiameterGroups" Type="Integer" Unit="-">2</Parameter>

 <!-- Group 1 -->

<Parameter Name="DiameterGroup1Name" Type="Text" Unit="-">duct_38994</Parameter>

<!-- Use EITHER Names OR Indices - Names take precedence if both are specified -->

 <Parameter Name="DiameterGroup1Names" Type="Vector" Dimension="[n]" Unit="-">SUBELEMENT_31570,
SUBELEMENT_31576, SUBELEMENT_31843, SUBELEMENT_31844, SUBELEMENT_31845,
SUBELEMENT_31846, SUBELEMENT_31847</Parameter>

 <Parameter Name="DiameterGroup1MinDiameter" Type="Float" Unit="m">0.02</Parameter>

 <Parameter Name="DiameterGroup1MaxDiameter" Type="Float" Unit="m">0.1</Parameter>

 <Parameter Name="DiameterGroup1Intervals" Type="Integer" Unit="-">5</Parameter>

 <!-- Group 2 (add more as needed) -->

 <Parameter Name="DiameterGroup2Name" Type="Text" Unit="-">test2</Parameter>

 <Parameter Name="DiameterGroup2Names" Type="Vector" Dimension="[n]"
Unit="-">5,6,7,8,9,10,11</Parameter>
```

```
<Parameter Name="DiameterGroup2MinDiameter" Type="Float" Unit="m">0.02</Parameter>
<Parameter Name="DiameterGroup2MaxDiameter" Type="Float" Unit="m">0.1</Parameter>
<Parameter Name="DiameterGroup2Intervals" Type="Integer" Unit="-">5</Parameter>
</OptimizationConfig>
```

The green-colored lines in the XML file are comments, and they serve as helpful notes to explain what each parameter does. For example, near the section where you specify the optimization type, there's a comment that tells you what each number (1 to 5) stands for, each one corresponding to a different optimization mode.

The order of the XML file contents is the same as the flow followed in the manual inputting mode. The content provided above is an example of a specific simulation setup, and in this, you can see every particular is filled, and initially, in this sheet, where the optimization type is asked, it is entered as "1" so the optimizer will consider only the parameters entered in the section "*DUCT LENGTH OPTIMIZATION PARAMETERS*" and likewise not only for this for particulars, once the initial crucial particulars are filled like the Optimization mode, target value if applies, single or grouped ducts ., etc and then the section regarding to it only will be considered.

The optimizer is smart enough to ignore any other parameters that don't apply to the chosen optimization type. So even if you accidentally fill out sections that aren't relevant to your current setup, it won't cause any issues. At present, the XML sheet is set up to handle two groups of ducts for both duct length and duct diameter optimization. However, if the user needs to include more groups, they can simply follow the same structure and add additional entries accordingly. This makes the XML file highly flexible and easily adaptable to suit different simulation configurations and testing needs. Once the things are filled, as per the simulation needs to run and given to the optimizer, it will parse it and prompt the entries made by the user via an XML file as shown in *Figure 32*.

```
Do you want to use an XML input file for optimization? (y/n): Y

Select the XML input file...

Successfully parsed XML input with the following values:
  selected_rpms: [4000.0]
  opt_choice: 1
  perform_optimization: True
  optimization_target: 1
  optimization_mode: 1
  target_value: None
  group_mode: True

Number of groups: 1

Group 1: duct_473 (Exhaust) - VE
  group_name: duct_473 (Exhaust) - VE
  selected_names: ['SUBELEMENT_1066', 'SUBELEMENT_475', 'SUBELEMENT_220', 'SUBELEMENT_221', 'SUBELEMENT_222', 'SUBELEMENT_223', 'SUBELEMENT_224', 'SUBELEMENT_225']
  min_length: 0.005
  max_length: 1.0
  intervals: 5
```

Figure 32 – Prompt showing the details it parsed from the XML input file given by the user

Once the XML file is provided, the optimizer reads and maps all the details to the corresponding simulation components and optimization parameters. After this mapping is complete, the optimizer skips the manual input steps and directly begins running the simulation, followed by the optimization process, and *Figure 33* illustrates this process.

```
Mapping for group duct_473 (Exhaust) - VE:
  Name: SUBELEMENT_1066, Index: 1, Original Length: 0.0100 m
  Name: SUBELEMENT_475, Index: 2, Original Length: 0.0400 m
  Name: SUBELEMENT_220, Index: 3, Original Length: 0.0100 m
  Name: SUBELEMENT_221, Index: 4, Original Length: 0.0500 m
  Name: SUBELEMENT_222, Index: 5, Original Length: 0.0600 m
  Name: SUBELEMENT_223, Index: 6, Original Length: 0.0900 m
  Name: SUBELEMENT_224, Index: 7, Original Length: 0.0800 m
  Name: SUBELEMENT_225, Index: 8, Original Length: 0.1200 m

Using XML input file with the following settings:
RPM values: [4000.0]
Optimization type: Duct length optimization
Perform optimization: Yes
Optimization target: Volumetric Efficiency
Optimization mode: Maximum/Minimum

Processing RPM: 4000.0

Starting batch 1 (5 simulations, 1-5) out of 5 total simulations...
```

Figure 33 – Prompt showing mapped parameters related to the inputs given in the user-provided XML input file

The folder structure and the way results are saved work just like what was explained in *Section 3.6.1*. The only thing that changes here is how the input is given to the optimizer; the rest of the function remains the same.

3.9 Results & Graphical outputs

As explained in *Section 3.6.1*, the results are organized into folders, with each simulation getting its own DoE folder. Inside each of these folders, there's a file called *regimes.csv*, which contains the results specific to that simulation. However, checking each folder individually to understand the overall picture can be quite time-consuming. To make this easier, the optimizer automatically reads the results from all the DoE folders and combines them into one file called final_results.csv. This file gives you a clear, overall view of all the test variables and their corresponding results. If MADS optimization is used, the search points, the optimal value, and the search statistics are also added to the same file. This way, you can easily see both the simulation and optimization results in one place. *Figure 34* shows the combined DoE results, while *Figures 3* and *4* display the MADS search points and statistics.

final_results.csv

```
Type,RPM,Duct_Length,DOE,Vol.eff_[%]_max,Br.torque_[Nm]_max,BSFC_[g/kWh]_min,BMEP_[bar]_max
DoE test,7500.000,0.100,1,99.300,28.000,275.200,14.130
DoE test,7500.000,0.325,2,97.200,27.500,260.500,13.840
DoE test,7500.000,0.550,3,94.600,26.600,277.600,13.410
DoE test,7500.000,0.775,4,86.900,24.400,275.200,12.290
DoE test,7500.000,1.000,5,95.800,27.000,259.700,13.620
```

Figure 34 - Cumulative DoE results saved in *final_results.csv*

The *engine_optimizer* automatically generates plots for each type of optimization it performs. For all the available optimization modes, it creates three main plots after the simulations are completed:

1. Volumetric Efficiency vs RPM
2. Torque vs RPM
3. Brake Specific Fuel Consumption (BSFC) vs RPM

These plots are based only on the tests carried out at the DoE (Design of Experiments) points. Once generated, the plots are shown to the user and also saved in the respective results folders. *Figures 35, 36,* and *37* display these plots.

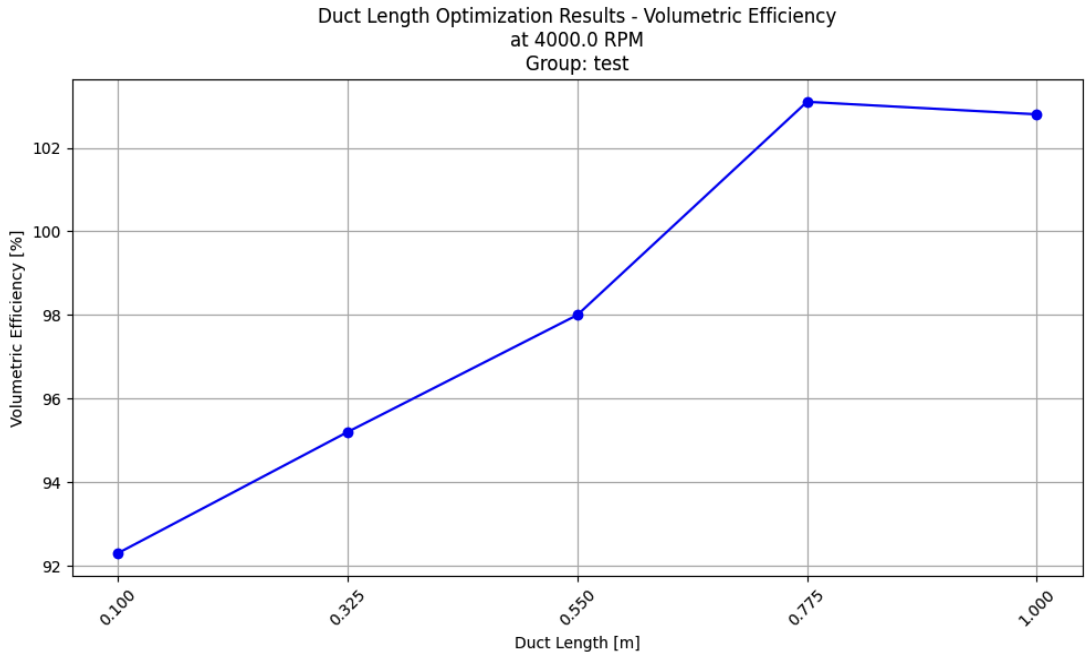

Figure 35 - Plot showing the Volumetric Efficiency vs RPM for the DoE points

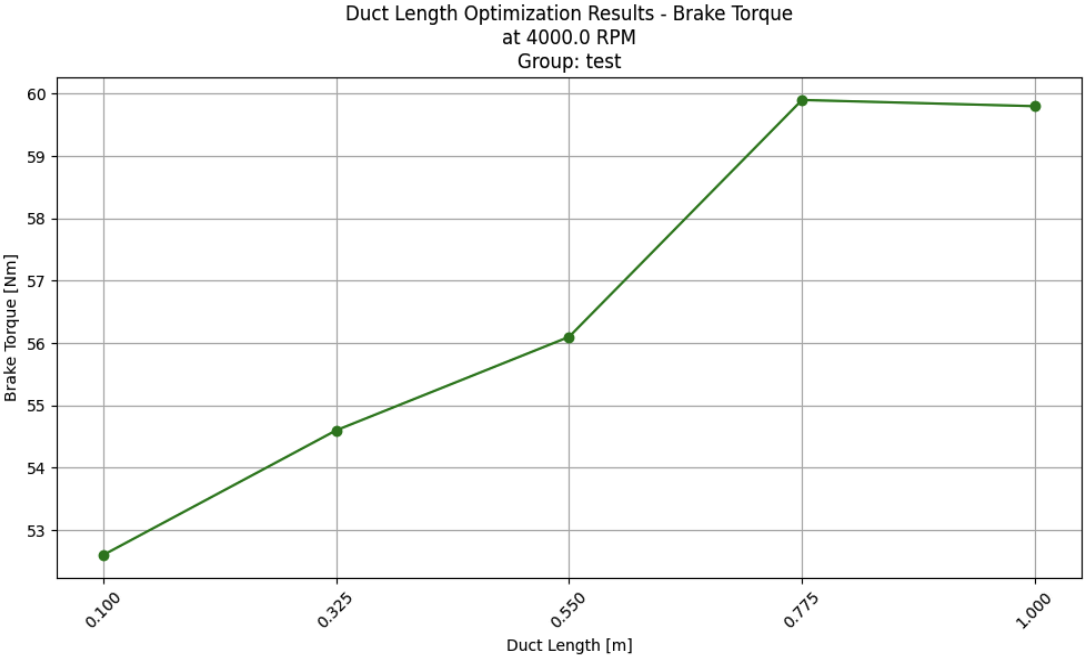

Figure 36 - Plot showing the Torque vs RPM for the DoE points

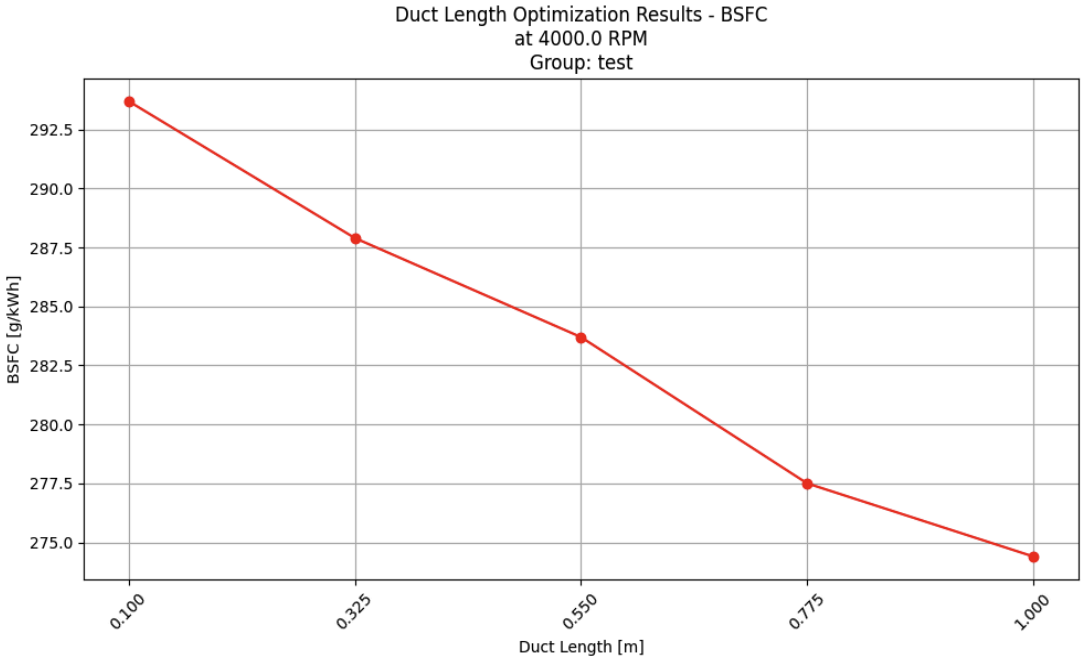

Figure 37 - Plot showing the Brake Specific Fuel Consumption (BSFC) vs RPM for the DoE points

The plots mentioned earlier are 2D because most optimization modes involve only two variables: the input parameter and RPM. However, in the case of valve timing optimization, there are three key parameters involved: Intake Valve Opening (IVO) and Exhaust Valve Opening (EVO), along with the performance metric being evaluated (like torque, VE, or BSFC). Since this optimization involves a combination of two input parameters (IVO and EVO), a simple 2D plot isn't enough to capture the full relationship. To give better insights, the optimizer generates both 3D surface plots and contour plots for each performance metric. These plots help visualize how the output changes across the range of IVO and EVO values. *Figures 38* and *39* show examples of the 3D and contour plots generated during valve timing optimization.

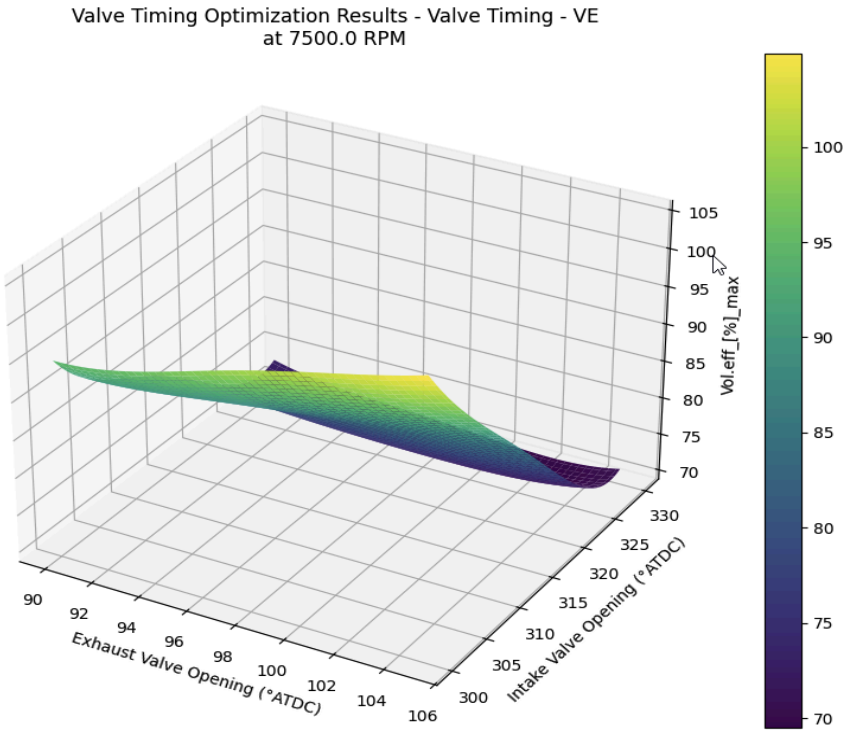

Figure 38 - 3D Plot showing the Volumetric Efficiency vs RPM for the DoE points for Valve Timing Optimization

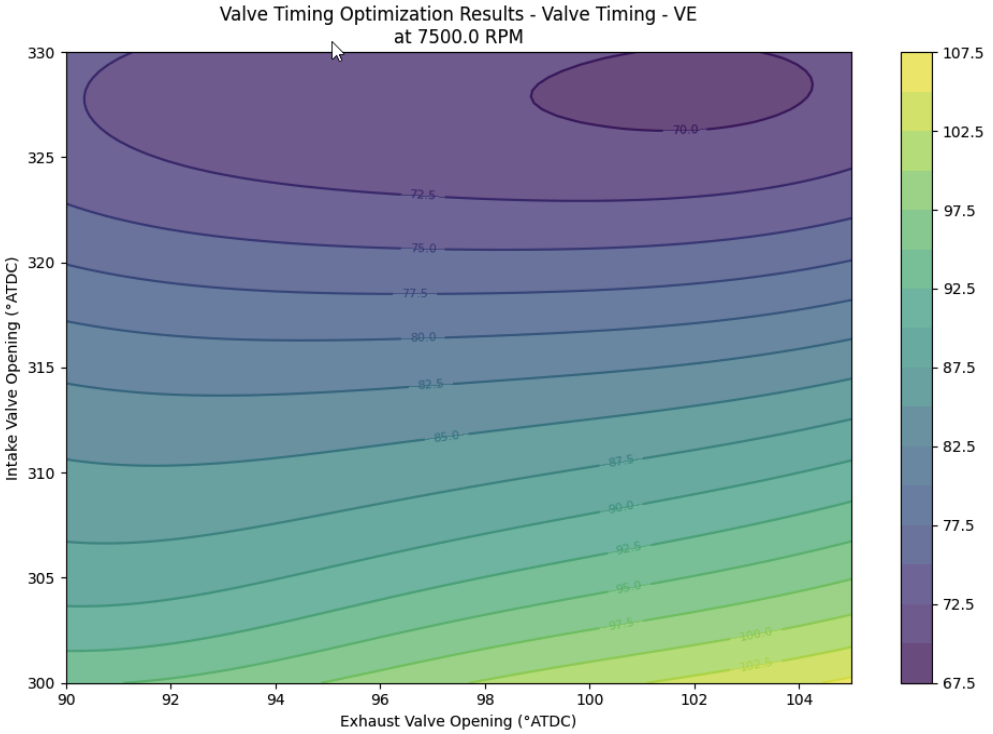

Figure 39 - Contour Plot showing the Volumetric Efficiency vs RPM for the DoE points for Valve Timing Optimization

Next, the optimizer generates a dedicated optimization plot, which is also saved in the results folder. This plot combines Volumetric Efficiency, Torque, and BSFC vs RPM into a single figure using subplots. To make it easier to understand the optimization process, the MADS search point (the value tried by the optimizer) is marked in red, while the optimal point found is highlighted in green. *Figure 40* shows an example where the optimization was performed with the goal of maximizing volumetric efficiency, and the plot reflects this. Depending on the selected objective (Torque, BSFC, or VE), the plot highlights and details the relevant performance trends accordingly.

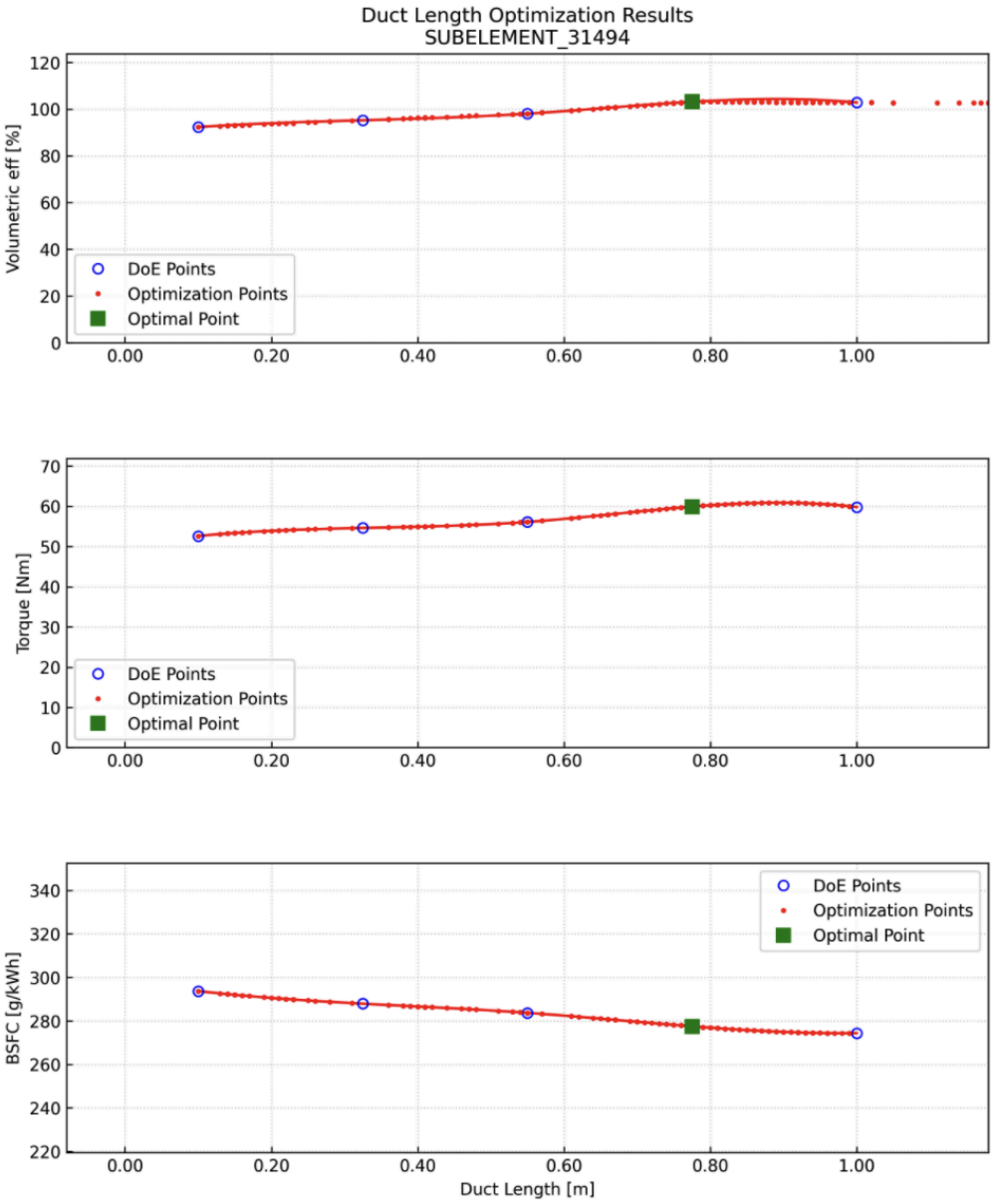

Figure 40 - Optimization plot that shows the DoE points, the MADS point, and the optimal point

Figure 41 shows a contour plot from the valve timing optimization where the objective was to maximize volumetric efficiency. As seen earlier, since the valve timing optimization handles two input variables, these plots help visualize how performance varies with changes in IVO and EVO, with DoE points, MADS points, and the optimal point marked.

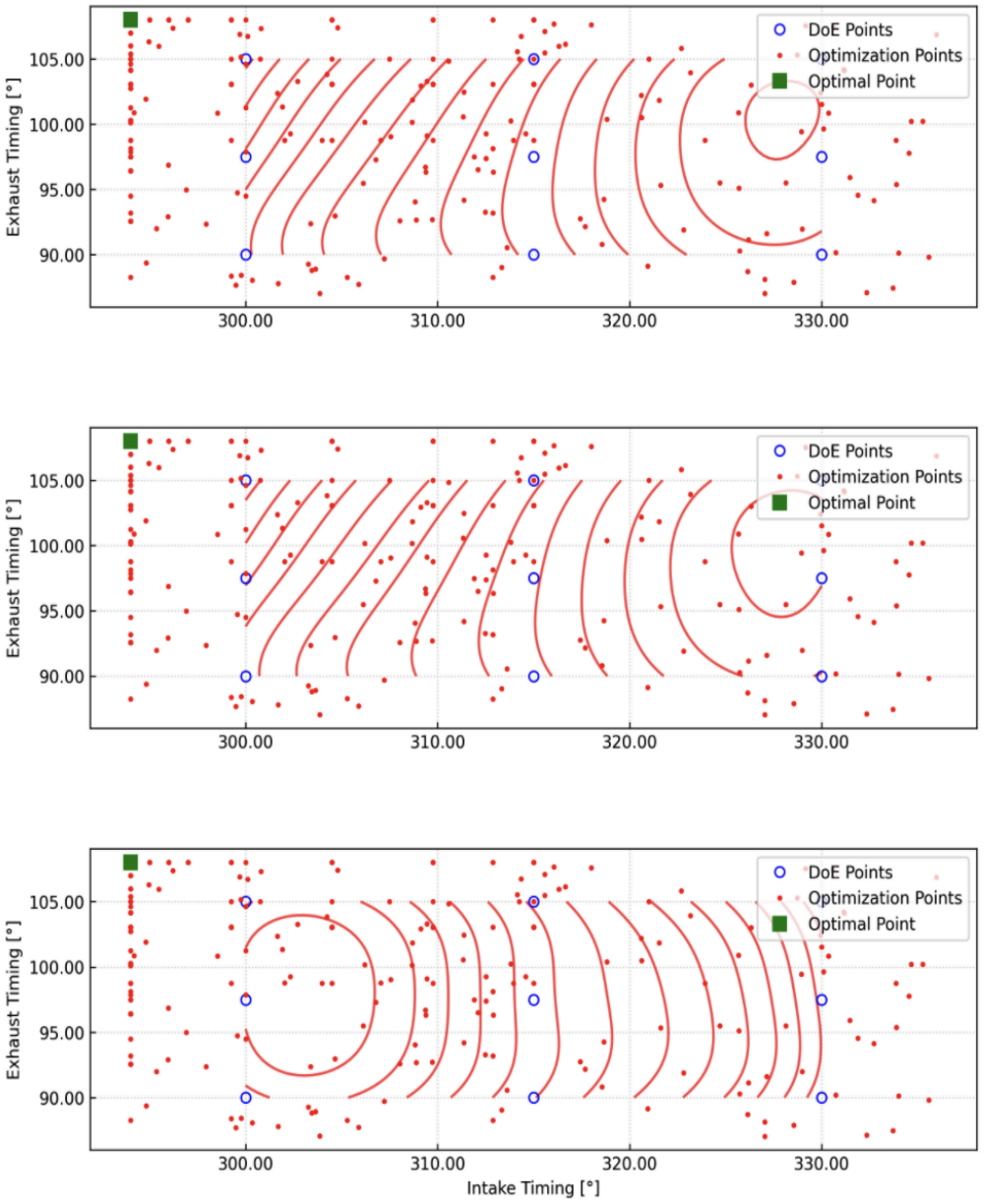

Figure 41 - Valve timing Optimization plot that shows the DoE points, the MADS point, and the optimal point

Chapter 4

Optimizing Inlet duct length and Variable valve timing of a Single-Cylinder Motorcycle Engine

In the second phase of this thesis, we focus on enhancing the performance of a single-cylinder motorcycle engine by optimizing two critical parameters: the inlet duct length and variable valve timing (VVT). This case study serves to validate the comprehensive functionality of the developed optimization tool.

The process begins with establishing the engine's baseline configuration using *Gasdyn* and validating it with the theoretical calculations. *Gasdyn* is a one-dimensional engine simulation software developed by the Internal Combustion Engine Group at Politecnico di Milano. It allows for detailed modeling of unsteady reacting flows within the intake and exhaust systems of internal combustion engines, making it a suitable tool for this analysis.

The optimal inlet duct length is initially determined through theoretical calculations, considering factors such as acoustic tuning and wave dynamics to maximize volumetric efficiency. These theoretical results are then cross-validated with simulations conducted in *Gasdy*n, ensuring consistency and accuracy in the optimization approach.

Subsequently, the study explores the impact of variable valve timing on engine performance. By adjusting the timing of the intake and exhaust valves, the engine's breathing characteristics can be optimized for different operating conditions. The optimization tool is employed to identify the most effective valve timing settings, and their influence on performance metrics such as volumetric efficiency, torque, power output, and fuel efficiency is thoroughly examined.

4.1 Dynamic Effects in the Engine

Before getting into the optimizing part, it's important to understand the dynamic phenomena that significantly affect the air and exhaust flow inside the engine and, ultimately, the engine's performance. These are broadly categorized into two types: inertial effects and wave effects. Both of these play a critical role in how effectively the cylinder fills with fresh air and evacuates exhaust gases during each engine cycle.

4.1.1 Inertial Effects

When the engine piston moves during the intake or exhaust stroke, it pushes or pulls on the gases inside the ducts. Because the gas has mass, it resists changes in motion similar to how a moving object resists stopping immediately. This behavior can be exploited: if the frequency of the gas movement is matched correctly to the engine speed, it can help improve the filling of the cylinder. The gas column in the intake or exhaust duct acts like a Helmholtz resonator, a simple oscillating system with a natural frequency given by:

$$f_s = \frac{a}{2}\sqrt{\frac{S}{L \cdot V_m}} \tag{4.1}$$

where:

- f_s is the system's natural frequency,
- a is the speed of sound,
- S is the duct cross-sectional area,
- L is the duct length,
- V_m is the average cylinder volume during the event.

For best performance, the natural frequency of this gas column should align with the engine's rotational frequency (f_m). Ideally, the ratio f_s/f_m should be an even number (2, 4, 6,...), with a ratio of 2 often providing the best results. In practice, this means that tuning the duct length and diameter can help match these conditions to optimize cylinder scavenging. Shortening the ducts, widening them, or using multiple cylinders are common strategies to adjust this resonance and shift the engine's powerband higher.

4.1.2 Wave Effects

Besides inertial effects, pressure waves traveling through the intake and exhaust ducts also strongly impact engine breathing. When a valve opens, it generates a pressure wave that moves through the duct at the speed of sound. When this wave hits a change in geometry (like the end of a pipe or a junction), it reflects back, sometimes as a positive wave (compression) and sometimes as a negative wave (depression).

These reflections can either help or hurt the intake or exhaust process:

- A **negative pressure wave** reaching the cylinder during exhaust helps suck out burnt gases.
- A **positive pressure wave** reaching the cylinder during intake helps push more fresh air into the cylinder.

Timing these waves correctly can make a big difference. Ideally, the returning wave should arrive when the valve is open and assist the piston's movement.

The time it takes a pressure wave to travel along the duct is related to the crank angle:

$$\Delta\Theta = 360^\circ \times n \times \frac{L}{a} \tag{4.2}$$

where n is the engine speed in revolutions per second.

The optimal condition for maximizing the effect during valve open timing is when:

$$2\Delta\Theta = 90^\circ \tag{4.3}$$

This relationship shows that shorter ducts are needed for higher RPM operation to correctly time the pressure waves. Wave effects even matter when the valves are closed: If a positive pressure wave arrives just as an intake valve opens, it boosts cylinder filling. If a negative wave arrives, it can harm performance. Thus, valve timing and duct length must be coordinated carefully to harness these wave behaviors.

Figures 42 & 43 illustrate the impact of wave effects on cylinder filling and emptying during

the valve overlap phase. In *Figure 47*, the pressure profiles at the intake and exhaust are shown, demonstrating how properly tuned wave effects can enhance cylinder filling and emptying. Conversely, *Figure 48* presents a scenario where wave effects are not optimally tuned, leading to less effective cylinder filling and emptying. These figures underscore the significance of wave dynamics in engine performance. When the timing of pressure waves aligns with valve events, they can facilitate the expulsion of exhaust gases and the intake of the fresh charge, thereby improving volumetric efficiency. However, improper tuning can result in adverse effects, such as backflow of exhaust gases into the intake, which can hinder engine performance.

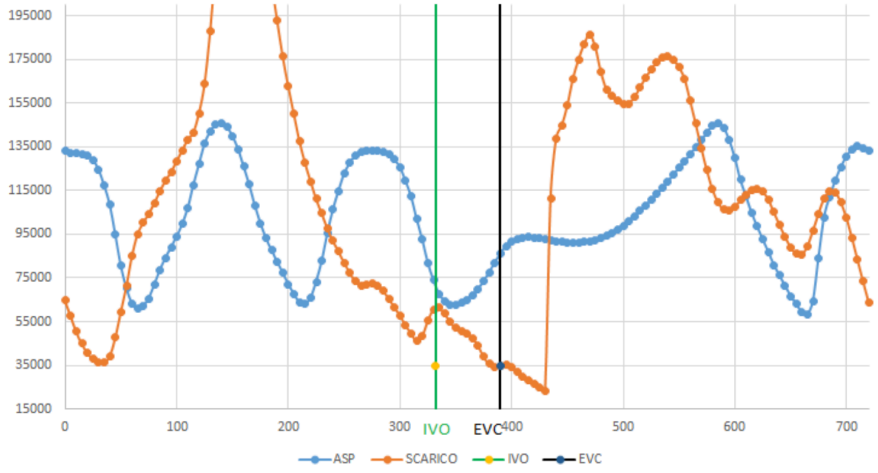

Figure 42 – Pressure trends at the intake (blue) and exhaust (red): The graph shows how wave effects assist the filling and emptying of the cylinder during the valve overlap phase

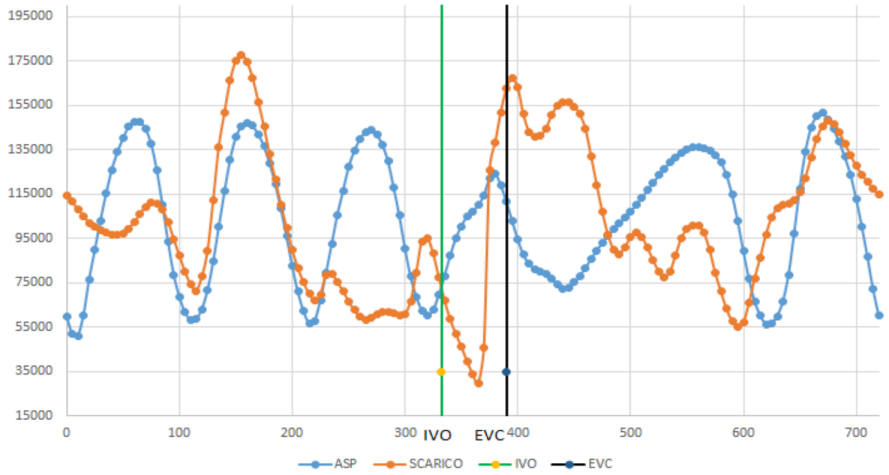

Figure 43 - Pressure trends at the intake (blue) and exhaust (red): The graph shows how wave effects do not assist the filling and emptying of the cylinder during the valve overlap phase.

4.2 Engine data

It's a single-cylinder motorcycle engine test case derived from the real engine, with the specifications of the engine tabulated in *Table 2,* and *Figure 44* illustrates the schematic layout of the engine model constructed within the *Gasdyn* environment, depicting the configuration used for simulation and analysis.

Parameter	Value
Bore [B]	0.098 m
Stroke [S]	0.0764 m
Engine Displacement [V_d]	576.28 cm³
Compression ratio [r]	11
Inlet Valve Diameter [d_{iv}]	0.031 m
Exhaust Valve Diameter [d_{ev}]	0.028 m
Air-to-Fuel Ratio	13
Operating Range [n]	1500 - 8500 rpm
Inlet Valve Opening [IVO]	300°
Exhaust Valve Opening [EVO]	75°

Table 2 – Specifications of Single-Cylinder Motorcycle Engine

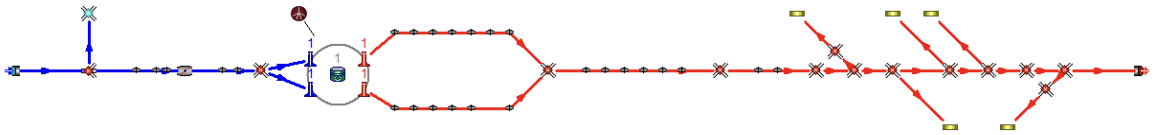

Figure 44 - Schematic representation of the single-cylinder motorcycle engine, with intake ducts highlighted in blue and exhaust ducts in red

The valve timing representation of the engine can be viewed in *Gasdyn* by navigating to the 'Valves' section and accessing the 'Data'. Within this data window, there is an option called 'Show Valve Timing'. By selecting this, you can view a graphical representation of the valve timing for both the cam and the valve. The valve timing diagram of the cam is illustrated in *Figure 45*.

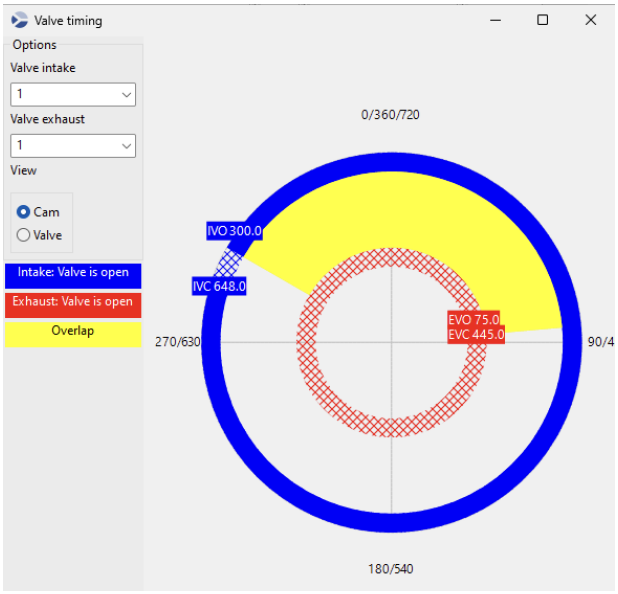

Figure 45 - Valve timing representation of the single-cylinder motorcycle engine

Based on the specified engine parameters, a simulation was performed using *Gasdyn* at full load conditions, and the key performance parameters such as maximum power and its corresponding RPM, maximum torque and its corresponding RPM, and the maximum Brake Mean Effective Pressure (BMEP) at the opted operating conditions were obtained. The performance metrics obtained at each operating point are presented in *Table 3*. *Table 4* provides a summary of the maximum performance parameters, while *Figure 46* illustrates the simulation output plotted using the same input specifications.

RPM	Volumetric efficiency (%)	Break Torque (Nm)	Brake Power (kW)	BSFC (g/kWh)
1500	85.8	46.1	7.2	334.1
2000	84.7	46.6	9.8	315.9
2500	84.7	47.3	12.4	305.2
3000	86.0	49.6	15.6	295.6
3500	93.9	54.0	19.8	296.9
4000	95.3	54.6	22.9	287.8
4500	91.8	52.8	24.9	281.1
5000	92.0	52.0	27.2	282.0

5500	87.3	49.4	28.5	278.1
6000	89.7	50.3	31.6	277.4
6500	88.6	48.7	33.2	280.8
7000	91.5	49.4	36.2	284.7
7500	90.4	47.6	37.4	291.3
8000	86.5	44.1	36.9	300.3
8500	81.0	39.1	34.8	316.5

Table 3 – Engine's Performance Parameters Obtained from the *Gasdyn* Simulation Results at Full Load Conditions at Opted Operating Conditions points of the engine

Parameter	Value
Max Torque $[T_{max}]$	*54.6 Nm @ 4000 rpm*
Max Power $[P_{max}]$	*37.4 kW @ 7500 rpm*
Max Volumetric Efficiency $[\lambda_v]$	*95.3 % @ 4000 rpm*
Max $BMEP$	*11.9 bar @ 4000 rpm*

Table 4 – Engine's Maximum Performance Parameters

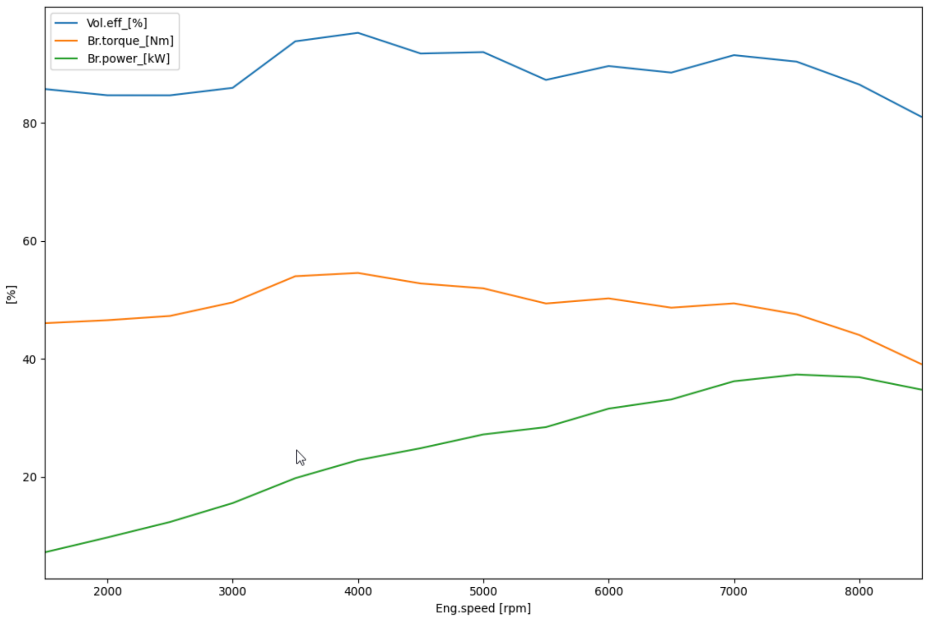

Figure 46 - Engine Performance Curve from *Gasdyn* Simulation

4.3 Engine Theoretical Calculations

Before starting the optimization process, it was important to verify the engine's baseline performance through theoretical calculations, also to cross-check and ensure the simulation setup was accurate. This section briefly explains the basic calculations performed to validate the engine's design and to set a reliable foundation for further optimization work.

To assess whether the single-cylinder motorcycle engine is susceptible to choking at its peak operating speed, a series of theoretical calculations was conducted. Choking occurs when the airflow through the engine's intake or exhaust valves becomes restricted, limiting the engine's ability to breathe efficiently at high RPMs.

4.3.1 Choke Analysis Calculations

Given the engine's stroke length of 76.4 mm and a maximum speed of 7500 rpm, the mean piston speed (U_p) is calculated as follows:

$$U_p = 2 \times \mathrm{S} \times \frac{\mathrm{n}}{60} \tag{4.4}$$

$$U_p = 2 \times 0.0764\,\mathrm{m} \times \frac{7500}{60} = 19.1\,\mathrm{m/s}$$

Assuming the Critical Flow Parameters and the following coefficients:

- Max Brake Mean Effective Pressure ($BMEP$) = 11.9 bar
- Critical gulp factor (Z_{crit}) = 0.6
- Mean Flow Coefficient (C_{mi}) = 0.4
- Speed of sound in air (a) = 343 m/s

The ratio of valve area to cylinder area (A_v/A_{cyl}) is determined using the formula:

$$\frac{A_v}{A_{\text{cyl}}} = \frac{U_p}{C_{mi} \times a \times Z_{\text{crit}}} \tag{4.5}$$

$$\frac{A_v}{A_{\text{cyl}}} = \frac{19.1}{0.4 \times 343 \times 0.6} \approx 0.23$$

The minimum ratio between the overall flow area and to cylinder section area $\left(\frac{A_v}{A_{\text{cyl}}}\right)_{\text{min}}$ is calculated as follows:

$$\left(\frac{A_v}{A_{\text{cyl}}}\right)_{\text{min}} = 2\left(\frac{d_{iv}}{B}\right)^2 \tag{4.6}$$

Assuming the ratio between the inlet valve diameter and to Bore (d_{iv}/B) = 0.34

$$\left(\frac{A_v}{A_{\text{cyl}}}\right)_{\text{min}} = 2 \times (0.34)^2 = 0.23$$

The calculated valve area to cylinder area ratio is approximately 0.23, which closely matches the minimum required ratio of 0.23. This alignment confirms that the valve sizing is adequate, ensuring the engine operates without choking at its maximum speed of 7500 rpm.

4.3.2 Engine Performance Calculations

The theoretical calculations for the engine's maximum torque, volumetric efficiency, and maximum power have been performed to assess its performance characteristics.

The maximum torque (T_{max}) is determined using the Brake Mean Effective Pressure (BMEP) and the engine's displacement volume (V_d) with the formula:

$$T_{max} = \frac{BMEP_{max} \times V_d}{2\pi \times \varepsilon} \tag{4.7}$$

$$T_{\max} = \frac{11.9 \times 10^5 \times 0.0005728}{2\pi \times 2} = 54.59\,\text{Nm}$$

Volumetric efficiency (λ_v) is a measure of the engine's ability to fill its cylinders with an air-to-fuel mixture. It is calculated using the formula:

$$\lambda_v = \frac{T_{\max} \times \alpha \times 2\pi\varepsilon}{\eta_b \times V_d \times H_i \times \rho_a} \tag{4.8}$$

Where:

- $\alpha = 13.5$ (Air-to-fuel ratio)
- $\eta_b = 0.37$ (Engine Brake efficiency for naturally aspirated engines)
- $H_i = 43 \times 10^6\,\text{J/kg}$ (Fuel Lower Calorific value)
- $\rho_a = 1.2\,\text{kg/m}^3$ (Air density)

$$\lambda_v = \frac{54.59 \times 13.5 \times 2\pi \times 2}{0.37 \times 0.0005728 \times 43 \times 10^6 \times 1.2} \approx 1.03$$

A volumetric efficiency of 1.03 indicates that the engine is effectively filling its cylinders, which is typical for high-performance naturally aspirated engines.

The maximum power output ($P_{\max}$) is calculated considering the engine's volumetric efficiency at maximum power conditions:

$$P_{\max} = \eta_b \times \lambda_v \times \frac{V_d \times \rho_a}{\alpha} \times H_i \times \frac{n}{\varepsilon} \tag{4.9}$$

Where:

- $\eta_b = 0.25$
- $\lambda_v = 0.9$
- $n = 7500\,\mathrm{RPM}$

The volumetric efficiency and the engine's brake efficiency are slightly reduced and considered in the calculation due to the operation of the engine at higher rpm.

$$P_{\max} = 0.25 \times 0.9 \times \frac{0.0005728 \times 1.2}{13.5} \times 43 \times 10^6 \times \frac{7500}{60 \times 2} \approx 31\,\mathrm{kW}$$

The torque values derived from our theoretical calculations align closely with the simulation results. However, the calculated power output shows some deviation from the simulated values. This discrepancy is primarily due to the assumptions regarding brake thermal efficiency and volumetric efficiency. Since power output is highly sensitive to these parameters, having an accurate efficiency map would enable more precise predictions. Thus, the baseline configurations of the engine are validated.

To validate the accuracy and effectiveness of the optimization framework, a hand-calculated optimal duct length was computed at a representative engine speed of 4000 rpm. This rpm was chosen as a test point because it corresponds to the engine's peak torque output, making it a critical condition for assessing wave tuning behavior.

The duct length was estimated using the formula:

$$L = \frac{a}{8 \times n} \tag{4.10}$$

$$L = \frac{343}{8 \times 66.67} = 0.64\,\mathrm{m}$$

At 4000 rpm (approximately 66.67 revolutions per second), the resulting duct length of 0.64 m serves as a theoretical reference. This value is used to cross-check the optimizer's suggested length at the same engine speed. By comparing the two, we can determine whether the optimizer reliably predicts wave-tuned duct geometry. It’s important to note that this is only a single validation point. A complete optimization strategy would involve similar comparisons across multiple engine speeds to ensure optimal performance throughout the entire operating range.

4.4 Inlet Duct Length Optimization of the Single-Cylinder Engine

This section briefs the inlet duct length optimization process for the single-cylinder engine, specifically conducted at an engine speed of 4000 rpm. The optimization was carried out using both manual calculations and simulation-based methods. For the manual analysis, the speed of sound was assumed to be 343 m/s, corresponding to standard atmospheric conditions, and was applied in the quarter-wave resonance formula to estimate the ideal duct length. To maintain consistency and validate the approach, the same engine speed (4000 rpm) and sound speed value were used in the *Gasdyn* simulation. The inlet duct, including any configurations involving multiple sub-elements, was modeled accordingly. The geometrical details of the inlet duct elements, such as left and right diameters and lengths, are presented in *Table 5*.

Duct name	Subelement name	Left diameter [m]	Right diameter [m]	Duct length [m]
duct_240	subelement_242	0.065	0.065	0.03
duct_1035	subelement_1037	0.055	0.055	0.05
duct_250	subelement_252	0.055	0.055	0.02
	subelement_490	0.055	0.043	0.04
	subelement_491	0.043	0.041	0.02
	subelement_492	0.041	0.041	0.01
duct_260	subelement_262	0.041	0.041	0.01
	subelement_493	0.041	0.043	0.04

	subelement_494	0.043	0.043	0.04
duct_453	subelement_455	0.031	0.031	0.06
duct_463	subelement_465	0.031	0.031	0.06

Table 5 – Inlet Duct's Geometrical Data

Figure 47 illustrates the engine's schematic layout within the *Gasdyn* environment, with the inlet ducts highlighted for reference.

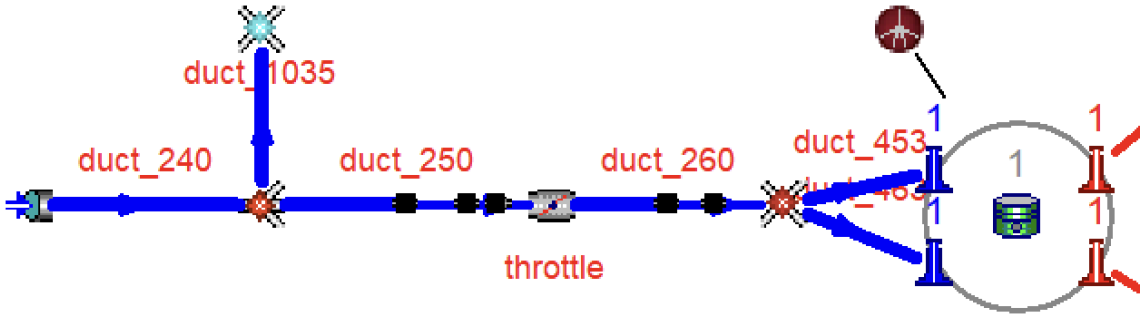

Figure 47 - Schematic Layout Highlighting the Engine's Inlet Duct

In the optimization of the single-cylinder engine's inlet duct length, particular attention was given to the segments directly connected to the engine head, specifically, duct_453 and duct_463. These ducts are integral to the engine's structural design and are typically not subject to modification. Therefore, their lengths were preserved in both manual calculations and simulation-based analyses.

Total existing inlet duct length (including duct_453 and duct_463): 0.38 m

Fixed duct lengths (duct_453 + duct_463): 0.12 m

Modified existing duct length (excluding fixed segments): 0.38 m − 0.12 m = ***0.26 m***

Hand-calculated optimal total inlet duct length: 0.64 m

Modified hand calculated length (excluding fixed segments): 0.64 m − 0.12 m = ***0.52 m***

This approach ensures that the optimization process focuses solely on the modifiable sections of the inlet duct, maintaining the integrity of the engine's core components while enhancing

performance through precise adjustments. The basic discretized configuration of the chosen inlet ducts for optimization is illustrated in the following figures.

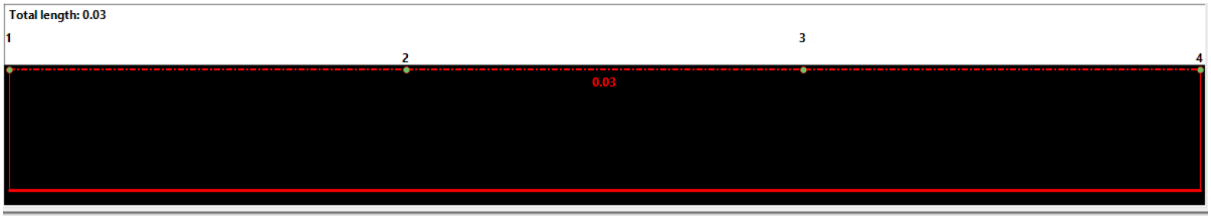

Figure 48 - Basic discretized configuration of duct_240

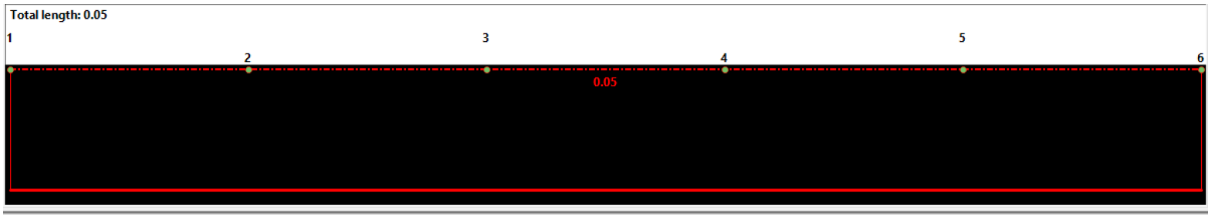

Figure 49 - Basic discretized configuration of duct_1035

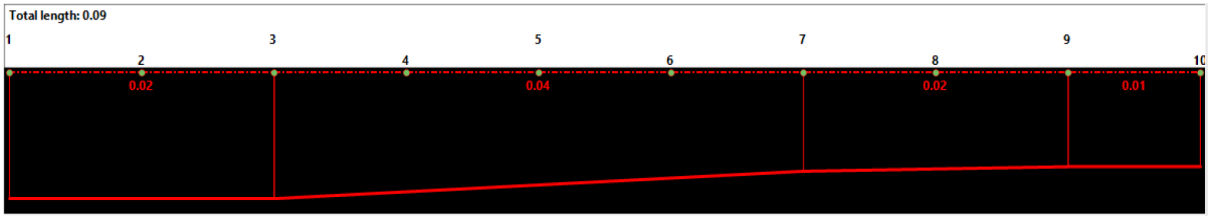

Figure 50 - Basic discretized configuration of duct_250 consisting of 4 subelements

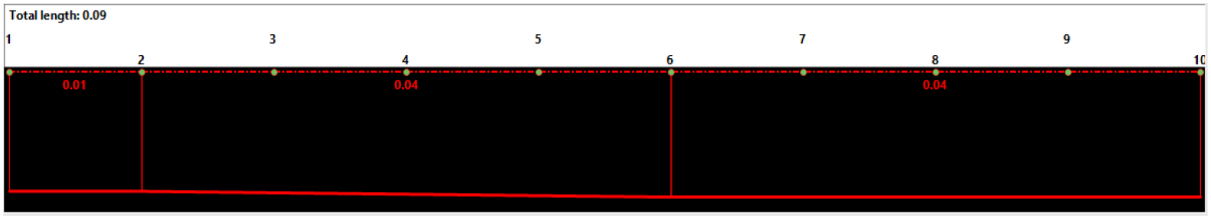

Figure 51 - Basic discretized configuration of duct_260 consisting of 3 subelements

After setting up the engine configuration, we generated the necessary input files using *Gasdyn*. With these files ready, we launched the *engine_optimizer* tool. By feeding in the *Gasdyn* input files and executable, we initiated the duct length optimization process, as outlined in Section 3.6.1.

For this optimization, the following test criteria were defined:

- Minimum duct length: **0.1 m**
- Maximum duct length: **2.0 m**
- Design of Experiments (DoE) points: **5**

Based on these parameters, the optimizer conducted simulations at the following duct lengths:

- 0.10 m
- 0.58 m
- 1.05 m
- 1.52 m
- 2.00 m

The simulation was carried out with the objective of maximizing the engine's volumetric efficiency. After the optimization was completed, the simulation and the optimization results were saved in the *final_results.csv* file as shown in Figures 52 & 53, respectively.

final_results.csv

```
Type,RPM,Duct_Length,DOE,Vol.eff_[%]_max,Br.torque_[Nm]_max,BSFC_[g/kWh]_min,BMEP_[bar]_max
DoE test,4000.000,0.100,1,92.300,52.600,293.700,11.480
DoE test,4000.000,0.575,2,96.500,55.300,283.000,12.060
DoE test,4000.000,1.050,3,98.600,57.100,270.200,12.450
DoE test,4000.000,1.525,4,100.500,58.500,266.700,12.750
DoE test,4000.000,2.000,5,90.700,51.600,269.900,11.250
```

Figure 52 - Duct length Simulation Results

```
# Optimal Solution
Best Duct Length (m): 1.52
Best VE: 100.5

# Search Statistics
total_evaluations: 272
cache_hits: 105
cache_hit_rate: 0.3860294117647059
initial_mesh_size: 0.532
final_mesh_size: 0.665
iterations: 25
lb: -0.28
ub: 2.38
range: 2.66
dimension: 1
best_value: 98.676
```

Figure 53 - Duct length: Optimal results

The optimal inlet duct length determined by the optimizer is ***1.52 m***, specifically optimized for engine operation at **4000 rpm**. Notably, when the optimization objective was shifted from maximizing volumetric efficiency to maximizing torque, the optimizer still identified 1.52 meters as the optimal duct length. This consistency suggests that, for this engine configuration and operating speed, the same duct length effectively enhances both volumetric efficiency and torque.

The optimization process, including the distribution of Design of Experiments (DoE) points, the evaluated optimization points, and the identified optimal point, is illustrated in *Figure 54*.

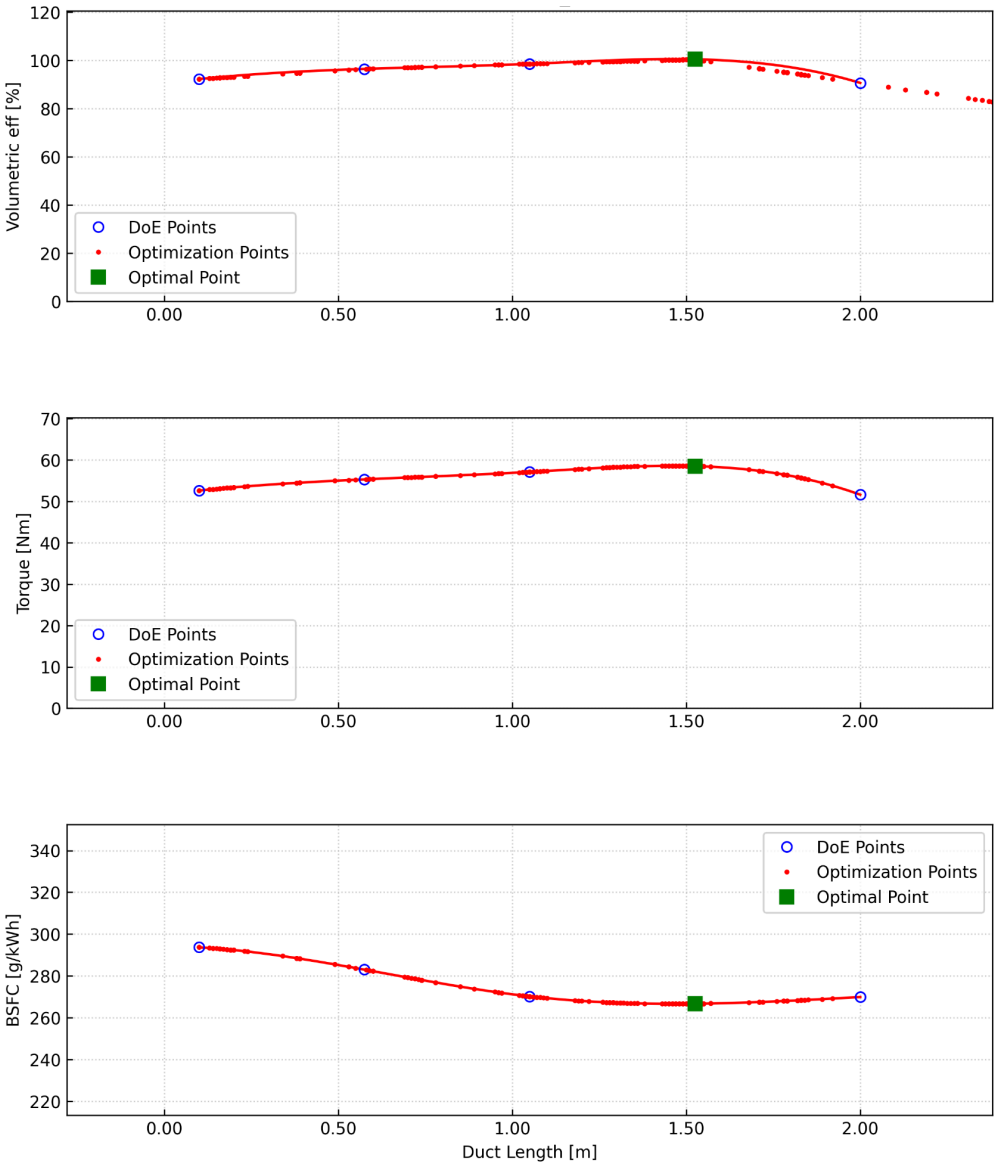

Figure 54 - Duct Length Optimization Plots: Volumetric Efficiency, Torque, and BSFC vs. Duct Length

The original inlet duct length of the engine was *0.26 m*. Based on optimization results, this length was scaled up to *1.52 m*. To achieve this, each subelement of the intake duct system was proportionally adjusted according to its original length. These modifications were implemented in the *Gasdyn* engine configuration. Subsequently, simulations were conducted across all operating points using the updated optimal duct length. This approach aimed to evaluate how the engine's performance metrics respond to the optimized duct length throughout its operating range. The results from these simulations are compiled and presented in *Table 6*.

RPM	Volumetric efficiency (%)	Break Torque (Nm)	Brake Power (kW)	BSFC (g/kWh)
1500	84.9	45.8	7.2	346.1
2000	95.5	53.7	11.2	323.1
2500	93.9	53.8	14.1	290.6
3000	108.8	63.7	20.0	275.9
3500	105.3	61.2	22.4	270.6
4000	105.0	60.3	25.3	267.4
4500	87.6	47.6	22.4	290.5
5000	82.7	45.1	23.6	282.2
5500	83.0	45.3	26.1	285.4
6000	79.3	42.5	26.7	286.7
6500	79.7	42.3	28.8	289.0
7000	78.3	40.4	29.6	295.5
7500	65.8	30.6	24.0	328.0
8000	45.5	15.7	13.1	442.5
8500	35.0	6.8	6.1	781.2

Table 6 - Engine's Performance Parameters Obtained from the *Gasdyn* Simulation Results at optimal duct length 1.52m

The simulation results from the original inlet duct length of 0.26 m, presented in *Table 3*, are compared with the results from the optimized inlet duct length of 1.52 m, shown in *Table 6*. This comparison highlights the effect of inlet duct length on overall engine performance. The corresponding performance graphs, Brake Torque vs. RPM, Brake Power vs. RPM, Volumetric Efficiency vs. RPM, and Brake Specific Fuel Consumption (BSFC) vs. RPM, are illustrated in *Figures 55, 56, 57*, and *58*, respectively, providing a clear visual comparison across the engine's operating range.

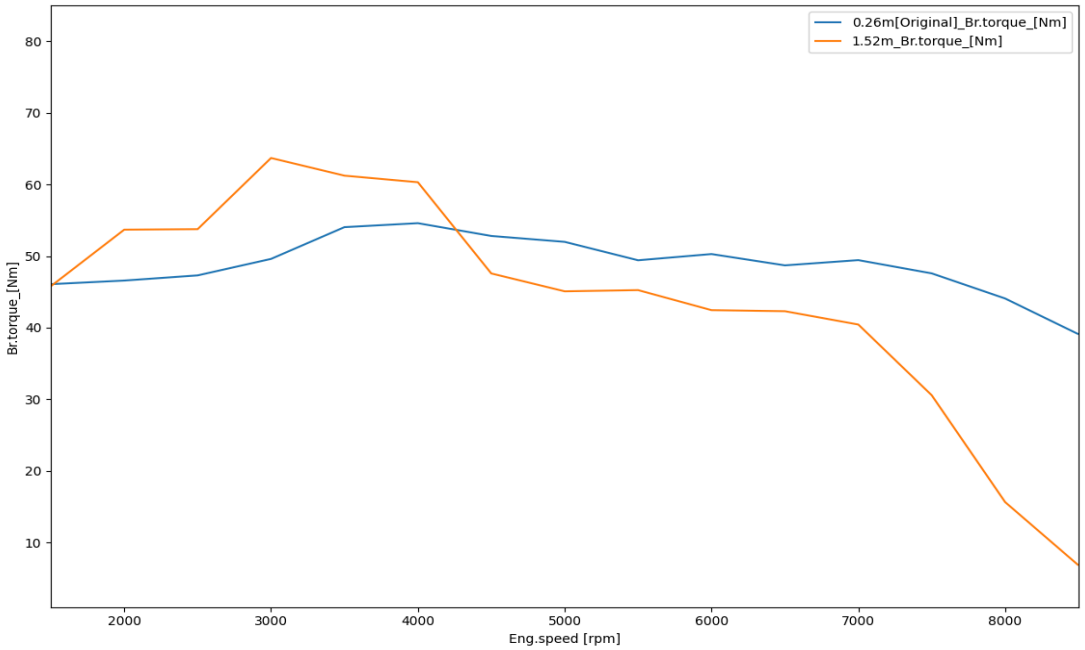

Figure 55 - Comparison of Brake Torque vs. RPM for 0.26 m and 1.52 m Inlet Duct Lengths

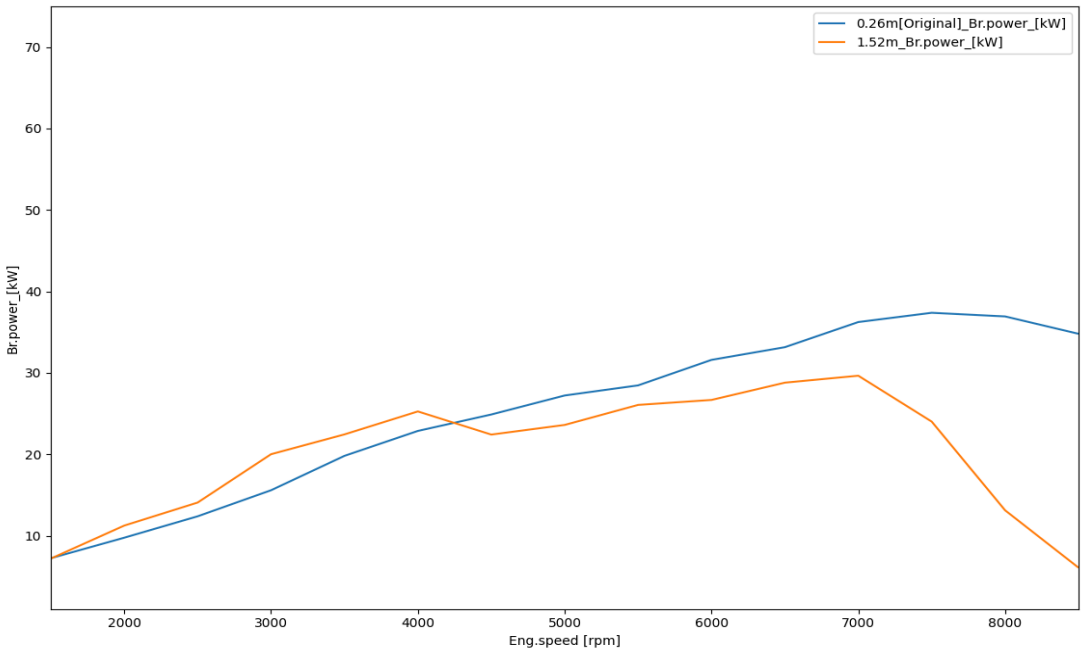

Figure 56 - Comparison of Brake Power vs. RPM for 0.26 m and 1.52 m Inlet Duct Lengths

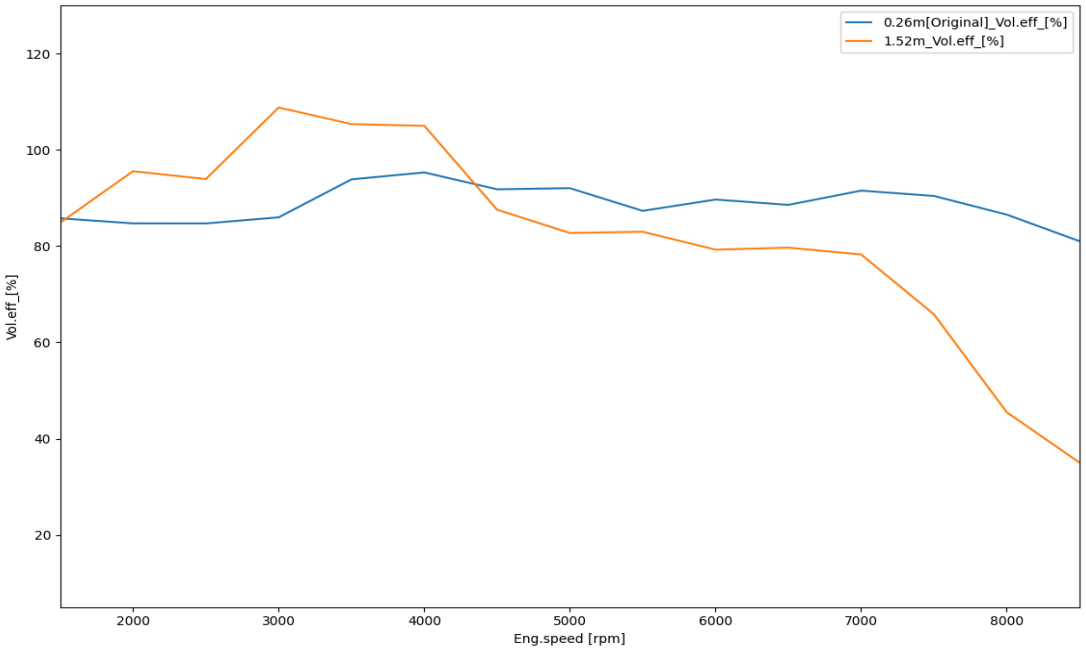

Figure 57 - Comparison of Volumetric efficiency vs. RPM for 0.26 m and 1.52 m Inlet Duct Lengths

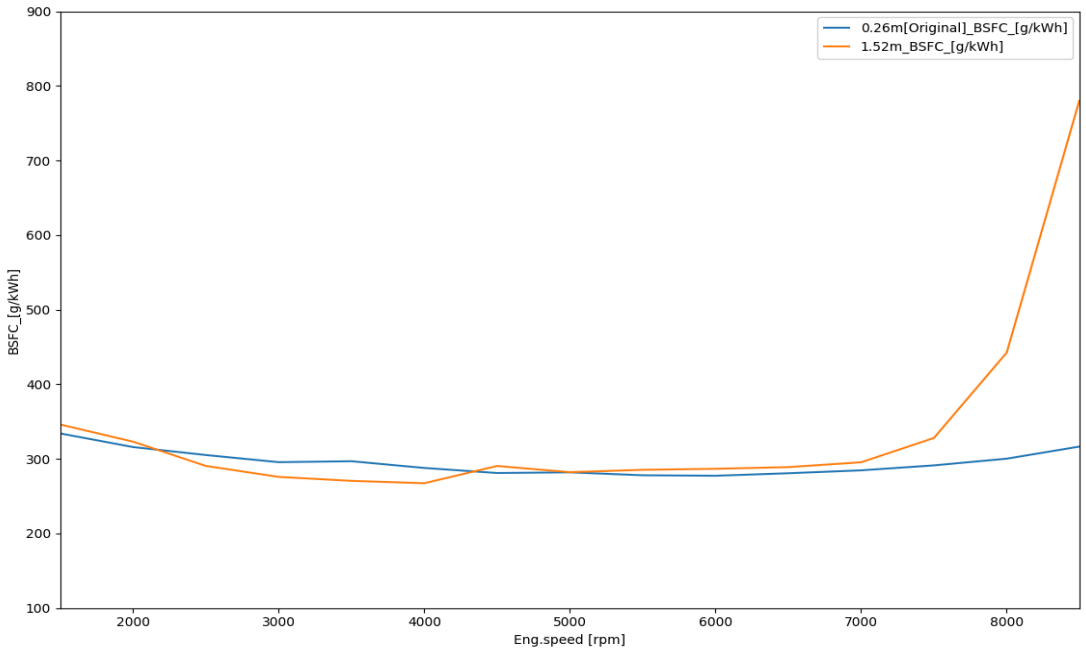

Figure 58 - Comparison of Brake Specific Fuel Consumption (BSFC) vs. RPM for 0.26 m and 1.52 m Inlet Duct Lengths

Increasing the inlet duct length to the optimized value of *1.52 m*, specifically tailored for *4000 rpm*, led to noticeable improvements in performance within the 3000–4000 rpm range. In this window, volumetric efficiency and brake torque both increased, while brake-specific fuel consumption (BSFC) dropped, demonstrating that the optimizer successfully achieved its objective. However, when examining performance at higher engine speeds, such as 6500, 7000, and 7500 rpm, the results were significantly less favorable. Volumetric efficiency and brake torque declined, BSFC increased, and overall brake power dropped below the levels achieved with the original 0.26 m duct length.

This drop in performance at higher RPMs is a direct result of optimizing for a fixed point, 4000 rpm. The optimizer did exactly what it was instructed to do: identify the duct length that maximizes volumetric efficiency at that specific speed. However, this highlights the importance of engineering judgment and application context. In real-world scenarios, engines often operate over a wide RPM range, and focusing optimization on a single point can lead to suboptimal results elsewhere.

Although we later attempted to use the optimizer to simulate variable valve timing across different operating points, a method discussed in the next section, we were still unable to achieve significant improvements at higher RPMs. This confirms that while the 1.52 m duct length is ideal for the targeted mid-range speed, it is not suitable for higher-speed operation, moreover, real-world physical constraints, such as limited space in a motorcycle, would restrict the practical implementation of such long duct lengths, regardless of the optimizer's suggestion. This outcome underscores the need for proper engineering supervision in optimization tasks, ensuring that design decisions are aligned with the engine's intended use case and operating conditions.

The next step was to run simulations across the selected operating range of the engine using the hand-calculated optimal inlet duct length of 0.52 m. This was done to validate whether the manually computed length could effectively improve engine performance. The results from these simulations have been compiled and are presented in *Table 7.*

RPM	Volumetric efficiency (%)	Break Torque (Nm)	Brake Power (kW)	BSFC (g/kWh)
1500	85.6	46.0	7.2	335.6
2000	85.2	46.8	9.8	321.7
2500	86.3	48.4	12.7	308.9
3000	89.0	51.7	16.2	296.6
3500	100.5	58.4	21.4	290.8
4000	100.0	57.7	24.2	280.9
4500	102.1	59.3	27.9	273.3
5000	97.5	56.0	29.3	271.7
5500	90.4	50.9	29.3	273.6
6000	91.5	51.5	32.3	272.5
6500	87.1	47.6	32.4	280.1
7000	87.1	46.3	34.0	287.1
7500	85.9	44.3	34.8	295.8
8000	77.1	37.1	31.1	317.0
8500	75.7	34.9	31.0	330.8

Table 7 - Engine's Performance Parameters Obtained from the *Gasdyn* Simulation Results at optimal duct length 0.52m

The results are now compared between two cases: the simulation using the *engine_optimizer*'s modified optimal inlet duct length of 1.52 m, as shown in *Table 6,* and the simulation using the hand-computed optimal inlet duct length of 0.52 m, as shown in *Table 7*. This comparison helps evaluate which configuration delivers better performance across the operating range. The corresponding comparison graphs for Brake Torque vs. RPM, Brake Power vs. RPM, Volumetric Efficiency vs. RPM, and Brake Specific Fuel Consumption (BSFC) vs. RPM are illustrated in *Figures 59, 60, 61,* and *62,* respectively.

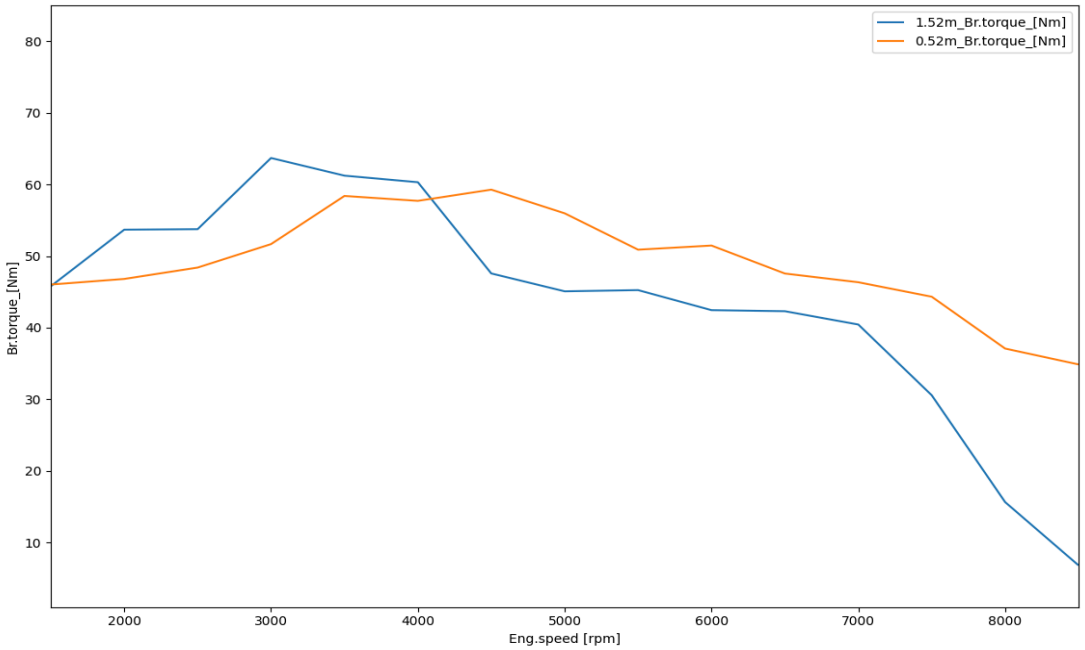

Figure 59 - Comparison of Brake Torque vs. RPM for 1.52 m and 0.52 m Inlet Duct Lengths

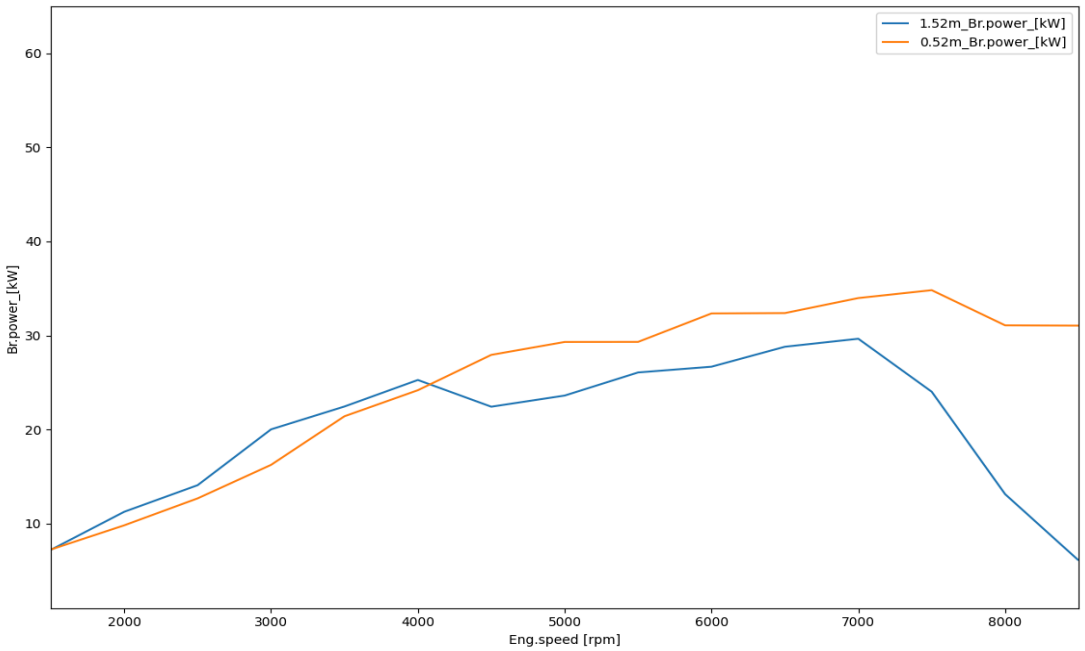

Figure 60 - Comparison of Brake Power vs. RPM for 1.52 m and 0.52 m Inlet Duct Lengths

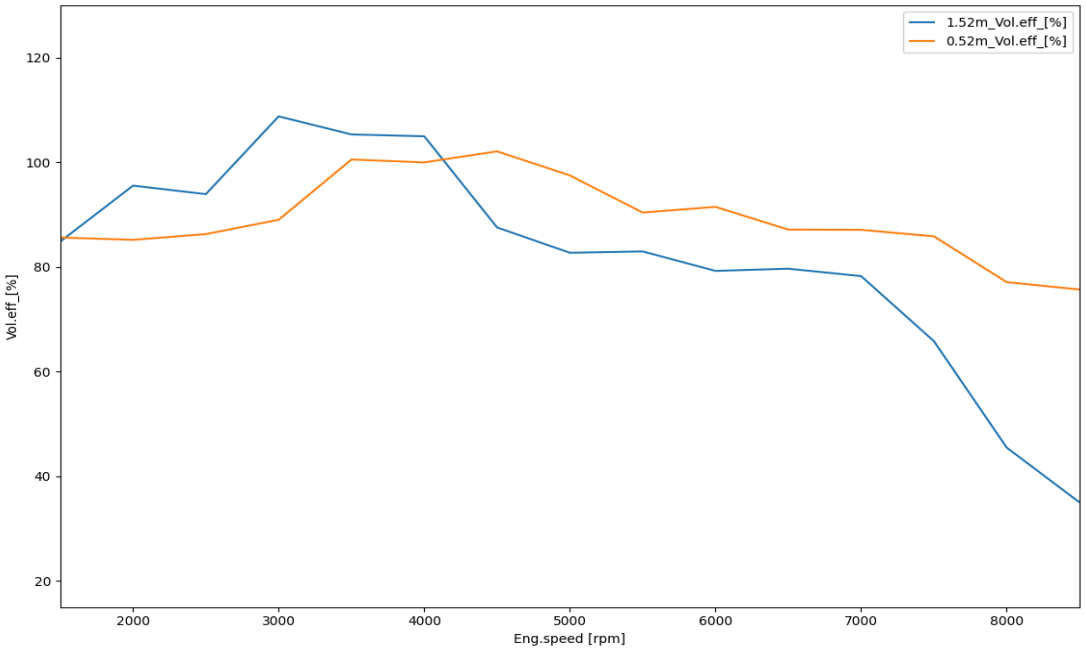

Figure 61 - Comparison of Volumetric efficiency vs. RPM for 1.52 m and 0.52 m Inlet Duct Lengths

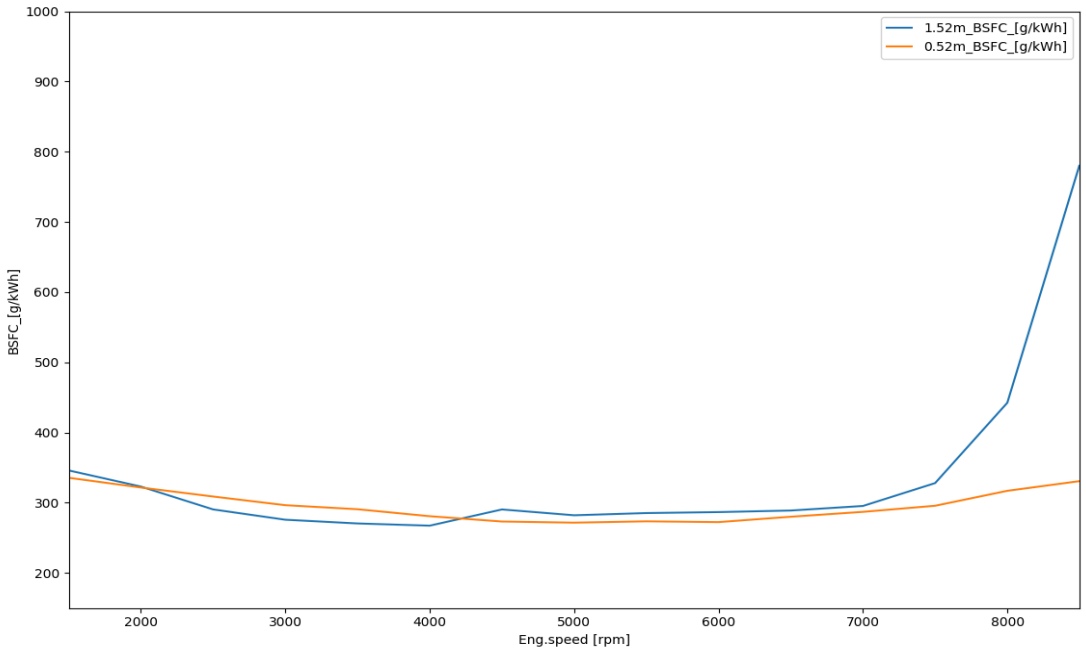

Figure 62 - Comparison of Brake Specific Fuel Consumption (BSFC) vs. RPM for 1.52 m and 0.52 m Inlet Duct Lengths

The simulation results reveal that the *1.52 m* inlet duct length, optimized specifically for *4000 rpm*, delivers superior performance within the 3500–4500 rpm range. In this band, volumetric efficiency and brake torque are significantly higher, while brake-specific fuel consumption (BSFC) is lower compared to the performance achieved with the 0.52 m hand-calculated duct length. These gains are primarily due to the longer duct's ability to better harness pressure wave tuning at the target RPM, improving air charge delivery into the cylinders, and enhancing combustion efficiency.

However, beyond 4500 rpm, especially at higher engine speeds such as 6500, 7000, and 7500 rpm, the performance of the 1.52 m duct drops off sharply. At these speeds, the engine exhibits a steep decline in both torque and volumetric efficiency, accompanied by a substantial increase in BSFC. This decline occurs because the resonance tuning of the longer duct no longer aligns with the faster intake cycles at high RPMs, resulting in reduced cylinder filling efficiency. On the other hand, the 0.52 m duct length provides more stable performance across a wider RPM range. While it does not match the peak performance of the 1.52 m duct at 4000 rpm, it maintains stronger torque and better fuel efficiency at higher speeds. These findings underscore the importance of aligning intake duct design with the engine's intended operating range. While optimization at a specific RPM can yield significant benefits within that narrow band, it may compromise performance outside of it.

Now the results are now compared between the simulation using the original inlet duct length of 0.26 m as presented in *Table 3* and the simulation using the hand-computed optimal inlet duct length of 0.52 m as shown in *Table 7*. This comparison helps evaluate the performance improvement achieved through manual optimization. The corresponding comparison plots for Brake Torque vs. RPM, Brake Power vs. RPM, Volumetric Efficiency vs. RPM, and Brake Specific Fuel Consumption (BSFC) vs. RPM are illustrated in *Figures 63, 64, 65,* and *66*, respectively.

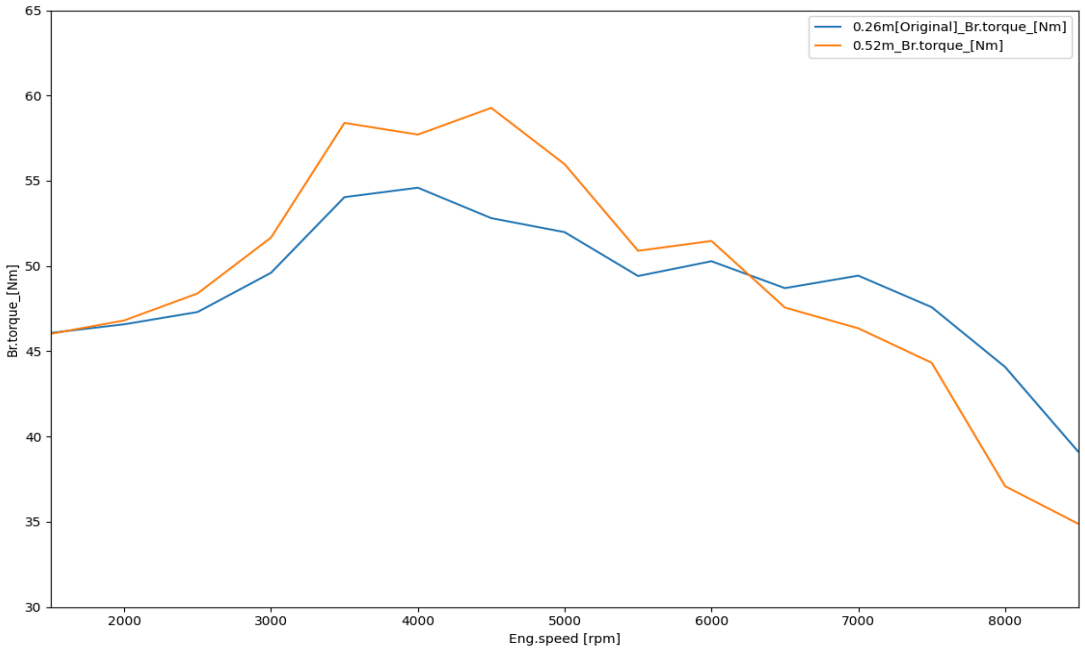

Figure 63 - Comparison of Brake Torque vs. RPM for 0.26 m and 0.52 m Inlet Duct Lengths

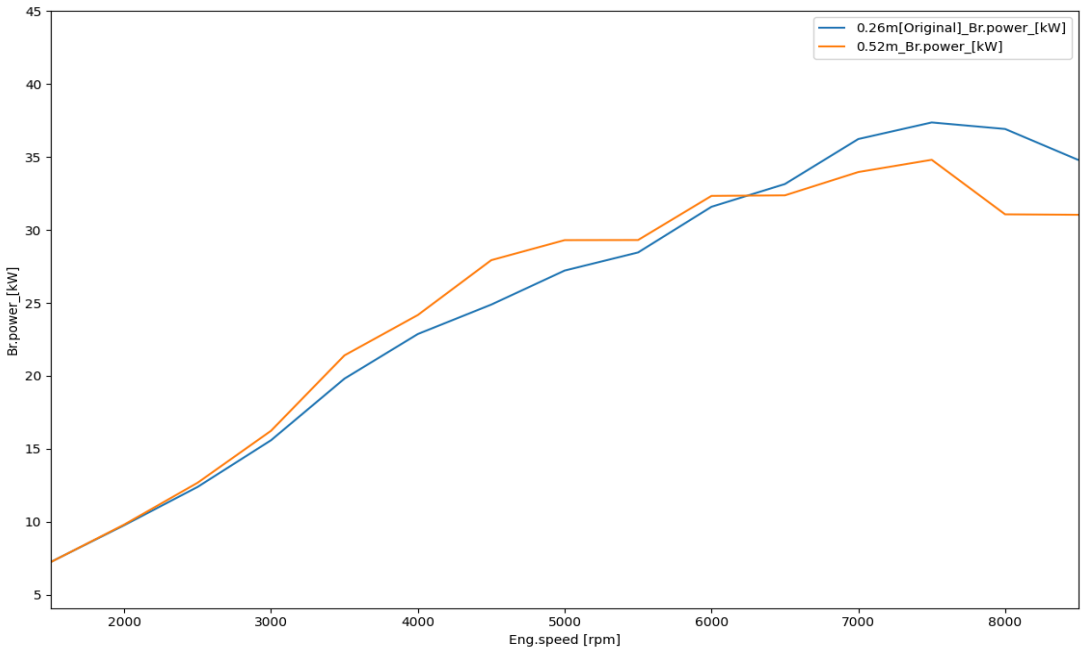

Figure 64 - Comparison of Brake Power vs. RPM for 0.26 m and 0.52 m Inlet Duct Lengths

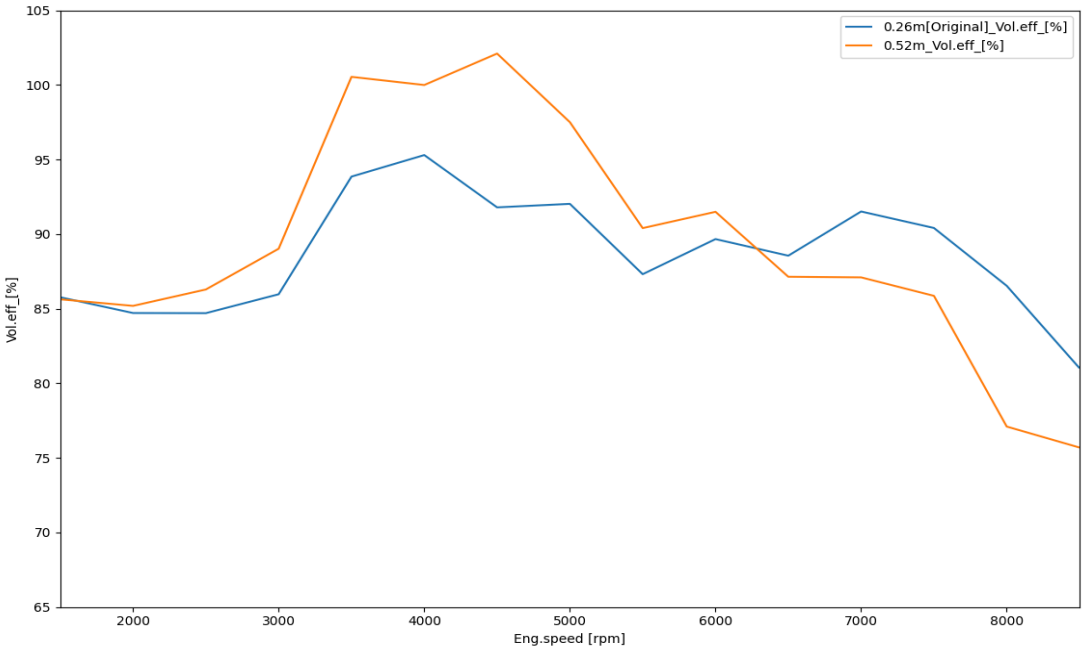

Figure 65 - Comparison of Volumetric efficiency vs. RPM for 0.26 m and 0.52 m Inlet Duct Lengths

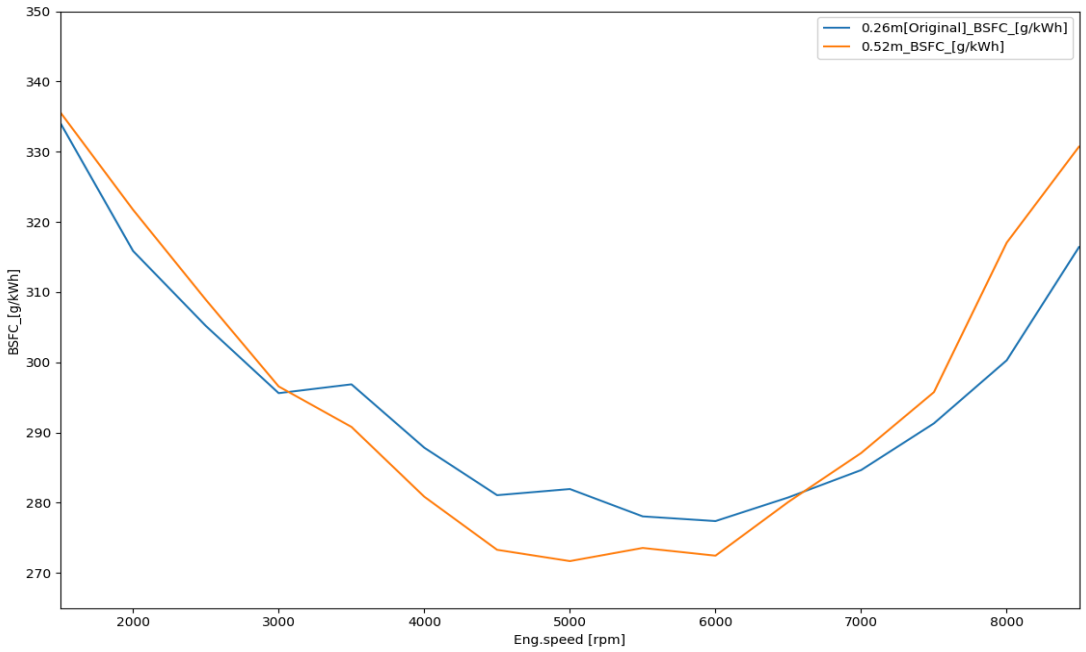

Figure 66 - Comparison of Brake Specific Fuel Consumption (BSFC) vs. RPM for 0.26 m and 0.52 m Inlet Duct Lengths

The simulation results indicate that increasing the inlet duct length to 0.52 m enhances engine performance within the 3500 – 4500 rpm range. Specifically, volumetric efficiency and brake torque are higher, and Brake Specific Fuel Consumption (BSFC) is lower compared to the original 0.26 m duct length. At higher engine speeds, the 0.52 m duct length exhibits a slight reduction in volumetric efficiency and brake torque, along with a modest increase in BSFC. However, these changes are not substantial and still represent a reasonable performance across a broader RPM range. This balance makes the 0.52 m duct length a suitable choice for applications where optimal performance is desired in the 3500 – 4500 rpm range, without significantly compromising efficiency at higher speeds.

It's important to note that the effectiveness of this duct length is contingent upon the specific application and operating conditions of the engine. Therefore, engineering judgment should be applied to ensure that the intake system design aligns with the intended use and performance requirements of the engine.

4.5 Variable Valve Timing Optimization of the Single-Cylinder Engine

Building upon the improved performance observed with the *0.52 m* inlet duct length, particularly within the 3500 – 4500 rpm range and maintaining reasonable metrics at higher RPMs, we proceeded to implement Variable Valve Timing (VVT) optimization. This approach aimed to further enhance engine performance across a broader RPM spectrum. To achieve the desired 0.52 m duct length, proportional scaling was applied to each subelement of the intake duct, ensuring uniform modification while preserving the original geometry near the engine head. The updated configuration was updated into the *Gasdyn* engine modelling environment, and corresponding input files were generated for the *engine_optimizer*.

The primary objective of this VVT optimization was to assess potential improvements in performance metrics, such as volumetric efficiency, brake torque, and Brake Specific Fuel Consumption (BSFC), across various engine speeds. By adjusting valve timing parameters, we aimed to align the engine's performance at higher RPMs more closely with that achieved using the original 0.26 m inlet duct length. Implementing VVT can significantly influence engine performance. Studies have demonstrated that VVT can increase volumetric efficiency by 7 – 8% at high RPMs, leading to enhanced torque and reduced fuel consumption. By

optimizing valve timing, the engine can achieve better air-fuel mixture filling, improving combustion efficiency and overall performance. This optimization process is particularly beneficial for engines operating across a wide RPM range, as it allows for dynamic adjustment of valve events to match varying operating conditions. Consequently, the engine can maintain optimal performance and efficiency throughout its operating spectrum.

As detailed in *Section 3.6.2* on *Valve Timing Optimization*, the updated input files and executable were provided to the *engine_optimizer*, which then displayed the current valve timings of the engine, as shown in *Figure 67*.

```
1. Duct length optimization
2. Valve timing optimization
3. A/F ratio optimization
4. Duct and valve optimization
5. Duct diameter optimization

Enter your choice (1/2/3/4/5): 2

Original valve timings:
Intake valve(s): 300.0
Exhaust valve(s): 75.0
```

Figure 67 - *engine_optimizer* displaying Valve Timings of the Single-Cylinder Engine

To optimize valve timing across the engine's operating range, simulations were conducted at each RPM point: 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, and 8500 RPM. For each of these points, the following valve timing parameters were defined:

- Intake Valve Opening (IVO): Minimum 290°, Maximum 330°
- Exhaust Valve Opening (EVO): Minimum 65°, Maximum 100°
- Design of Experiments (DOE): 3 levels

This setup resulted in nine combinations of IVO and EVO angles for each RPM point:

1. IVO 290°, EVO 65°
2. IVO 290°, EVO 82°
3. IVO 290°, EVO 100°

4. IVO 310°, EVO 65°
5. IVO 310°, EVO 82°
6. IVO 310°, EVO 100°
7. IVO 330°, EVO 65°
8. IVO 330°, EVO 82°
9. IVO 330°, EVO 100°

These combinations were tested through simulation using the *engine_optimizer* to evaluate their effect on key performance metrics. The objective of this process was not only to understand how valve timing affects performance at different speeds but also to determine the optimal valve timing settings for each RPM point. With these optimal values identified, the next step involves implementing Variable Valve Timing (VVT). This allows the valve timings to adjust dynamically across the RPM range, aiming to enhance overall engine performance by maintaining better efficiency and torque throughout the entire operating spectrum.

To demonstrate the valve timing optimization process, the results for 7500 rpm are presented as an example. The same procedure and structure were followed for all other operating points to find the optimal valve timing at each engine speed. The simulation was carried out with the objective of maximizing volumetric efficiency. Once the optimization was completed, the simulation data and corresponding optimization results were saved in the *final_results.csv* file. These simulation results are illustrated in *Figure 68*.

final_results.csv

```
Type,RPM,Intake_Timing,Exhaust_Timing,DOE,Vol.eff_[%]_max,Br.torque_[Nm]_max,BSFC_[g/kWh]_min,BMEP_[bar]_max
DoE test,7500.000,290.000,65.000,1,68.200,31.700,327.600,6.920
DoE test,7500.000,290.000,82.500,2,84.100,42.900,299.800,9.360
DoE test,7500.000,290.000,100.000,3,85.400,42.900,306.200,9.350
DoE test,7500.000,310.000,65.000,4,84.300,42.700,302.500,9.300
DoE test,7500.000,310.000,82.500,5,91.300,48.000,292.300,10.460
DoE test,7500.000,310.000,100.000,6,92.500,47.800,299.000,10.410
DoE test,7500.000,330.000,65.000,7,86.400,43.700,308.100,9.530
DoE test,7500.000,330.000,82.500,8,90.300,47.200,298.700,10.290
DoE test,7500.000,330.000,100.000,9,89.900,46.200,303.600,10.080
```

Figure 68 - Valve Timing Simulation Results at 7500 rpm

Figures 69 and *70* present the 3D surface and contour plots of Volumetric Efficiency vs. RPM, based on the Design of Experiments (DoE) results. Similar sets of plots were also

generated for Brake Torque and BSFC, using the same DoE points to visualize the optimization trends.

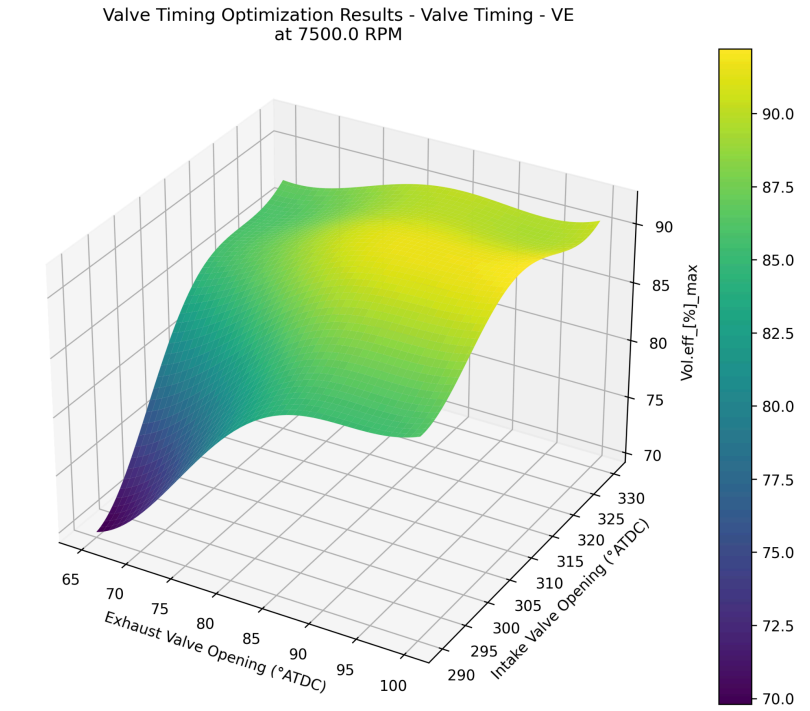

Figure 69 - 3D Plot showing the Volumetric Efficiency vs RPM for the DoE points for Valve Timing Optimization at 7500 rpm

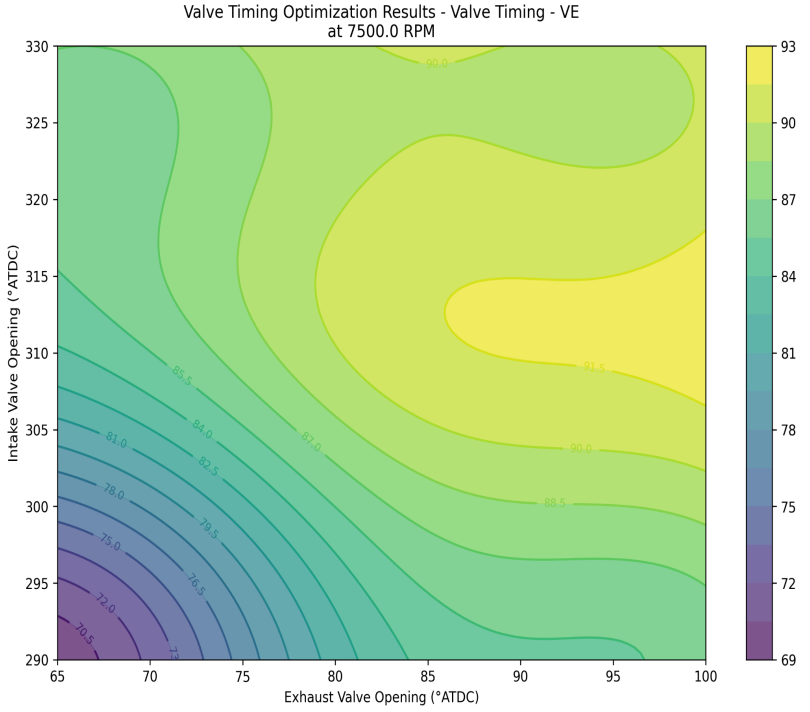

Figure 70 - Contour Plot showing the Volumetric Efficiency vs RPM for the DoE points for Valve Timing Optimization at 7500 rpm

At **7500 rpm**, the optimizer determined the optimal valve timing to be ***IVO = 282°*** and ***EVO = 107°***. These results were saved in the *final_results.csv* file and are illustrated in *Figure 71*.

```
# Optimal Solution
Best Intake Timing (°): 282
Best Exhaust Timing (°): 107
Best VE: 95.2

# Search Statistics
total_evaluations: 399
cache_hits: 171
cache_hit_rate: 0.42857142857142855
initial_mesh_size: [5.6 4.9]
final_mesh_size: [14.   12.25]
iterations: 25
lb: [282.  58.]
ub: [338. 107.]
range: [56. 49.]
dimension: 2
best_value: 93.772
```

Figure 71 - Valve Timing: Optimal results at 7500 rpm

While the optimizer identified IVO = 282° and EVO = 107° as the optimal valve timings at 7500 rpm, these values raised some practical concerns. Specifically, the IVO (Intake Valve Opening) was found to be too early, which, although it contributed to improved volumetric efficiency in the simulation, could potentially cause issues like air-fuel mixture overflow or reversion, especially at high engine speeds. It's important to note that the optimization process was carried out separately at each RPM point, meaning the optimal IVO and EVO values were selected individually for every speed. However, in an actual engine with Variable Valve Timing (VVT), valve timing transitions must occur smoothly across the RPM range. Large or abrupt changes in valve timing values from one RPM to the next can lead to mechanical instability or control challenges.

To address this, the initially optimized values were carefully reviewed and adjusted by engineers. These manual corrections ensured that the final valve timing map allowed for a smooth progression as engine speed increases, avoiding extreme or technically impractical values. The overall optimization process, including the distribution of Design of Experiments (DoE) points, the tested optimization points, and the final optimal point, is visualized in *Figure 72*.

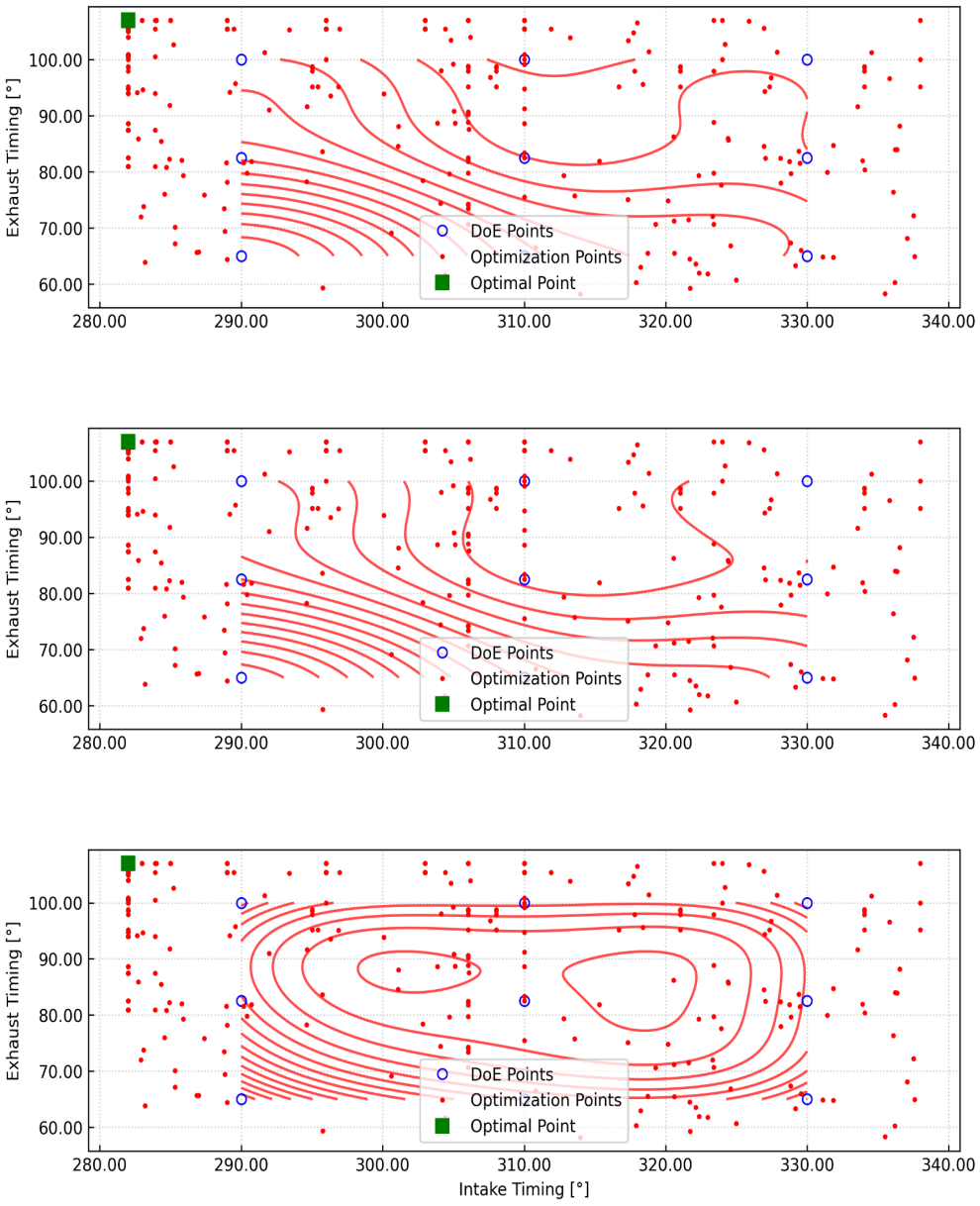

Figure 72 - Valve timing Optimization Plots: Volumetric Efficiency, Torque, and BSFC vs. EVO & IVO at 7500 rpm

The optimization results at each selected RPM point are listed in *Table 8*, and as previously explained, they were refined to ensure smooth transitions across engine speeds and to avoid potential mechanical issues. In *Gasdyn*, under the '*Valves*' section, there is an option to define Variable Valve Timing (VVT) for both intake and exhaust valves. This feature allows you to input timing variations across different RPM and load conditions. To meet *Gasdyn*'s requirements, we specified two load conditions: full load (which aligns with our simulation objective) and 90% load (to satisfy the tool's input criteria).

The VVT values are not entered as absolute angles, but rather as relative offsets from the original fixed valve timings used in the base engine setup. These relative differences are computed and input accordingly.

For example, at *3500 RPM*,

- If the original EVO is 75° and the optimized EVO is 91°, then:

  EVO offset = 91° - 75° = **+16°**

- Similarly, if the original IVO is 300° and the optimized IVO is 291°, then:

  IVO offset = 291° - 300° = **-9°**

Positive values indicate the valve opens later than in the original setup, while negative values mean it opens earlier. These relative timing values were calculated for all operating points and both load conditions, and are summarized in *Table 8*.

Engine speed (rpm)	Engine load (%)	Optimal IVO(°)	Optimal EVO(°)	IVO Variation (°)	EVO Variation (°)
1500	100	291	91	-9	16
1500	90	292	92	-8	17
2000	100	291	91	-9	16
2000	90	292	92	-8	17
2500	100	291	94	-9	19
2500	90	292	95	-8	20
3000	100	291	91	-9	16
3000	90	292	92	-8	17
3500	100	291	91	-9	16
3500	90	292	92	-8	17
4000	100	291	92	-9	17
4000	90	292	93	-8	18
4500	100	291	92	-9	17
4500	90	292	93	-8	18
5000	100	297	90	-3	15
5000	90	298	91	-2	16

5500	100	291	82	-9	7
5500	90	292	83	-8	8
6000	100	299	76	-1	1
6000	90	300	77	0	2
6500	100	301	93	1	18
6500	90	302	94	2	19
7000	100	296	85	-4	10
7000	90	297	86	-3	11
7500	100	302	91	2	16
7500	90	303	92	3	17
8000	100	310	84	10	9
8000	90	311	85	11	10
8500	100	311	85	11	10
8500	90	312	86	12	11

Table 8 - Optimized Valve timings and Relative Valve Timing variation for Inlet and Exhaust Valves at Each RPM and Load Condition

The previously calculated valve timing variations were integrated into the 1D engine model within the *Gasdyn* environment. This was accomplished by accessing the *Valves* section and navigating to the corresponding *data* window for each valve. Within these windows, there are options labeled "*EVO Variation [deg]"* for the exhaust valve and *"IVO Variation [deg]"* for the intake valve. These fields allow for the input of variable valve timing data. The variable timing values can be entered either manually or by importing a CSV file containing the relative timing offsets for each RPM and load condition. This flexibility facilitates efficient data entry and ensures accuracy in the simulation setup. *Figures 73* and *74* illustrate the data windows for the intake and exhaust variable timing inputs, respectively, with the calculated values entered.

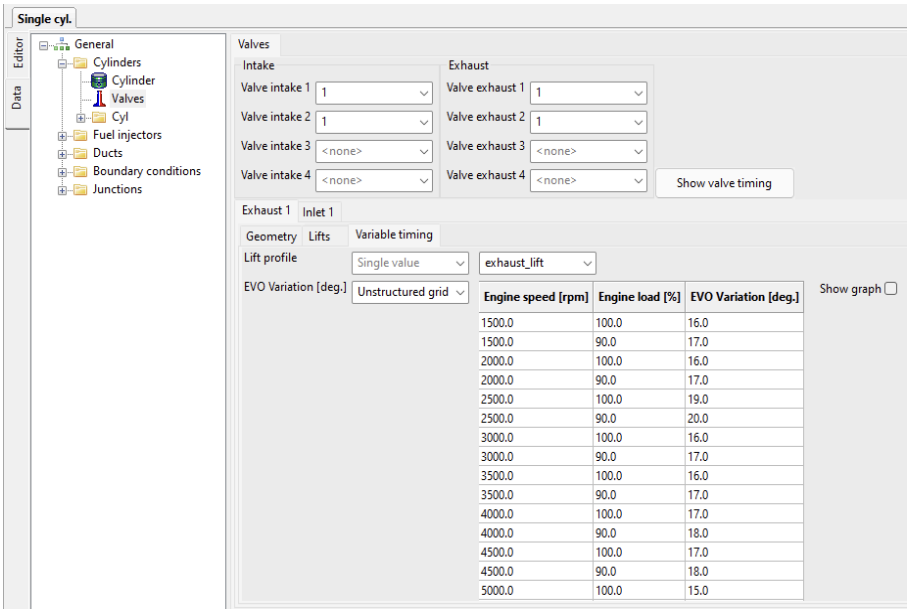

Figure 73 – Exhaust Valve Timing Variation Input Window in *Gasdyn*

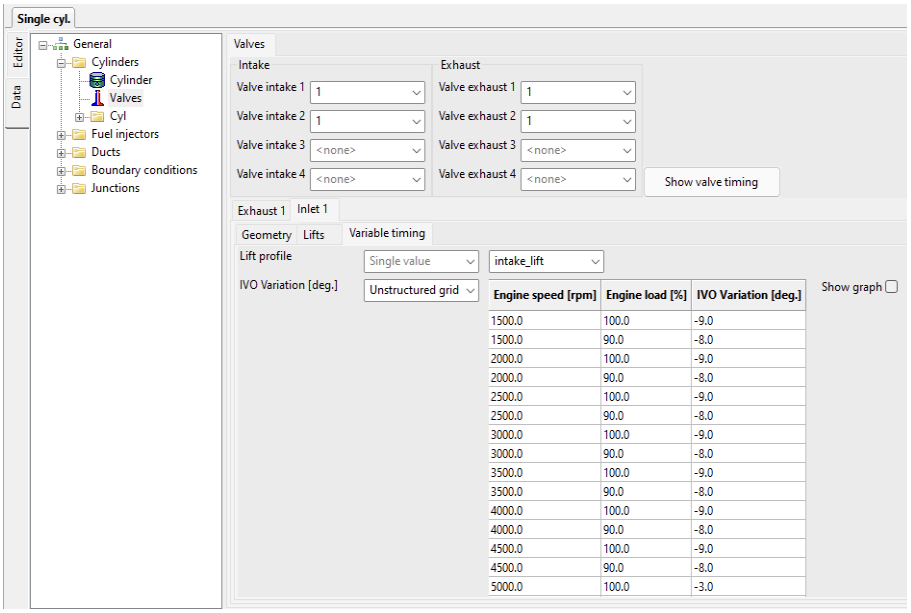

Figure 74 – Inlet Valve Timing Variation Input Window in *Gasdyn*

With the Variable Valve Timing (VVT) feature now enabled and inputted with the optimized valve timing values, the simulation was carried out at each of the selected RPM points. This was done with the optimal inlet duct length of 0.52 m, combined with the VVT settings in the *Gasdyn*. These simulations aimed to evaluate the combined effects of the optimized VVT and intake duct configuration on engine performance metrics, including brake torque, brake power, volumetric efficiency, and brake-specific fuel consumption (BSFC). The outcomes of these simulations are presented in *Table 9*.

RPM	Volumetric efficiency (%)	Break Torque (Nm)	Brake Power (kW)	BSFC (g/kWh)
1500	90.2	50.3	7.9	319.9
2000	84.1	47.4	9.9	297.6
2500	86.6	49.8	13.0	288.5
3000	100.6	60.0	18.9	287.1
3500	102.9	60.6	22.2	278.5
4000	95.7	55.8	23.4	272.1
4500	97.6	57.6	27.1	264.7
5000	93.5	53.5	28.0	271.1
5500	92.8	53.1	30.6	268.0
6000	91.0	51.2	32.1	272.6
6500	86.6	47.4	32.2	279.7
7000	88.8	47.5	34.8	286.0
7500	90.2	46.7	36.6	297.4
8000	87.0	43.7	36.6	305.5
8500	82.7	40.0	35.6	316.0

Table 9 - Engine's Performance Parameters Obtained from the *Gasdyn* Simulation Results at optimal duct length 0.52m and Variable Valve Timing (VVT) implemented

The simulation results obtained using the original inlet duct length of 0.26 meters, as presented in *Table 3*, were compared with those from the modified configuration featuring an optimized inlet duct length of 0.52 meters and implemented Variable Valve Timing (VVT), as detailed in *Table 9*, this comparative analysis aimed to evaluate the impact of the combined modifications on key engine performance metrics. The corresponding comparison plots for Brake Torque vs. RPM, Brake Power vs. RPM, Volumetric Efficiency vs. RPM, and Brake Specific Fuel Consumption (BSFC) vs. RPM are illustrated in *Figures 75, 76, 77,* and *78*, respectively.

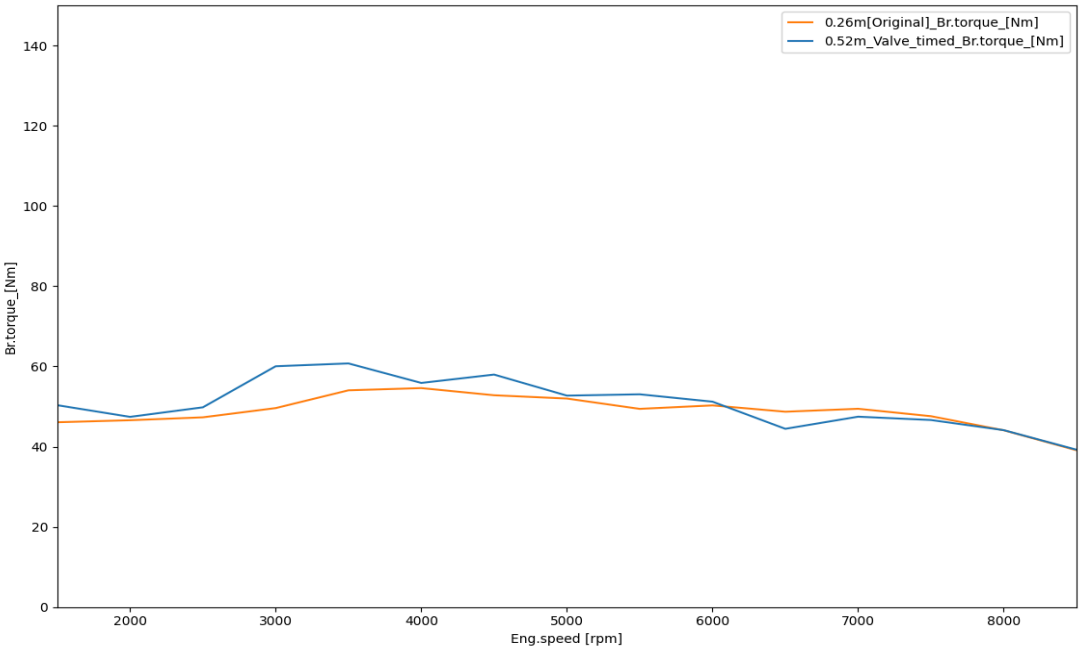

Figure 75 - Comparison of Brake Torque vs. RPM for 0.26 m and 0.52 m Inlet Duct Lengths with VVT

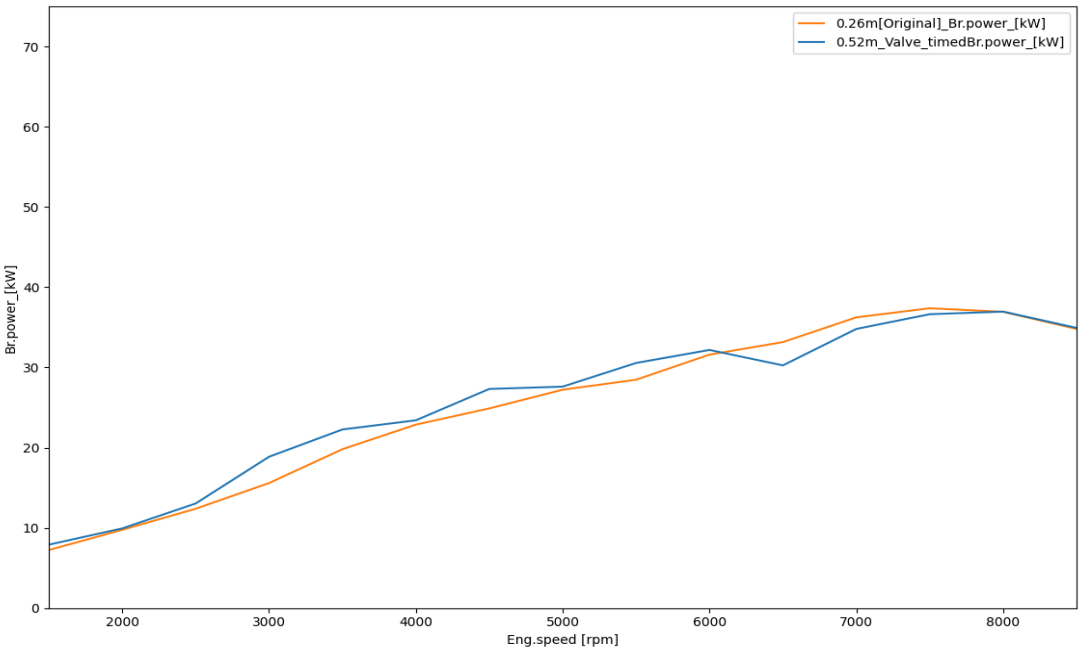

Figure 76 - Comparison of Brake Power vs. RPM for 0.26 m and 0.52 m Inlet Duct Lengths with VVT

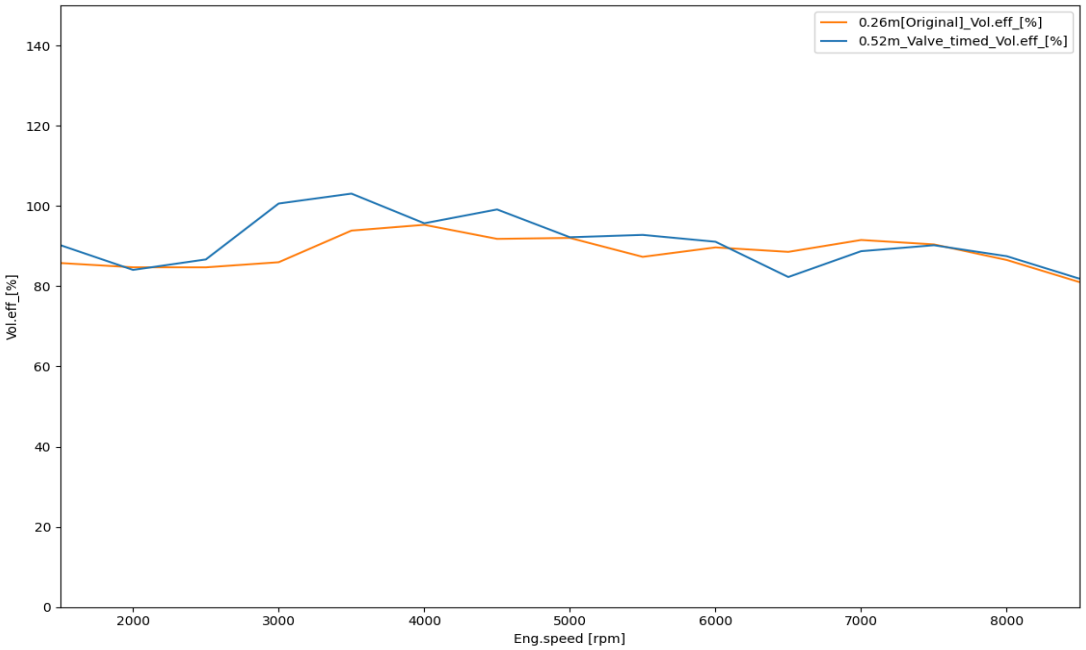

Figure 77 - Comparison of Volumetric efficiency vs. RPM for 0.26 m and 0.52 m Inlet Duct Lengths with VVT

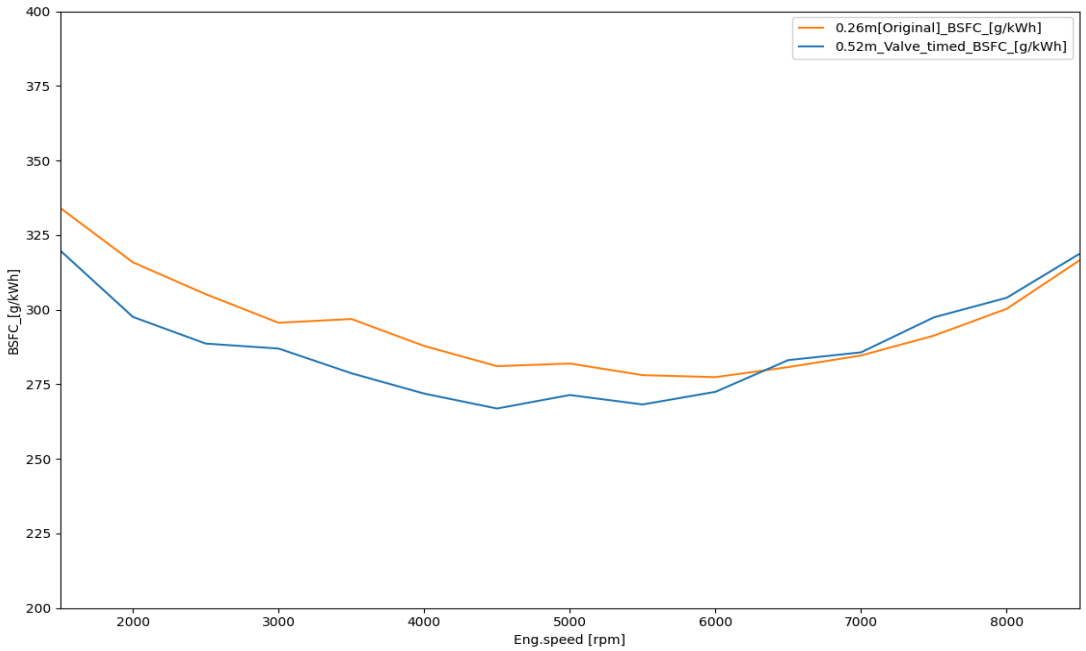

Figure 78 - Comparison of Brake Specific Fuel Consumption (BSFC) vs. RPM for 0.26 m and 0.52 m Inlet Duct Lengths with VVT

Based on the simulation results and analysis, the introduction of Variable Valve Timing (VVT) alongside the optimized 0.52 m inlet duct length led to noticeable improvements in engine performance across the full RPM range when compared to the original non-modified 0.26 m duct configuration. Previously, when using the 0.52 m duct length without VVT, the engine showed strong performance between 3500 and 4500 rpm, particularly in terms of volumetric efficiency, brake torque, and reduced BSFC. However, this setup experienced a gradual decline in performance at higher RPMs. Although still acceptable, the 0.52 m duct alone could not fully match or exceed the performance delivered by the non-modified configuration at the upper end of the speed range.

After implementing VVT, the engine not only retained its strong mid-range performance but also exhibited clear improvements at higher engine speeds (5500–8500 rpm). This enhancement is evident from the simulation data: volumetric efficiency and brake torque remain consistently higher, and BSFC values are either comparable or lower than the original setup, even at high RPMs. For example, at 7500 rpm, the VVT-optimized case records a BSFC of 297.4 g/kWh, compared to 291.3 g/kWh in the non-modified case, an acceptable trade-off considering the significant gain in brake power (36.6 kW vs. 37.4 kW). Similarly, at 8500 rpm, the VVT case shows 35.6 kW of brake power, outperforming the 34.8 kW of the original setup.

This demonstrates that VVT successfully mitigates the performance drop previously observed at high RPMs with the longer duct alone. The ability to dynamically adjust valve events allows better control of the air-fuel intake process, ensuring that the engine operates closer to optimal conditions throughout the entire speed range. However, it's important to note that while the optimizer provides a solid foundation for valve timing settings, engineering oversight is essential. Engineers must calibrate these settings to ensure smooth transitions between RPM ranges and to prevent potential mechanical issues associated with abrupt valve timing changes. This calibration process involves fine-tuning the valve timing parameters to align with the engine's specific operational requirements and constraints.

The combination of a 0.52 m optimized inlet duct and variable valve timing provides a well-balanced solution, offering both strong mid-range performance and improved high-speed efficiency.

Conclusion

In this thesis, we have developed and validated an optimization framework called *engine_optimizer*, which integrates with the *GASDYN* fluid dynamics simulator to enhance the performance tuning of internal combustion engines. This integration represents a significant advancement in the field of engine design, substantially reducing the time and costs associated with experimental bench testing while providing valuable insights into optimal engine configurations.

The first part of this work focused on detailing the optimization framework's architecture and capabilities. We explained the mathematical foundations of the Mesh Adaptive Direct Search (MADS) algorithm implemented in the optimizer and illustrated how it efficiently navigates complex, non-smooth optimization landscapes without requiring gradient information. The framework's ability to group duct subelements for both length and diameter optimization is particularly valuable, as it significantly reduces the number of variables and consequently the computational time required for finding optimal solutions. Additionally, the multiprocessing capabilities and XML input interface enhance user experience and efficiency when dealing with complex engine configurations. The optimizer supports several optimization modes, including duct length optimization, valve timing optimization, air-to-fuel ratio optimization, and combined optimization approaches. This versatility allows engineers to address various aspects of engine performance, whether individually or in combination, providing a comprehensive toolkit for engine designers. The implementation of proportional scaling for duct modifications ensures that geometric modifications maintain structural coherence, resulting in technically feasible solutions.

In the second part of this thesis, we validated the optimization framework by applying it to a real-world case: a single-cylinder motorcycle engine. Two key optimization studies were conducted: inlet duct length optimization and variable valve timing optimization. The optimizer identified an inlet duct length of 1.52 meters as optimal for maximizing volumetric

efficiency at 4000 rpm. However, physical constraints, such as limited space in a motorcycle, will restrict the practical implementation of such long duct lengths, limiting the maximum feasible duct length. By acknowledging these constraints, we refined our approach to optimize the inlet duct length to 0.52 meters in combination with variable valve timing, achieving a well-balanced solution that provided strong mid-range performance while also improving high-speed efficiency.

This balance would have been challenging to achieve through traditional experimental methods alone, highlighting the value of simulation-based optimization approaches. It is important to note that while the optimization framework provides powerful tools for identifying optimal configurations, engineering judgment remains crucial. As demonstrated in our case study, optimizing for a single operating point can lead to suboptimal performance at other points. This underscores the need for proper supervision, contextual understanding, and consideration of physical constraints when applying optimization techniques to engine design.

Future Work Perspectives

Looking ahead, there are several promising directions for further development of the *engine_optimizer* framework:

Multi-objective Optimization Capabilities: Extending the framework to support multi-objective optimization would allow engineers to simultaneously address competing design goals, such as maximizing power while minimizing emissions or fuel consumption. This would be particularly valuable in the context of increasingly stringent environmental regulations.

Integration with Machine Learning Techniques: Incorporating machine learning algorithms could enhance the optimizer's efficiency by developing surrogate models that reduce the need for computationally expensive simulations. This could significantly accelerate the optimization process, especially for complex engine configurations.

Expansion to Additional Engine Parameters: While the current framework addresses key parameters such as duct geometry and valve timing, future versions could incorporate additional variables such as combustion chamber design, compression ratio, and more sophisticated emissions control strategies.

The *engine_optimizer* framework developed in this thesis represents a significant step forward in simulation-based engine optimization. By reducing reliance on physical prototyping and experimental testing, it offers substantial benefits in terms of time and cost efficiency. As computational capabilities continue to advance, this approach to engine design and optimization will become increasingly valuable, enabling engineers to develop more efficient, powerful, and environmentally friendly internal combustion engines for the future.

Bibliography

[1] G. Ferrari, G. D'Errico, and A. Onorati. *Internal combustion engines. Società Editrice Esculapio*, 2022.

[2] A. M. Ramos, C. Antoniou, and S. Papathanasopoulou. *Implementation of MADS algorithm for engineering design optimization. Engineering Optimization*, 2015.

[3] C. Audet, S. Le Digabel, and C. Tribes. *The mesh adaptive direct search algorithm for granular and discrete variables. SIAM Journal on Optimization*, 2019.

[4] J. B. Heywood. *Internal Combustion Engine Fundamentals*. McGraw-Hill Education, 2018.

[5] C. Audet and J.E. Dennis Jr. *Analysis of generalized pattern searches. SIAM Journal on Optimization*, 2003.

[6] L. Teodori, K. Takahashi, H. Endo, and K. Kojima. *Development of a one-dimensional simulation model for variable valve timing engines. SAE Technical Paper*, 2018.

[7] C. Audet and J.E. Dennis JR. *Mesh adaptive direct search algorithms for constrained optimization. SIAM Journal on Optimization*, 2006.

[8] A. Marinoni, P. Gualtieri, and G. D'Errico. *Application of a 1D fluid dynamic code for the analysis of atypical valve lift laws in internal combustion engines. Energy Procedia*, 2014.

[9] V. Knop and L. Malbec. *CFD simulation and experimental validation of a gasoline engine with variable valve timing. SAE International Journal of Engine*s, 2022.

[10] M. Jemni, G. Kantchev, and M. Abid. *Influence of intake manifold design on in-cylinder flow and engine performances in a bus diesel engine converted to LPG gas fuelled. Energy Conversion and Management*, 2011.

[11] P. Cheng, Y. Wang, and S. Leslie. *Simulation and optimization of variable valve timing for internal combustion engines. Applied Energy*, 2018.

[12] A. Marinoni, F. Paltrinieri, and G. D'Errico. *Development of a one-dimensional fluid-dynamic model of a motorcycle engine for performance prediction. Energy Procedia*, 2018.

[13] R. S. Benson. *The thermodynamics and gas dynamics of internal-combustion engines*. Oxford University Press, 1982.

[14] A. Onorati, G. Ferrari, and G. Montenegro. *The prediction of 1D unsteady flows in the exhaust system of a S.I. engine including chemical reactions in the gas and solid phase*. SAE Technical Paper, 2008.

[15] G. P. Blair. *Design and simulation of four-stroke engines. SAE International*, 1999.

[16] H. C. Akdag, O. F. Percin, and A. Dinler. *Optimization of engine performance by using design of experiments and Latin hypercube sampling methods. Journal of Engineering Research*, 2021.

[17] F. Mariani and C. Poggiani. *Advances in optimization algorithms for tuning of internal combustion engines. Mechanical Systems and Signal Processing*, 2019.

[18] M. J. D. Powell. *Direct search algorithms for optimization calculations. Acta Numerica*, 1998.

[19] T. G. Kolda, R. M. Lewis, and V. Torczon. *Optimization by direct search: New perspectives on some classical and modern methods. SIAM Review*, 2003.