Swansea University
Prifysgol Abertawe

Development of Design Optimisation Techniques with Application to the Transonic Performance of the Skylon Spaceplane

Ben Hickling Smith

Swansea University

Submitted to Swansea University in fulfilment of the requirements for the degree of Doctor of Philosophy

2024

Declaration

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed (candidate): [redacted]

Date: 17/01/2025

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed (candidate): [redacted]

Date: 17/01/2025

I hereby give consent for my thesis, if accepted, to be available for electronic sharing.

Signed (candidate): [redacted]

Date: 17/01/2025

The University's ethical procedures have been followed and, where appropriate, that ethical approval has been granted.

Signed (candidate): [redacted]

Date: 17/01/2025

This thesis is dedicated to my father.

Acknowledgement

I would like to thank my supervisors Prof Ben Evans and Dr Sean Walton for their unwavering support and confidence in me during my time at Swansea. They were everything I could have asked for as academic supervisors, and more.

I also wish to acknowledge Reaction Engines Ltd for their support and contribution to funding. In particular, I would like to thank, Martin Dodds and Dr Neil Taylor for their contributions.

I express my thanks to Dr Alma Rahat for his time, support and guidance, every conversation with him made me a better engineer and academic. All staff from the faculty of science and engineering hold themselves to the highest standard and I am grateful to have been part of such a hard working, intelligent and welcoming community. My PhD experience would not have been the same without *The office where procrastination leads to productivity.* They know who they are and they helped make my time in the office and at conferences more enjoyable than I could have hoped. Also, the individuals that made Swansea such a memorable experience can not go unspoken; Danny, Richard, Jakub, Jo, Hannah and Becky to name but a few.

I would finally like to thank my family. My uncle and grandma for all their support during my undergraduate studies. Most notably of all, I thank my mum. For everything.

Abstract

The aim of this thesis was the explore design optimisation techniques that could be used in the optimisation process of the Skylon spaceplane. The research began by investigating the performance of the existing design at a critical design point, the results of which were presented to industry partners. Parameterisation techniques were then explored and a novel mesh morphing framework was developed. The method uses Radial Basis Functions to ensure that a mesh remains suitable for computational fluid dynamic analysis following the application of a morph function to a subset of the geometry. This framework was used to explore optimised designs with the aim of reducing the drag coefficient of the geometry, which was achieved with a reduction in drag coefficient of up to 2% for the Skylon spaceplane at Mach 1.2.

Contents

List of Tables

List of Figures

List of Abbreviations

ARD Automatic Relevance Determination. 107

BBO Black-Box Optimisation. 98

BO Bayesian Optimisation. vi, x, xi, 98, 99, 108–112, 114, 116, 117

CAD Computer Aided Design. 2, 4, 5, 12, 13, 50, 60, 61, 129

CFD Computational Fluid Dynamics. vi, vii, xii, 2, 7, 10–13, 16, 17, 21–24, 51, 71, 85, 95, 96, 98, 110, 114, 119, 124, 126, 127, 129, 133, 136, 140, 141

CMA-ES Covariance Matrix Adaption Evolution Strategy. 109–111, 113

CPU Central Processing Unit. 17

CRM Common Research Model. viii, 51, 52

DOE Design of Experiments. vi, 84, 85, 89, 97, 111, 112, 115, 117

EA Evolutionary Algorithms. vi, 124, 125, 141

EI Expected Improvement. xi, 108, 109, 111, 116

FDGD Fast Dynamic Grid Deformation. 4

FFD Free-Form Deformation. 4

FVM Finite Volume Method. 12, 17

GP Gaussian Process. vi, 99, 101, 105, 108–111, 113, 114

HLLC Harten-Lax-van Leer-Contact. 12

HOTOL Horizontal Take Off and Landing. 5

HPC High Performance Cluster. 17, 111

ICAO International Civil Aviation Organization. 9

KDE Kernel Density Estimation. x, 87, 89, 95

LCB Lower Confidence Bound. 109

LEO Low Earth Orbit. 5

LHS Latin Hypercube Sampling. ix, x, 85, 86, 109, 111, 115

LOOCV Leave-One-Out-Cross-Validation. 41

MQ Multiquadric. 34

NURBS Non-Uniform Rational B-Splines. 12, 13

PDEs Partial Differential Equations. 11

PoI Probability of Improvement. xi, 108, 116

PSO Particle Swarm Optimisation. xii, 124–126, 137, 140–142

RBF Radial Basis Function. v, vi, ix, x, xii, 4, 34–37, 39–42, 44, 48, 49, 60, 61, 65, 66, 68–71, 83, 85–87, 89, 90, 95, 96, 106, 110, 143, 149, 150

RBF's Radial Basis Functions. 34, 35, 75

REL Reaction Engines Ltd. vii, 2, 5–7, 9, 11, 54

RMSE Root Mean Square Error. viii, xi, 42, 43, 129–131

SA Spalart-Allmaras. 12, 21

SSTO Single-Stage To Orbit. 5, 6

t-SNE t-Distributed Stochastic Neighbour Embedding. x, 87–89, 91

TPS Thin Plate Spline. 34

UCB Upper Confidence Bound. 109

Nomenclature

α	Angle of attack.
$\boldsymbol{\beta}$	RBF Polynomial weights.
$\boldsymbol{\lambda}$	RBF weights.
$\boldsymbol{\mu}$	Mean of a distribution.
$\boldsymbol{\Sigma}$	Variance of a distribution.
$\boldsymbol{A}$	Interpolation matrix.
$\boldsymbol{q}$	Heat Flux
$\boldsymbol{u}$	Velocity
$\boldsymbol{x}_c$	RBF Centers.
$\boldsymbol{Y}$	Unsure
δ	Small change in a quantity.
ϵ	Global shape parameter.
η	Local support radius.
$\hat{\phi}_n$	Design parameters for Sphere.
$\mathbb{N}$	Natural number set
$\mathbb{R}$	Real number set
$\mathcal{GP}$	Gaussian process.
$\mathcal{N}$	Normal distribution.
μ	Dynamic viscosity coefficient
μ_m	Mach angle
ν	Kinematic viscosity coefficient
Φ	Radial function.
ϕ	No Subscript - Univariate basis function.
ϕ_n	Design parameters for Skylon.

ρ	Density
τ	Deviatoric stress tensor
θ	Kernel hyperparameters.
$\widetilde{\phi}_n$	Design parameters for DPW.
C	Constrained nodes within a mesh morph.
C_d	Drag coefficient.
C_f	Lateral force coefficient.
c_f	Skin friction coefficient.
C_l	Lift coefficient.
d	Distance metric for RBF.
E	energy
f	Real valued function to approximate.
g	Acceleration due to gravity
h	Additional polynomial.
J	Cost function
M	Mach
q	Dynamic pressure.
r	Distance between two points calculated by metric d.
S	Cross sectional area at a point
s	Interpolation function, approximates f.
T	Target nodes within a mesh morph.
t	Time
U	Unconstrained (free) nodes within a mesh morph.
y^+	Non-dimensional wall distance.
y_1	Height of first layer in boundary layer.
y_{max}	Height of final layer in boundary layer.

Chapter 1

Introduction

1.1 Motivation & Background

Spaceplanes are aerospace vehicles engineered to operate in both the Earth's atmosphere and in the vacuum of space. This dual capability necessitates a hybrid design, influenced by the aerodynamic properties of aircraft and the orbital capabilities of spacecraft. The inherent complexity of spaceplane design emerges from the integration of multiple disciplines, each enforcing strict design requirements on the vehicle. The design must perform well across subsonic, transonic, and supersonic flight conditions when transitioning from ground level to the outer atmosphere.

The main focus of this research is to explore methods for the design optimisation of spaceplanes, with a particular focus on their transition through the transonic regime. This regime is particularly challenging due to the sharp increase in drag experienced as a result of the formation of shock waves over the vehicle. The current industry standard for modelling flow within this domain, Computational Fluid Dynamics (CFD), is used throughout this thesis to analyse the performance of spaceplane designs through the transonic regime. Optimisation techniques that can be used in conjunction with CFD are also discussed. The trade-off between performance increase and computational time/cost is also discussed from the perspective of the methods application to industry cases.

This research was conducted in close partnership with industry sponsors, Reaction Engines Ltd (REL), a British aerospace company REL, based in Oxfordshire, England. They are the main sponsor behind this research and collaborated on the project throughout it's duration. REL work to create novel solutions to some of the most challenging problems that currently prevent the effective use of spaceplanes in many real-world scenarios. The culmination of their work is the development of Skylon, an ongoing spaceplane concept design under development.

The expected output of this research collaboration with REL was a tool capable of optimising future spaceplane designs. The tools should also however be suitable for use across the full range of their business applications. It should therefore provide components that are agnostic from the design process for a spaceplane. The tool should also be capable of being used with existing Computer Aided Design (CAD) geometry's and meshes defined using different file specifications.

The remainder of this chapter provides a short analysis of the different types of design optimisation methods available, highlighting those that will be explored in depth within this thesis and introduces the primary industry case study used throughout this thesis,.

1.2 Design Optimisation

Design optimization is the refinement of a design or system to improve desired performance goals. It can involve changing various aspects of a design such as geometry, structure, or sizing, and seeks to strike a delicate balance between, often conflicting, objectives and constraints. Methods for the design optimisation of geometric bodies can be categorised into two key domains; topology optimisation and shape optimisation. As well as selecting an appropriate optimisation method, a suitable parameterisation technique should be applied to construct the valid design space for a desired optimisation procedure.

1.2.1 Topology Optimisation

Topology optimisation is the optimisation of the distribution of material within a specific domain space [84, 12]. This process often results in very innovative unconventional shapes emerging. It is an effective method for internal problems, such as identifying the optimal shape of components under structural analysis [61].

The method has been extended to fluid flow problems, initially applied to Stokes flow [15]. A significant amount of literature has followed this, all of which was summarised, up until the year 2020, by Alexandersen et al [4]. The majority of flow conditions explored are for very low Reynolds numbers. Some work on topology optimisation for compressible flows, applying topology optimisation to supersonic flows [38, 75, 57]. The compressible flow solvers used during these optimisation procedures however are inviscid, and so still not suitable for transonic flow cases. This is a key design criterion for the spaceplane optimisation explored throughout this work. The challenges faced using topology optimisation within this regime suggest that topology optimisation is not a suitable tool to explore for this application.

1.2.2 Shape Optimisation

Shape optimisation focuses on modifying the surface domain of the geometry or shape with a given topology, rather than the entire domain of the design space [100]. It provides a means to fine-tune structures, and the cost function of a geometry is independent of the evaluation method. This allows any evaluation method to be used to determine the performance of a geometry. For example, fully compressible and viscous flow can be solved over the domain, and the aerodynamic properties of this solution can be calculated. Application of shape optimisation to external fluid optimisation problems is far more readily available in literature than topology optimisation [68].

Shape optimisation is the minimisation of a cost function, J, defined by multiple objectives, driven by a set of design parameters $x \subseteq \mathbb{R}^n$, subject to any number of constraints, g_i. This formulation is given in Equation 1.1. The constraints construct the set of admissible designs within the design space, $\Omega = \{g_i(x) \leq 0 \text{ for } i = 1, ..., n : x \in \mathbb{R}^n\}$.

$$min\{J(x) : x \in \Omega\} \tag{1.1}$$

1.2.2.1 Parameterisation

While topology optimisation can be largely subdivided into density based methods and level set methods, there are many different methodologies for shape optimisation that are highly dependent on the parameterisation scheme used on the geometry. An in depth review into the different options is given by Skinner & Behtash [93]. In this section, some of these methods are discussed including some of their benefits and drawbacks for aerodynamic design optimisation of spaceplanes in the transonic regime.

Parameterised CAD based methods can be effective in some optimisation procedures. However, in multidisciplinary applications, multiple CAD representations may be required [106]. Translation between CAD representations can generate translation errors that result in a representation unsuitable for mesh generation [77]

Spline based methods can also be used. Splines are constructed from a set of control nodes, which form the degrees of freedom of the system [78]. Moving the control nodes effects the shape of the surface it is used to define [84]. This can lead to complex shapes, and does not require significantly large numbers of degrees of freedom. However, local control can sometimes prove challenging [93]. It is often difficult to construct an entire complex geometry using spline based methods [84]. In a different concept to spline based methods, but still reliant on construction of analytical functions, Hicks and Henne introduced a method that used weighted sums of shape functions to construct an airfoil analytically [50]. This method has been proven to be highly effective for airfoil design optimisation but does not transfer over to complex geometries well.

The methods discussed thus far have been constructive methods. The parameters of analytical functions form the design variables, and the geometry is constructed from their definition. For existing geometries, another class of parameterisation methods are volume based methods [93].

Volume based methods define a volume around the geometry for which a deformation to the enclosed geometry is applied to. Free-Form Deformation (FFD) is a popular example of this type of volume based method [90]. A more generalised definition of this method are domain element approaches. In this approach, groups of nodes are enclosed within a domain, and as the domain moves, the parametric coordinates within the domain remain constant while the global coordinates change [84]. This method can be applied to both continuous CAD geometries and discrete meshes. Methods such as Fast Dynamic Grid Deformation (FDGD) use this approach to propagate mesh deformation throughout a mesh [63].

A third class of parameterisation methods are deformative methods. Similar to the volume based methods, these methods are applied to existing geometries and modify the existing design.

Mesh morphing is a key example of a deformative method that modifies nodes of an existing mesh to create a new geometry. Any subset of nodes can be treated as design variables, providing a high degree of flexibility. However, care has to be taken that not too many nodes are chosen as design points. The coupling between the large number of variables in this case can cause the design space to become too complex to explore. When using deformative approaches, it can also be difficult to maintain smoothness [84]. Interpolation schemes such as Radial Basis Function (RBF) can be used to smoothly propagate translations into the surrounding mesh, maintaining smoothness [13]. There are many other methods for propagating the motion of control nodes throughout the boundary [71].

1.2.3 Optimisation Methods

When seeking to minimise a problem, it is not always possible to exhaustively search the design space. For example, the parameters may be represented by continuous real values and the result may be highly sensitive to changes in the inputs. Optimisation methods can be used to explore a limited subset of the domain that use known or calculated information about the space to search the most relevant designs. They can be categorised using many definitions, and some of the main categories are described here.

Optimisation methods are usually mathematically well defined, however this is not always the case. Metaheuristics are high level problem solving strategies that can provide approximate near-optimal solutions.

All methods can be divided into two categories, deterministic and stochastic [20]. Deterministic methods always return the same result for a given set of initial conditions and parameters, for example, calculating the gradient of a function and advancing toward an optimal based on the derivative. Stochastic methods on the other hand contain an element of randomisation in their calculation. Repetitions of the same test may yield different results.

Two other important definitions for algorithm classification are global search methods and local search methods. Functions can contain global and local optima. A global optimum over a domain is defined as the point where no other point within the domain performs better than it. A local optimum within a domain can be defined as any point where there exists a ball around that point for which it no point out performs the local minima within that localised region. Some search algorithms are classified as local search algorithms if they are not able to explore new regions once a local optima is located. For example, consider a gradient-based method that has located a local optima, it would not deem the gradient in any direction to be favourable and so would not explore the function beyond its current location.

Some examples of different optimisation methods are introduced throughout this thesis and details of their classification will be provided where appropriate.

1.2.4 Summary

Given the target use case for this thesis, optimising performance within the transonic regime, shape optimisation methods would be the most appropriate method to use to generate the design space. The geometry parameterisation chosen should enable the widest range of use cases. Considering the challenges that can be faced when converting from different CAD definitions to different mesh specifications, a parameterisation that uses just the node and element connectivity is chosen as this information is readily available and the bare minimum information required for any computational analysis of a design. The design space will therefore be generated using mesh modification, controlling the node locations.

1.3 Skylon

The optimisation methods explored throughout this research are applied to an industry test case, the Skylon spaceplane. The project dates back to the early 1980's and is the evolution of a prior British led project, Horizontal Take Off and Landing (HOTOL). The goal of both projects was to develop a Single-Stage To Orbit (SSTO) spaceplane capable of reaching Low Earth Orbit (LEO). Skylon was developed with the objective of improving the technically feasible HOTOL design to a level that could attract more international interest of support [65]. To achieve the feat of overcoming the shortfall in available mass ration, a number of technological breakthroughs had to be made [108]. The engine pre-cooler in particular required significant research and development. The engineering challenge was to normalise the air flow into the engine across a wide Mach range (approximately 0-5), which required slowing and cooling high speed airflow. At the time Skylon was conceptualised this kind of technology did not exist however a design has now been developed and tested by REL.

As a commercially sensitive project, much of the technical information about technological developments is not made public. However, there exists some literature on the development of the project. Progress updates have been made available sporadically since 2004 [107, 108, 46, 65]. Business analysis and projected business models have been developed to help draw attention to the potential impact of the project [49, 48, 47]. There has also been much work on analysing the hypersonic performance of the spaceplane, both numerical [67, 26] and, more recently, empirical [52].

Between public and private information regarding the design, much has been done to determine the performance within subsonic, supersonic and hypersonic regimes. To date, transonic performance has not been characterised in the same level of detail. This is therefore the main area of interest focused on in this research. An initial analysis of the existing performance was performed and the baseline design was then used as a case study for design optimisation using a range of methods.

As the vehicle is being designed as a world first SSTO spacecraft. This presents a unique challenge to the engineering design of the vehicle as it must be optimised across all aerodynamic regimes. The design must generate enough lift at take off to get off the ground, must be designed to exhibit minimal drag through the transonic regime and be able to withstand the extreme heat and structural loading experienced through the hypersonic regime. The existing design has been theoretically shown to be suitable to meet all these demanding requirements. It is believed that further improvement can still be made to the existing design.

The baseline design for Skylon has been developed over the past 40 years and remains largely unchanged from the original. The exact configuration we use as the baseline design in multiple test cases throughout this work is the C1 configuration. It is postulated that the transonic regime is where the most benefit can be gained from optimisation of the design and so this is the region we focus attention on.

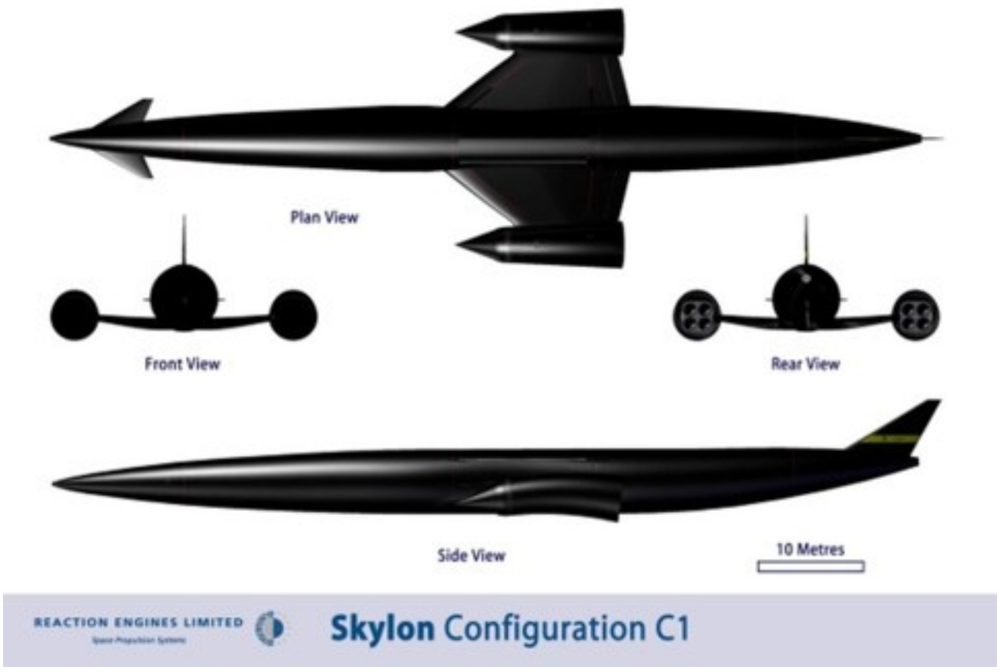


Figure 1.1: Baseline configuration of Skylon Spaceplane.

1.4 Industry Case Study

The Skylon spaceplane has undergone multiple design iterations and at each stage, optimised for the range of different environments it would have to endure if developed. In particular, the transonic regime was an area where REL suggested there may still be optimisation to be gained. This section describes the case study constructed to target this including the constraints to be considered.

1.4.1 Objective Function

The objective function for this case study is to minimise the drag coefficient of the Skylon spaceplane throughout the transonic regime. Based on the existing preliminary design work already carried out, REL identified that the area of the Skylon design they expected to achieve the most performance gain was within the transonic regime. Vehicles are described to be within the transonic regime when local pockets of both sub and super sonic flow exist at the same instance around the vehicle. The Mach number that this occurs at is described as the vehicles critical Mach value, M_{crit}. Soon after this point, the vehicle experiences a significant rise in C_d, at which point the vehicle is said to have reached is Mach drag divergence number, M_{dd}. It is within this regime that spaceplanes, and indeed any vehicle, experience their highest magnitude of drag coefficient [6].

For the Skylon spaceplane, Fig. 1.2 shows the approximate relationship between the thrust and drag against Mach number of the aircraft. The drag coefficient values are the result of CFD analysis. Free stream conditions for this are interpolated from Mach vs Altitude data provided by REL, discussed in more detail later in this chapter. Due to the ongoing development of the engine for the spaceplane, exact values for thrust are not available, and so an approximate assumption of the engines behaviour as described by REL is shown.

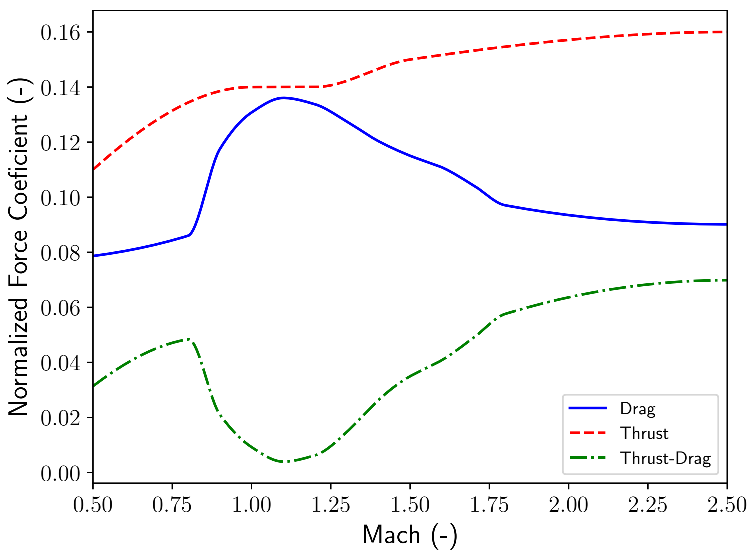


Figure 1.2: Thrust, drag and thrust-drag vs Mach number across the transonic regime. Drag is calculated using CFD, thrust is approximated based on expected behaviour of the engines.

Fig. 1.2 highlights that the minimum difference between thrust and drag lies between $M = 1.0$ and $M = 1.2$. Reducing the vehicle drag at this point means less thrust is required to maintain the required acceleration to follow the mission plan. The engine size is directly proportional to the magnitude of the thrust curve, therefore a reduction in required thrust equates to a reduction in engine size, and therefore mass. Assuming the lift remains fixed, and for the same fuel volume, this permits an increase in available payload mass, a leading business requirement of the vehicle.

1.4.2 Optimisation Constraints

Two assumptions made in the objective function are that the vehicles lift and fuel volume remain fixed.

Lift Constraint

The lift constraint is a strict equality constraint due to the fact that excess lift would result in a change of the overall mission profile of the spaceplane. While this is something that could be considered in future studies, the aim of this optimisation is to improve the performance of the current mission profile. For the purposes of this study, the baseline angle of attack across the full Mach range is assumed to be $\alpha = 4.0$, and the lift constraint is ensured by modifying the incidence of the modified designs at each Mach value.

Volume Constraint

The volume constraint can be handled by introducing a constraining parameter into the parameterisation. The value of this dependent parameter can be calculated after all other independent parameterisations have been applied to ensure the volume equality constraint is met. In the preliminary designs for the Skylon spaceplane, the fuel is stored in the fuselage of the vehicle. Therefore, to meet this constraint, the total volume of the fuselage is constrained.

1.4.3 Optimisation Formulation

For a given set of shape parameters represented by $\boldsymbol{x} = [x_1, ..., x_n]$ The objective function and constraints for the optimisation problem described in Section 1.4.1 1.4.2 can be formally described as

$$J(\boldsymbol{x}) = \min_{\boldsymbol{x}} C_d(\boldsymbol{x})$$

Subject to:

$M = 1.2$ (Mach number constraint)

$V(\mathbf{x}) \equiv V_0$ (Volume of the geometry remains the same as the volume of the original body.)

$C_l(\mathbf{x}) \equiv C_{l_0}$ (C_l of the geometry remains the same as that of the original body.)

$\mathbf{x} \in \mathcal{D}$ (Design space constraints).

Where:

- $C_l(\mathbf{x})$: Lift coefficient, as a function of the design variables $\mathbf{x}$.
- $C_d(\mathbf{x})$: Drag coefficient, as a function of the design variables $\mathbf{x}$.
- $\mathbf{x}$: Set of design variables (e.g., shape parameters, flow control variables).

- $\mathcal{D}$: Design space, defined as:

$$\mathcal{D} = \{\mathbf{x} \mid x_i^{\min} \leq x_i \leq x_i^{\max} \, \forall i\}.$$

1.4.4 Flight Conditions

Flight conditions across the domain were calculated based on interpolated flight path data consisting of the expected altitude at certain Mach values. The dataset consisted of a combination of flight path data provided by REL, with Mach values approximately within the range $M \in (0.8, 1.5)$ shown in Table 1.1, and flight path data provided by Mehta et al., for Mach values within the range $M \in [2.625, 16.969]$ [67].

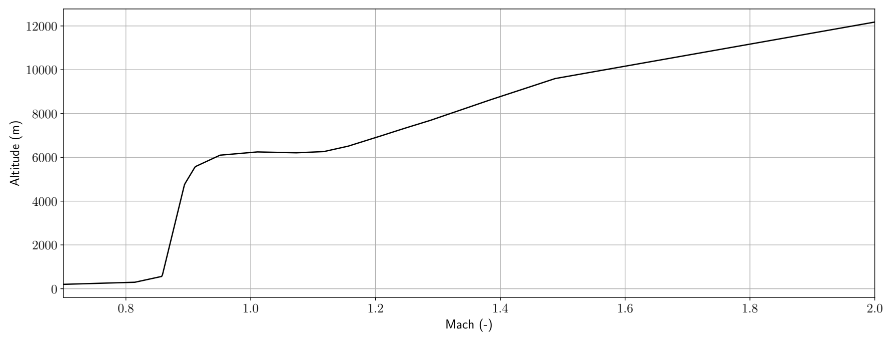


Figure 1.3: Mach vs altitude profile for Skylon. Data interpolated from the combined data set from Mehta 2015 and data provided by REL [67].

The flight path data is summarised in Fig. 1.3. The interpolated altitude from this dataset was used in parallel with the International Civil Aviation Organization (ICAO) standard atmosphere to calculate dynamic pressure and Reynolds number quantities for Skylon at each required Mach value [74].

1.5 Thesis Objectives

The primary aim of this thesis is to explore design optimisation techniques that can be applied to different engineering problems, such as those explored by the thesis sponsor REL. To achieve this aim, the following specific objectives will be met:

1. Provide indication of baseline performance for current Skylon spaceplane.
2. Develop a novel design optimisation methodology.
3. Investigate and evaluate different configurations of the proposed method.
4. Demonstrate methodology to a proof-of-concept test case provided by the industry sponsor.

These objectives collectively address the research problem by developing, validating and demonstrating a tool that can be used to achieve the primary aim.

Table 1.1: Free stream data throughout the transonic regime for Skylon over a potential expected trajectory to low earth orbit. The reference length used for the Reynolds number was 1. All other parameters are calculated using standard atmospheric conditions at the given altitudes. Altitude and Reynolds numbers are given to 0dp.

Mach Number (-)	Altitude (m)	Reynolds Number (-)
0.8	280	18170628
0.9	5050	12980907
1.0	6221	12799839
1.1	6241	14052096
1.2	6899	14312289
1.3	7807	14078645
1.4	8775	13643862
1.5	9658	13247700
1.6	10162	13344593
1.7	10666	13379114
1.8	11169	13276639

1.6 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 introduces the computational methods used throughout this thesis, and includes results of the preliminary study of the baseline design. Chapter 3 summarises the implementation of a mesh morphing framework developed to enable modification of geometries for design optimisation. Chapter 4 presents the results from a design of experiments optimisation study using the mesh morphing framework to minimise the drag on the Skylon spaceplane. Chapter 5 presents the results of the same design optimisation case, but uses a Bayesian optimiser to explore the design space. The configuration of the optimiser is explored in detail, highlighting how it can be applied effectively for similar computationally expensive optimisation problems. Chapter 6 also explores the design optimisation of Skylon, however uses a computationally cheaper method than CFD to evaluate the performance of each geometry. The final chapter, Chapter 7, summarises the conclusions of the research.

Chapter 2

Computational Methods and Baseline Performance

The main focus of this research was to develop a tool capable of optimising future spaceplane designs. Reaction Engines Ltd (REL) provided the geometry for a spaceplane concept currently under development, the Skylon spaceplane introduced in 1.3. The initial objective was to provide a baseline analysis of the design using an in-house Computational Fluid Dynamics (CFD) tool, FLITE3D, to validate the performance they had so far predicted. When a design optimisation framework had then been developed, the objective was to use it to optimise the performance of the design.

In this chapter, the FLITE3D solver is introduced, and a mesh convergence study is performed using this solver on the baseline geometry to identify suitable mesh resolution's for validation and in an optimisation study. The aerodynamic performance of the geometry using this baseline mesh is then discussed.

2.1 Computational Fluid Dynamics

The performance of spaceplane design's in the transonic regime must be analysed. CFD remains one of the most fundamental methods of solving airflow over a geometry and is used extensively to determine aerodynamic properties and coefficients of geometries. The continuous form of the governing Partial Differential Equations (PDEs) to be solved are the Navier-Stokes equations, a set of conservation equations. The equations are given in their derivative form in Equations 2.1, 2.2 and 2.3, under the assumptions of viscous compressible flow.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0 \tag{2.1}$$

$$\rho \frac{D\boldsymbol{u}}{Dt} = -\nabla p + \nabla \cdot \tau + \boldsymbol{F} \tag{2.2}$$

$$\frac{\partial(\rho E)}{\partial t} + \nabla \cdot (\rho E \boldsymbol{u}) = -\nabla \cdot (p\boldsymbol{u}) + \nabla \cdot (\boldsymbol{\tau} \cdot \boldsymbol{u}) - \nabla \cdot \boldsymbol{q} \tag{2.3}$$

Equation 2.1 represents the conservation of mass, where ρ is the density, and $\boldsymbol{u}$ is the velocity vector. Equation 2.2 is the Cauchy momentum conservation equation which includes the pressure p, deviatoric stress tensor $\boldsymbol{\tau}$ and the body forces acting on the fluid $\boldsymbol{F}$. Equation 2.3 represents the conservation of energy which introduces the total energy ρE, and heat flux $\boldsymbol{q}$.

The body forces most commonly takes the form $\boldsymbol{F} = \rho \boldsymbol{g}$, where $\boldsymbol{g}$ is the vector force pertaining to gravity. This term however has a negligible contribution to the overall system in high speed flows, as can be demonstrated during non dimensionalisation, and so is often dropped.

The components of the deviatoric stress tensor $\boldsymbol{\tau}$ are given by

$$\tau_{ij} = -\frac{2}{3}\mu \frac{\partial u_k}{\partial x_k}\delta_{ij} + \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$$

where δ is the Kronecker delta, and μ is the dynamic viscosity coefficient, which varies with respect to temperature and is often modelled using Sutherland's law.

Three assumptions are often made about the fluids to simplify them. Namely, the fluids compressibility, the fluids viscosity, and the time dependence of the system. If the solution is assumed to be steady state, then time derivatives disappear. Under the assumption the solution is incompressible then the density is constant. If the fluid is assumed to have no viscosity it is referred to as inviscid, and the equations can be simplified significantly. One of the challenges faced when modelling transonic flows is that these simplifications are inappropriate. The boundary layer plays a significant role in the transonic regime, of which the viscous forces are required to accurately model this. The significant changes in pressure and density around the shock waves that form within this regime mean the flow cannot be considered incompressible. As a result, any solvers used must consider all terms of the equations.

Equations 2.1, 2.2 and 2.3 represent the continuous form of the Navier-Stokes equations; however computational methods such as Finite Volume Method (FVM) operate on their discretized form. The FVM solver FLITE3D therefore solves the discretized form of the Navier-Stokes equations [45]. In this approach, the governing equations for mass and momentum conservation are integrated over discrete control volumes within the computational domain. Numerical techniques are then employed to approximate the flow variables and iteratively solve for pressure and velocity fields. Specifically, FLITE3D uses the Spalart-Allmaras (SA) turbulence model [102] and the Harten-Lax-van Leer-Contact (HLLC) scheme to compute fluxes across cell interfaces [105].

An example of the standard CFD process is outlined in Fig. 2.1. A continuous representation of the geometry is created in the form of a Computer Aided Design (CAD) model, which uses combinations of Non-Uniform Rational B-Splines (NURBS) to define the geometries surfaces. NURBS are constructed using polynomial equations defined at control points, enabling the generation of smooth continuous manifolds through the points. These equations can be manipulated by changing the number and location of control points, modifying the weights of each control point and the order of the polynomials, to construct highly complex surfaces. While these models provide a highly accurate description of the geometries surface, they are not suitable representations for use within most CFD solvers. For this, a discrete representation of the geometry is required, called a mesh.

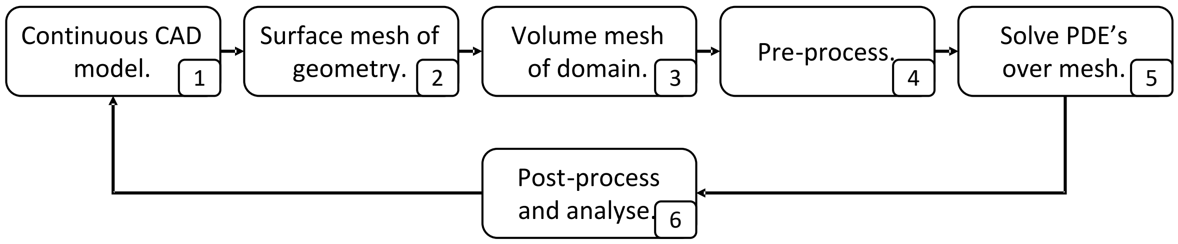
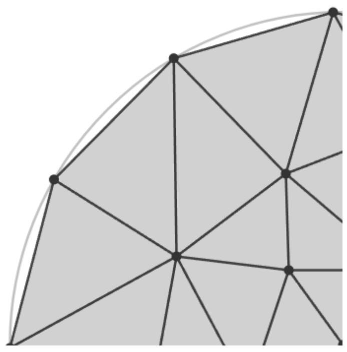


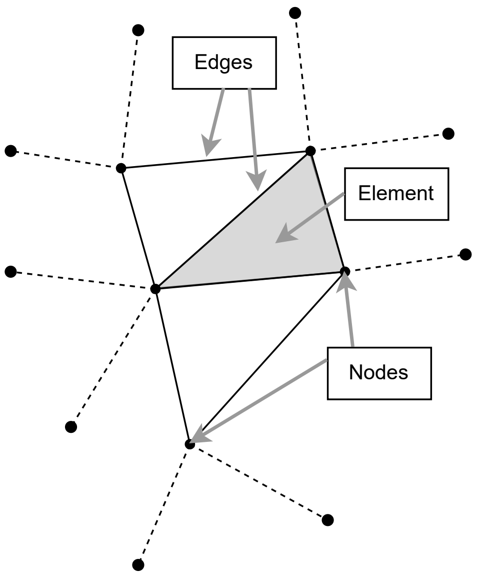
Figure 2.1: Summary of the CFD workflow for a standard design process.

Meshes consist of a set of nodes connected by edges, and where edges form closed curves, the enclosed space is defined as an element. A simple 2-D example of this is shown in Fig. 2.2. In 2-D geometries and on the surface of 3-D geometries, elements take the shape of 2D polygons, and within the domain of 3-D geometries the elements form polyhedrons. Meshes can be categorised as either structured or unstructured. Structured meshes contain regularly spaced nodes which result in regularly shaped elements. This has the benefit of leading to matrices that are easier to handle and are very easy to generate. Unstructured meshes comprise of irregularly spaced nodes, forming irregularly shaped elements. This irregularity makes these grids significantly more complex to generate however they are much more effective at fitting complex geometries where structured grids would be unable to do so [109]. The size of each element is dictated by the spacing at each node, which is defined as the average distance at each node from it's connected neighbours.

The first step in the generation of a 3-D discrete mesh from a continuous CAD model is discretization of the intersection curves between NURBS surfaces. Then, each surface is discretized, using the boundary discretization as the initial starting point. This ensures continuity between surfaces is maintained. The result is referred to as a surface mesh. The enclosing volume domain between the surface mesh and a prescribed far field is then discretised to create a volume mesh. The size of the element of the volume mesh is dictated by the existing surface discretization and the desired spacing throughout the domain.



(a)



(b)

Figure 2.2: A simplified example of a discretized mesh showing the discretization error of a continuous curve between the smooth curve and discrete elements, Fig. 2.2a and the key components of a mesh, Fig. 2.2b.

2.2 Mesh Quality Metrics

The discretization process introduces some error into the solution through its approximation of the exact geometry. Mesh quality is critically important to the accuracy of computational methods [59, 11]. As such, mesh dependency studies are performed to ensure the solution field is being driven by changes in the geometry and not changes in the mesh. Metrics are used to quantitatively define the quality of the elements across the mesh.

There are a range of quality metrics that can be used for this. In this thesis, the quality of planar faces used in the geometry surface mesh is of particular importance. These elements influence the quality of the volume mesh generated around it, and so it is important these elements are of suitable quality to avoid repercussions within the volume mesh.

The metrics considered here for planar triangular elements are size, shape and size-shape, and for planar quadrilateral elements the size, skew and size-skew metrics [60]. The skew metric is used in favour of shape metric for quadrilateral elements due to long thin anisotropic elements often being used to generate meshes for aerodynamic applications. The shape metric would result in poor mesh quality being erroneously recorded for the rectangular anisotropic elements.

The planar faces of 3-D elements can be translated onto a 2-D plane due to lack of curvature. Therefore, all calculations presented within this section are given using a 2-D coordinate system. A subscript following a metric definition, f^x, $x \in \mathbb{N}$, represents the number of nodes in the planar element that this calculation is valid for. For example, $x = 3$ for triangular elements and $x = 4$ for quadrilateral elements. Where omitted, it is assumed to be valid for any planar element.

2.2.1 Shape Metric

Define $\zeta = \frac{A}{A_r}$ as the ratio between the element area A, and a reference area, A_r. Then the relative size metric for an element is defined as:

$$f_{size} = min(\zeta, \frac{1}{\zeta}) \tag{2.4}$$

The reference area can be defined using a constant or calculated as a function of position in the mesh. One example of a constant definition is to take the average size of all elements on the surface, specifically A_r is the total surface area of the geometry divided by the total number of surface elements [60]. This can be generalised to use any subset of elements defining a constant value in localised areas [103].

When comparing two meshes that share the same topology, such as a morphed mesh with it's original baseline mesh, the reference area for calculating mesh quality in the morphed mesh can be taken as the respective area for each element in the baseline mesh [23]. This can highlight areas of the mesh where quality has changed during the morphing process.

2.2.2 Shape Metric - Triangular

The shape metric is defined by starting from a Jacobian matrix,

$$\Lambda_k = \begin{pmatrix} x_{k+1} - x_k & x_{k+2} - x_k \\ y_{k+1} - y_k & y_{k+2} - y_k \end{pmatrix}$$

The subscript $k \in 0, 1, 2$ denotes the node for which coordinates are taken from and is calculated modulo 3. The shape metric is then defined in Equation 2.5 where the υ is a symmetric 2×2 matrix with rows and columns $i, j \in 1, 2$ defined as

$$\upsilon^k = \Lambda_k^T \Lambda_k = \begin{pmatrix} \upsilon_{11} & \upsilon_{12} \\ \upsilon_{21} & \upsilon_{22} \end{pmatrix}$$

In the case of triangular elements, the f_{shape} metric is independent of the node it is calculated at, hence the superscript k for υ is omitted in Equation 2.5. This metric returns a value between 1 for perfectly equilateral elements and 0 for completely degenerate elements. Some examples of elements with varying values between perfect and nearly degenerate are shown in Fig. 2.3.

$$f^3_{shape} = \frac{det(\Lambda_k)\sqrt{3}}{\upsilon_{11} + \upsilon_{22} - \upsilon_{12}} \tag{2.5}$$

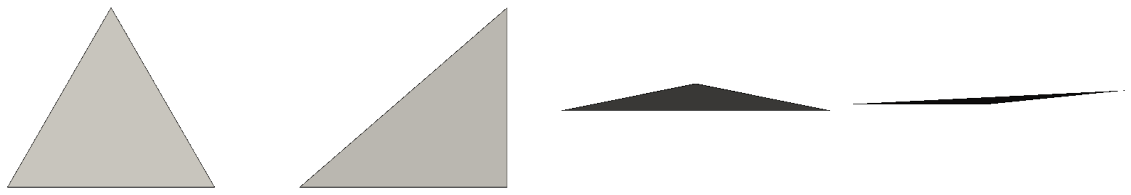

(a) $f_{shape} = 1.0$ (b) $f_{shape} = 0.85$ (c) $f_{shape} = 0.23$ (d) $f_{shape} = 0.05$

Figure 2.3: Examples of triangular elements with varying shape metric values.

2.2.3 Shape Metric - Quadrilateral

The Jacobian matrix for quadrilateral elements is

$$\Lambda_k = \begin{pmatrix} x_{k+1} - x_k & x_{k+3} - x_k \\ y_{k+1} - y_k & y_{k+3} - y_k \end{pmatrix}$$

where the same subscripts are used as for the triangular Jacobian, $k \in 0, 1, 2, 3$ but is now calculated modulo 4 [60]. Again $\upsilon^k = \Lambda_k^T \Lambda_k$ where υ^k is a 2×2 matrix with rows and columns $i, j \in 1, 2$. The skew metric is then given as in Equation 2.6 where unlike the triangular case, the script k is included. This is due to the calculation being dependent on the starting node used. The calculation is performed by averaging the contribution from each node.

$$f_{skew}^4 = \frac{4.0}{\sum_{k=0}^{3}((\sqrt{\upsilon_{11}^k \upsilon_{22}^k})/det(\Lambda_k))} \tag{2.6}$$

2.2.4 Combined Metrics

The size, shape and skew metrics can be combined as the product of powers, Equation 2.7. The compound metrics are named according to their constituent metrics, size-shape and size-skew. The parameters $a \in \mathbb{R}$ and $b \in \mathbb{R}$ can be adjusted to emphasise different metrics [60].

$$\begin{aligned} f_{ss} &= f_{size}^a f_{shape}^b \\ f_{ss} &= f_{size}^a f_{skew}^b \end{aligned} \tag{2.7}$$

2.3 Mesh Convergence Study

As highlighted in Section 2.1, the quality of a mesh can have a significant impact on the results of a CFD analysis. A highly refined mesh, coupled with an appropriate physics model, can accurately simulate the real world flow around a geometry. However, engineers are often limited by constraints on computational resource and time available, so trade-offs must be made between desired accuracy and available resource. When discretising the geometry, the mesh forms an approximation of the exact geometry and a degree of error is introduced relative to the exact solution. As the mesh fidelity increases, this error reduces. Beyond a certain point, an increase in the fidelity of the mesh will have a negligible effect on the result between two meshes; however, it will significantly impact the computational resources required and the wall clock runtime of the solution [29]. Therefore, a mesh convergence study must be performed to determine how the CFD result is affected by different meshes. Both the resolution of the boundary layer resolution and the element density are considered within both surface and volume meshes.

Computational methods such as FVM are iterative processes. The solution field for an appropriately configured mesh and solver will converge steadily towards a steady solution. The solution is generally considered to be more accurate the more iterations that are performed. The cut-off point in these simulations for the FLITE3D solver is often defined when the log density residual reaches a certain value. In some transonic simulations where the real flow conditions are often very unsteady, the log density residual may not converge below a certain threshold and may instead fluctuate. A different cut-off point must therefore be determined, after which the increase in computational time is deemed not to significantly change the outcome of the simulation.

In the remainder of this section, the criteria considered for iterative converge are discussed, as well as the mesh convergence study methodology for both the boundary layer resolution and the element density.

2.3.1 Convergence Criteria

The solution fields generated in these tests are both viscous and compressible simulations in a regime that is commonly known to be highly unsteady. There is therefore a possibility that the residual values will fluctuate about an equilibrium point rather than smoothly converging. The convergence criteria of a solution must therefore be considered [29].

There are three methods used to configure the convergence criteria for the FLITE3D solver: the magnitude of the log density residual, the maximum number of iterations or the total wall-clock duration. The log density residual is calculated as the log of the relative difference between the density value at a given timestep and that of the previous timestep. Alternatively a set number of iterations can be defined that the solution will terminate after, or the program runtime can be monitored to terminate after a set time. Fig. 2.5 shows the convergence history of the solution field for an example Skylon mesh at $M = 1.2$ using a mesh of approximately 40 million elements.

The FLITE3D solver used in this study can be parallelised over any number of available Central Processing Unit (CPU)'s. Figure 2.4 shows the iterations per hour completed compared to the number of CPU's used. The High Performance Cluster (HPC) these tests were run on has 40 CPU's per node (computer). It was identified that sharing a node with another job could impact the run time of the simulation, so the CPU count in Figure 2.4 increases in increments of 40 to ensure the job makes maximum use of the allocated node. As the number of CPU's increases, more iterations are completed per hour, up until a CPU count of 200, where the rate of improvement diminishes, highlighting limitations in scalability due to factors such as communication overhead between processors. Based on the values in this figure, it was decided that 120 CPU's would be used for each test in this study to balance a high iteration rate with computational expense. Using an additional 40 CPU's (or 1 node) would only lead to a 5% increase in speed for a 25% increase in resource cost.

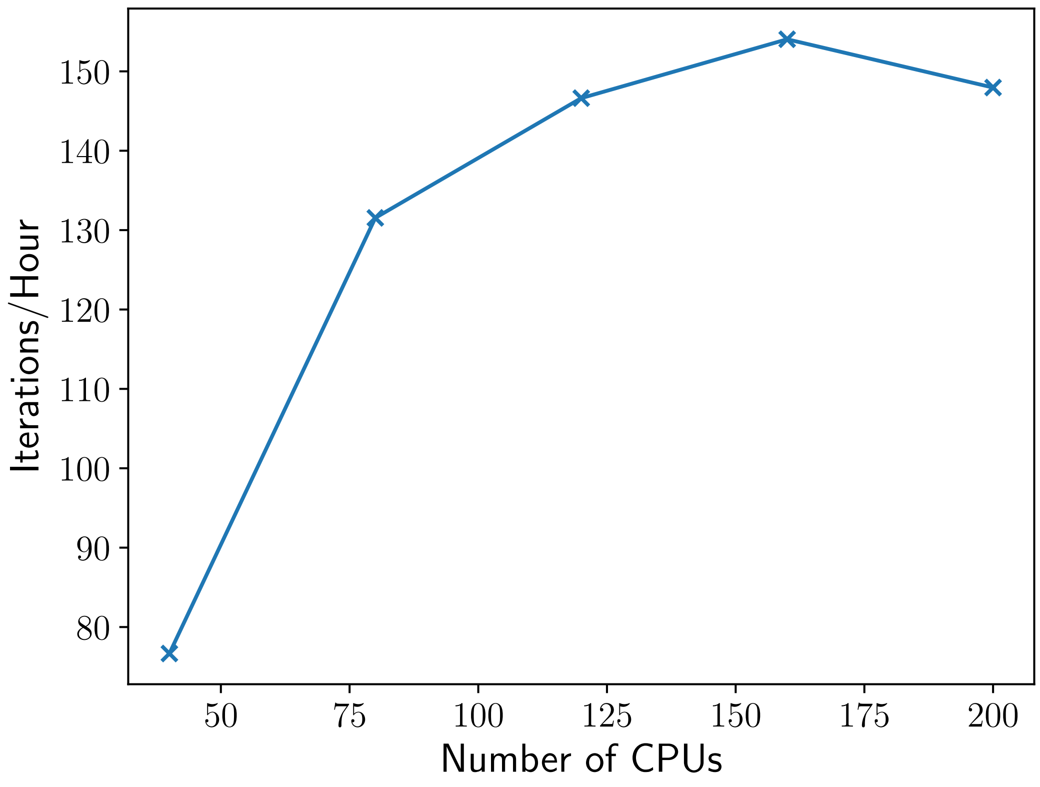
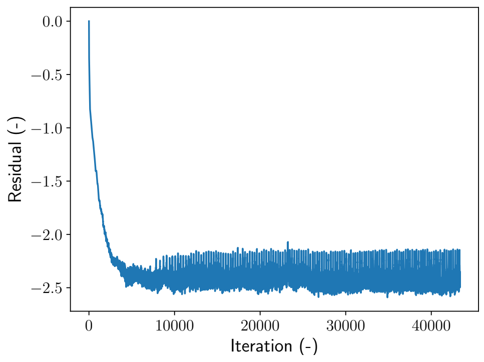
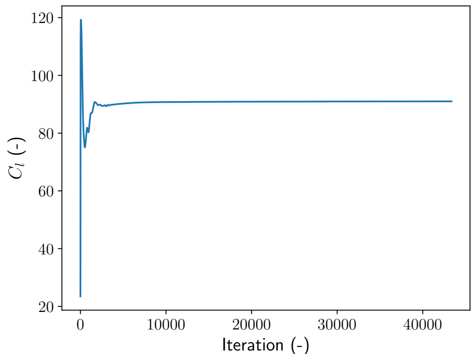


Figure 2.4: Comparison between number of iterations completed per hour vs CPU count in increments of 40.

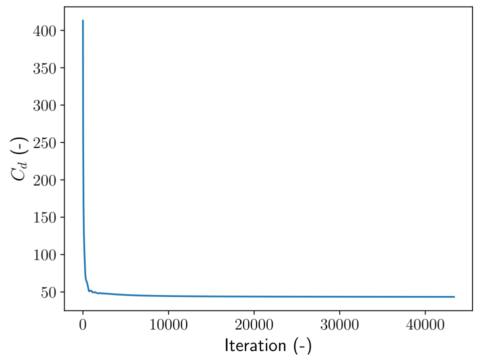
The convergence criteria was set to terminate when the log density residual reached -5. The first 250 iterations for each of the coefficient convergence plots fluctuate significantly due to the uniform initial conditions across the domain requiring some time steps to settle towards the solution. It is evident from this figure that the density residual does not converge to -5, instead fluctuating around -2.5. Although the aerodynamic coefficients appear to converge, they do change at a rate of approximately the order $\times 10^{-5}$ per iteration. Even at just 3000 iterations, the rate of change is as low as approximately 0.001 per iteration.



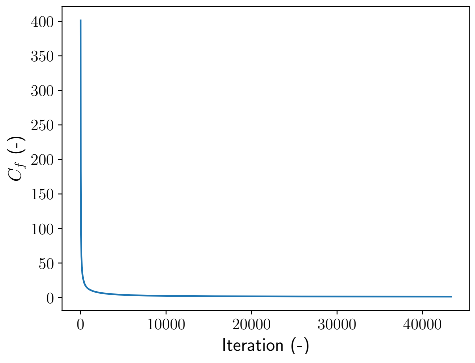
(a) Density Residual



(b) C_l



(c) C_d



(d) C_f

Figure 2.5: Convergence history plots for coarse Skylon mesh of approximately 40 million elements.

Terminating solutions at this point can provide a preliminary level of stability for exploratory optimization studies, however it is important to recognize that incomplete convergence can introduce misleading information into the optimization process, which in turn could direct the search toward suboptimal or incorrect solutions. The convergence threshold is carefully considered with the aim of balancing computational efficiency and the reliability of results. It is therefore acknowledged that limiting the iteration number in this way impacts the reliability of the results from this study. Numerical results from studys performed with this limitation should be interpreted with caution. The findings will reflect approximations rather than fully converged solutions and will confirm the functionality of the framework, rather than providing conclusive performance results.

2.3.2 Boundary Layer Convergence Study

The resolution of the mesh close to the surface of a geometry is critically important when handling viscous flow. An insufficiently resolved mesh in the boundary layer does not appropriately capture all of the complex flow features. The boundary layer construction method in the FLITE3D solver uses the advancing front method to build layers at user-specified heights out from the surface mesh into the surrounding volume [101]. The heights of these layers can be manually specified or generated using functions that are typically dependent on variables such as; first layer height (y_1), growth rate (r), the number of layers required (n) and the final layer height (y_{max}). A common method of calculating the height of each layer is using a geometric progression to calculate consecutive layer heights, Equation 2.8, a function of y_1, r and n. The growth rate is often $r = 1.2$ for this method.

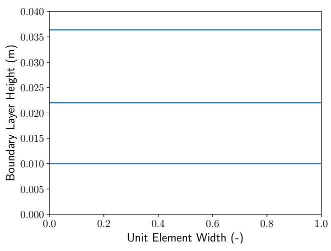
$$\text{Boundary Layer } = \{y_i = y_1 \cdot r^{i-1} : i \in [1, n] \subset \mathbb{N}\} \tag{2.8}$$

Rather than specifying the growth rate, the final layer height y_{max} can be specified, and an appropriate growth rate determined to achieve n layers between y_1 and y_{max} can be calculated using Equation 2.9. The layers between y_1 and y_{max} are then calculated using Equation 2.8.

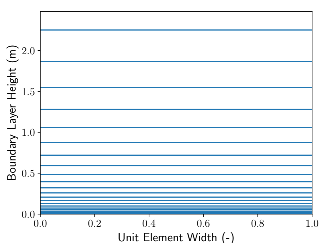
$$r = \left(\frac{y_{max}}{y_1}\right)^{1/(n-1)} \tag{2.9}$$

This alternative selection of parameters offers more control over the density of elements within the boundary layer, allows easier control blending the boundary layer into the surrounding mesh, and prevents the need for carefully calculating growth rates to define a reasonable boundary layer. Fig. 2.6c shows different boundary layer height constructed using both of these methods. When using a fixed growth rate of 1.2 the overall thickness of the boundary layer expands rapidly as the number of layers increases. This quickly limits the number of layers that can be included. When the elements in the boundary layer reach the same magnitude as those in the freestream, the boundary layer stops growing. This may occur before the desired number of layers is reached.

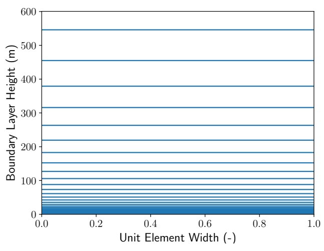
Specifying the final layer height instead ensures that the final layer can be chosen to be a similar size to the cells surrounding the boundary layer, and that the number of layers is as specified. When defining the boundary layer like this, Fig. 2.6c shows that the boundary layer can easily be constructed to include the desired number of layers and final cell height without over inflating the total boundary layer height by calculating the r using Equation 2.9.



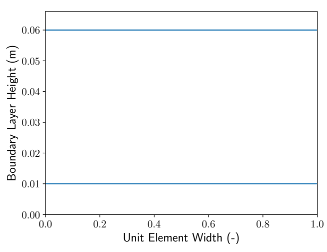
(a) $n = 2$, $r = 1.2$



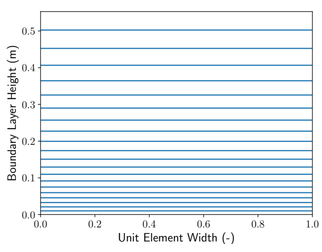
(b) $n = 20$, $r = 1.2$



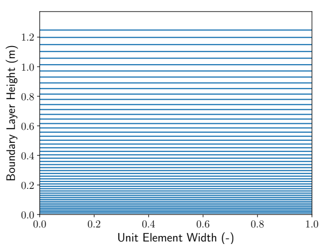
(c) $n = 50$, $r = 1.2$



(d) $n = 2$, $y_{max} = 0.05$



(e) $n = 20$, $y_{max} = 0.05$



(f) $n = 50$, $y_{max} = 0.05$

Figure 2.6: Boundary layer heights. Figures 2.6a-2.6c are calculated using geometric progression with $y_1 = 0.01$ and $n = \{2, 20, 50\}$ layers, respectively, and a growth rate of 1.2. Figures 2.6d-2.6f are calculated by defining the maximum layer thickness, with $n = \{2, 20, 50\}$ layers, respectively, with a maximum layer height $y_{max} = 0.05$, and the growth rate is calculated using Eq 2.9. Comparing the heights of the boundary layer in 2.6c and 2.6f, we can see how a reasonable boundary layer thickness can easily be generated by calculating the growth rate, rather than trying to define one.

In the convergence study, the relation between each variable in Equation 2.9 and the aerodynamic coefficients is explored. The baseline surface mesh introduced in Section 1.4.2 is used, shown in Fig. 2.15.

2.3.2.1 First Cell Height

The first variable explored is the height of the first cell away from the surface, y_1. To determine y_1, a dimensionless distance from the wall is introduced, y^+ [114]. This value is defined in Equation 2.10 where y represents the distance from the geometry surface of the closest node in the surrounding volume, ν is the kinematic viscosity of the freestream and u^τ is the friction velocity.

$$y^+ = \frac{yu^\tau}{\nu} \tag{2.10}$$

The friction velocity represents a velocity scale close to the surface and is calculated in Equation 2.11 using the wall shear stress τ_w for a given the freestream density ρ and [114]. The wall shear stress can be calculated from the skin friction coefficient normalised by the dynamic pressure as in Equation 2.12.

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{2.11}$$

$$\tau_w = c_f \cdot q_\infty \tag{2.12}$$

During CFD iterations, τ_w can be calculated along the surface of the geometry from the solution variables [113]. However, c_f, and therefore τ_w, is not known a priori. An approximation must be made to determine an appropriate initial value for the initial solution, and can then be refined based on calculated values of c_f. Many empirical approximations exist such as the Schlicting skin friction coefficient approximation Equation 2.13 [88].

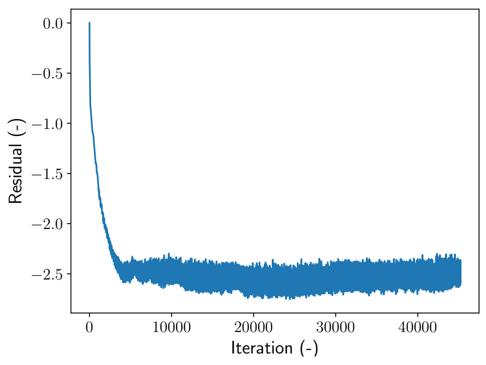
$$c_f = [2log_{10}(Re) - 0.65]^{-2.3} \tag{2.13}$$

The inverse of Equation 2.10 can be used to calculate y_1 for a target y_+. The desired y_+ value is dependent on the turbulence model used. For the SA turbulence model, a target y_+ between 1 and 10 is often considered a reasonable value. Using this approximation, for a target $y_+ = 1$, along with the freestream conditions calculated using the altitude data from Section 1.4.4, the height of the first layer is $y_1 = 1.8476e - 06$.

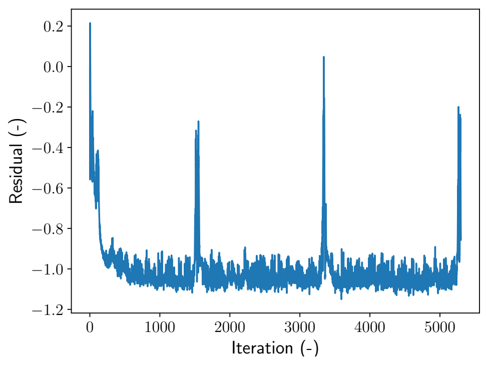
Another method of approximating c_f to determine y_1 is to use the Prandtl approximation, Equation 2.14 [34]. Using this method, the predicted c_f is 2.0, significantly higher than when using the Schlichting formula, and using this approximation results in a first layer height of $6.3717e - 08$.

$$c_f = 0.074Re^{-1/5} \tag{2.14}$$

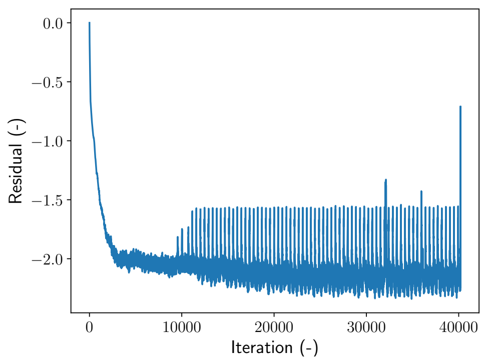
The convergence history of the log density residuals for solutions where the heights of the first layer in the boundary are calculated using the Schlichting and Prandtl formulas are shown in Fig. 2.7a and 2.7b respectively. In the case of the Prandtl approximation, the residual did not fall below -1.15 and included large fluctuations, reaching values that went back close to 0. This is significantly more unstable than that of solution using the Schlichting approximated first layer height.



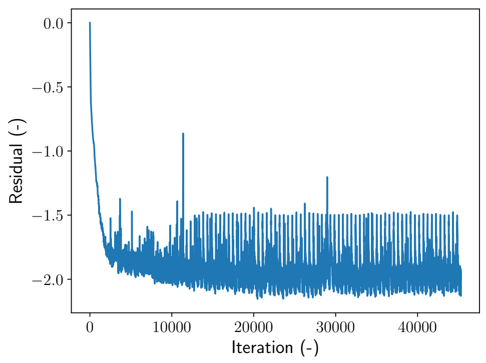
(a) Schlichting



(b) Prandtl



(c) Manual A



(d) Manual B

Figure 2.7: Density residual convergence using different methods of estimating the first layer height for the boundary layer.

Two additional tests were performed using c_f estimates between the two empirically estimated values to determine if this was an anomalous result or not. The predicted friction coefficients and calculated CFD values of all tests are given in Table 2.1.

Method	y_1	c_f Predicted	c_f Actual	y^+ Target	y^+ Actual
Schlichting	1.8476×10^{-6}	0.002416	0.0025	1.0	1.0182
Manual A	1.7808×10^{-7}	0.26	0.00932	1.0	0.1893
Manual B	1.3027×10^{-7}	0.486	0.01068	1.0	0.1483
Prandtl	6.3717×10^{-8}	2.031	0.01379	1.0	0.08239

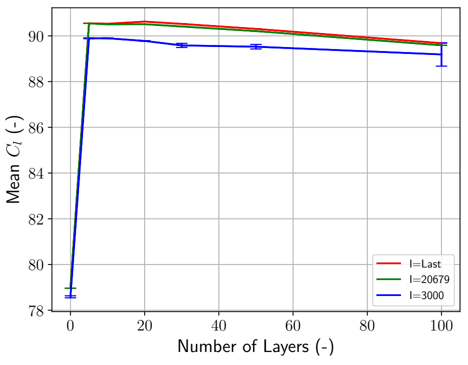
Table 2.1: Computational results of Skylon volume meshes created using different y_1 values. Each boundary layer had 30 layers up to a maximum height of 0.05m

For each test, the c_f values from the CFD tests were then used to determine the actual y^+ value of the cell height of the first layer, populated in the final column of Table 2.1. When using Schlichting's formula to approximate the height of the first layer, the actual value of y^+ was 1.08, close to the target of 1.0, suggesting that it was an accurate choice of the estimated value of y_1 for this test. Alternatively, the calculated y^+ value for the Prandtl estimated test was 0.08. The manually selected y_1 values performed similarly to the Prandtl approximation. The two manually selected tests demonstrate that the Prandtl estimated was not anomalous but is repeatable behaviour for $y_1 < 1.8476e-06$. As shown in Fig. 2.7c and 2.7d, they also have unstable convergence patterns.

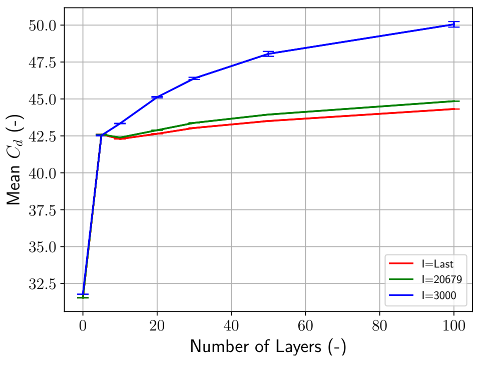
For the Spalart-Allmaras turbulence model, the ideal y^+ value is between 1 and 10. A value of $y^+ << 1$ can suggest that the mesh will result in an inaccurate prediction of the flow behaviour using this turbulence model. These tests confirm this and suggest that the initial boundary layer height calculation using Schlichting's formula to approximate c_f and calculate the height of the first layer is a suitable value to use for further test cases.

2.3.2.2 Number of Layers

The total number of layers in the boundary layer, n, is also explored. The initial layer is fixed using the approximate value from Section 2.3.2.1, and the final layer height was arbitrarily selected as 0.05. Fig. 2.8 shows the convergence history plots for the lift and drag coefficients as the total number of layers in the boundary layer increases. The convergence criteria for each solution were if the log density residual dropped below -5, indicating that the solution has reached a suitable level of convergence or if the solution ran for more than 3 days wall clock time. The value for coefficient was then calculated as the average of the final 250 iterations as defined in Section 2.3.1.



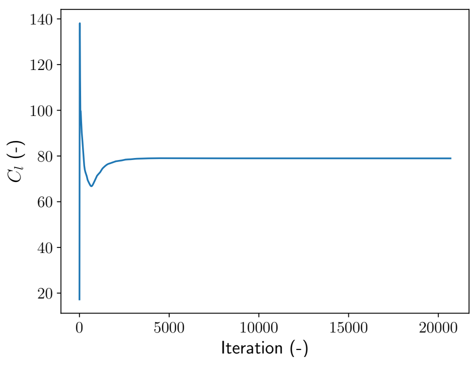
(a) C_l



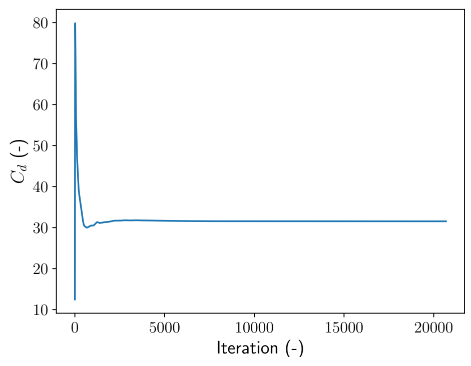
(b) C_d

Figure 2.8: Lift and drag coefficient convergence history as the total number of layers in the boundary increases for a fixed y_1 and y_{max}. Coefficient values are calculated as an average of the final 250 iterations. The coefficient values at specific iteration numbers are also shown (3000 and the lowest total number of iterations any of the tests ran for).

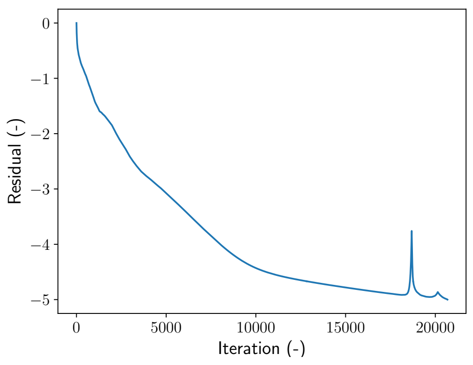
The only solution to converge with a density residual of -5 was when 0 boundary layers were used. The residual and coefficient convergence plots for this test are shown in Fig. 2.9. However, coefficient values are likely to include a high degree of error compared to the physical performance of the geometry.



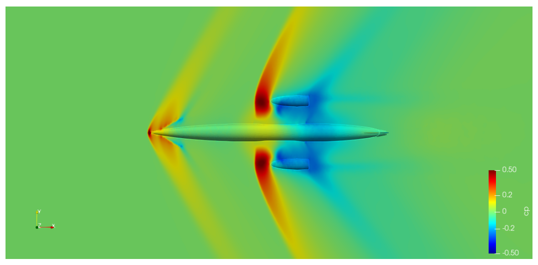
(a) C_l



(b) C_d



(c) Density Residual



(d) Solution Field

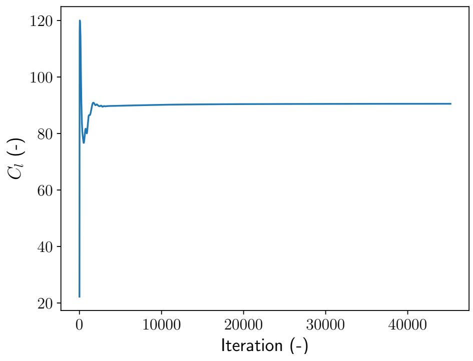
Figure 2.9: Residual and coefficient convergence histories, and plan view of the solution field when using 0 boundary layers. The solution field plot is cut at $z = 0.0$ and coloured by density.

All other solutions did not pass below a density residual of -3 and fluctuated within approximate limits. An example of the density residual convergence exhibiting this behaviour is given in Fig. 2.10c and is similar to the behaviour presented in Fig. 2.5. It is clear from the coefficient residual plots for the same test that the lift and drag slowly converge.

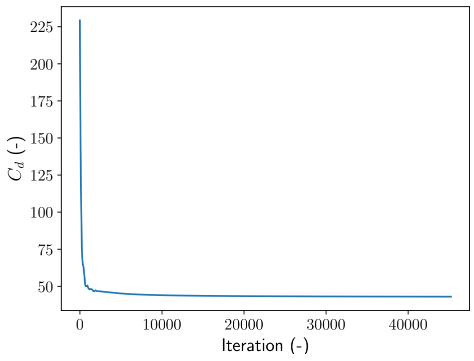
In physical applications, CFD is used as a tool to determine an accurate prediction of the aerodynamic properties of a geometry. For a mesh to be considered converged, the gradients of the convergence plot must be close to 0. However, for the purposes of this exploratory case study, the relative performance of different geometries is of interest.

Each solution ran for 3 days, however each completed a different number of iterations within this time due to differing mesh resolutions. All solutions steadily converged over time, by smaller and smaller amounts each iteration. As suggested in Section 2.3.1, for exploratory studies, it is suggested that solution fields can be terminated early and the relative performance of geometries can be compared after a small number of iterations.

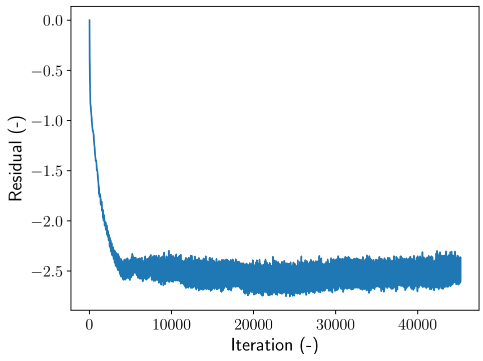
Fig. 2.8 therefore also shows the coefficient values taken at different stages of their convergence history. Every test in Fig. 2.8 ran for at least 20679 iterations, hence this value is used as the largest possible value that can be used to commonly compare. At this point, the change in coefficient values as the number of layers increases changes by very little, no more than approximately 1%. Even comparing solutions at as few as 3000 iterations into the convergence history, coefficient values are still within approximately 10% of their final values.



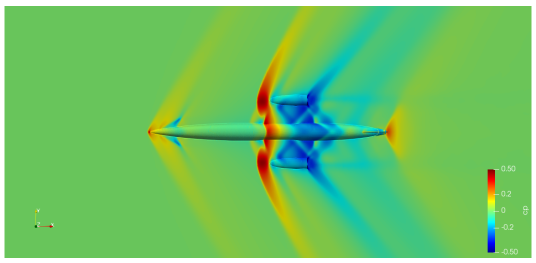
(a) C_l



(b) C_d



(c) Density Residual



(d) Solution Field

Figure 2.10: Residual and coefficient convergence histories, and plan view of the solution field when using 30 boundary layers. The solution field plot is cut at $z = 0.0$ and coloured by density

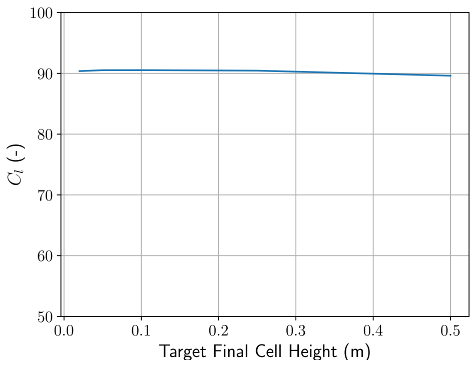
After using just 30 layers, the coefficient values do not change by more than 5% from the test result that uses 100 layers. The reduction in computation time taken is a contributing factor that should be considered here when making the decision of how many layers are a suitable trade-off between computational accuracy and computational cost.

2.3.2.3 Final Boundary Layer Cell Height

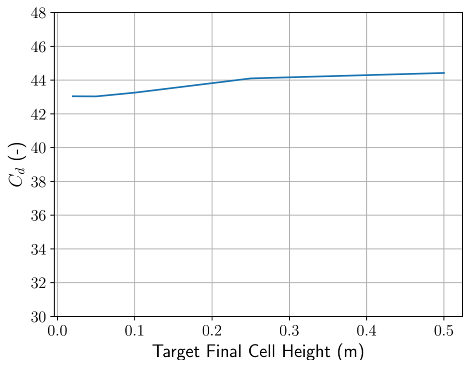
While the behaviour of the height of the first boundary layer is critically important, it is also important to ensure that the boundary layer is fully captured. Using the same equation as before, Equation 2.9, but now varying y_{max} results in different growth rates. If increasing y_{max}, r increases, and the overall effect of this on the boundary layer is that it's total height also changes. Tests were carried out with values of 30 and 100 for the total number of layers, and both tests used the Schlichting estimated height of the first layer from Section 2.3.2.1.

y_{max} (m)	$n = 30$		$n = 100$	
	sum(y_i) (m)	r (-)	sum(y_i) (m)	r (-)
0.02	0.073	1.378	0.223	1.098
0.05	0.169	1.422	0.510	1.109
0.1	0.319	1.456	0.959	1.116
0.25	0.747	1.503	2.222	1.127
0.5	1.427	1.539	4.213	1.135

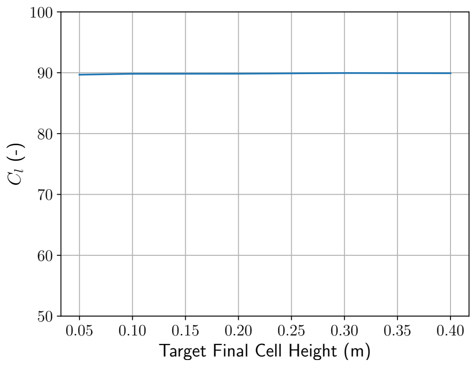
Table 2.2: Total height of boundary layer heights and growth ratios for a given final cell height. Two different datasets with different numbers of layers are given.



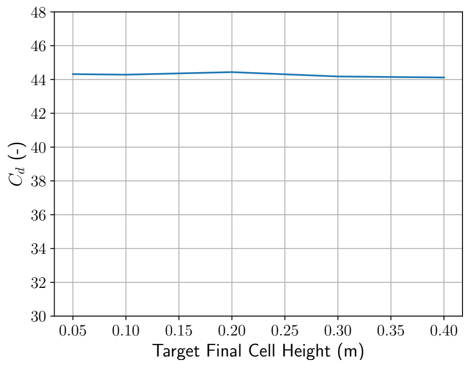
(a) $n = 30$



(b) $n = 30$



(c) $n = 100$



(d) $n = 100$

Figure 2.11: C_l and C_d values as the target final cell height increases. The total number of layers in each graph is fixed and highlighted in figure sub-captions.

The target mesh spacing for elements in the volume surrounding the boundary layer was 0.3m. The values for y_{max} were selected such that they gradually reached this spacing and slightly exceeded it. Each value of y_{max} was then tested using $n = 30$ and $n = 100$. Table 2.2 contains a summary for these tests and the total height of the boundary layer in each test. Fig. 2.11 shows the convergence of C_l and C_d as y_{max} increases for both tests. These figures show that the selection of maximum cell height values has very little effect on the resulting aerodynamic coefficients for fixed values of y_1, and

n. When $n = 100$, the graph is almost entirely plateaued for C_l and shows neglidible change in C_d by less than 1%. This suggests the boundary layer is likely already captured and so changing y_{max} has little affect the result. When $n = 30$ the most significant change is seen, with C_d changing by approximately 3% as y_{max} increases.

Where computational power is not critical, these figures show there is no detriment to using a larger number of layers and a slightly larger y_{max} to ensure the boundary layer is captured at $M = 1.2$ for this Skylon geometry, however for optimisation studies with limited computational resource, 30 layer with a $y_{max} = 0.05$ is sufficient.

2.3.2.4 Summary

The tests conducted within this section were carried out with the objective of determining a suitable resolution of the boundary layer for the Skylon spaceplane for a mesh that can be used to demonstrate the implementation of several optimisation algorithms. Given the convergence histories and with the aim of reducing the computational cost of each solution calculation, it is determined that a boundary layer with $y_1 = 1.8476e-06$, 30 layers, and $y_{max} = 0.05$ will produce a sufficiently resolved boundary layer.

2.3.3 Sources Convergence Study

Element size within the volume surrounding the boundary layer is also an important consideration when generating discrete meshes. This is commonly controlled by specifying the spacing of the mesh at each element, Section 2.1. In FLITE3D, the mesh spacing can be specified by defining a background mesh over the domain or using source distributions in localised regions. Both methods can be used in conjunction with each other to carefully control the mesh throughout the domain. In this section, a brief introduction to these methods is given followed by a convergence study controlling the mesh resolution using different sources. All geometries used throughout this thesis are 3-D so these concepts are introduced in 3D, however are equally as applicable in 2-D.

The background mesh divides the domain into tetrahedral elements and the spacing is defined at each node. The spacing for each node throughout the domain is then determined by interpolating across the background mesh. This method offers accurate control of the spacing across the global domain; however requires significant direct input from an engineer to define the resolution of the mesh in localised regions around the geometry using this method. Source distributions are an alternative method that can be used to control the element sizes of a mesh in specific regions of the geometry. They control the spacing by defining a function of the distance x from a specified centre. Source distributions can be defined as point, line, or triangular sources.

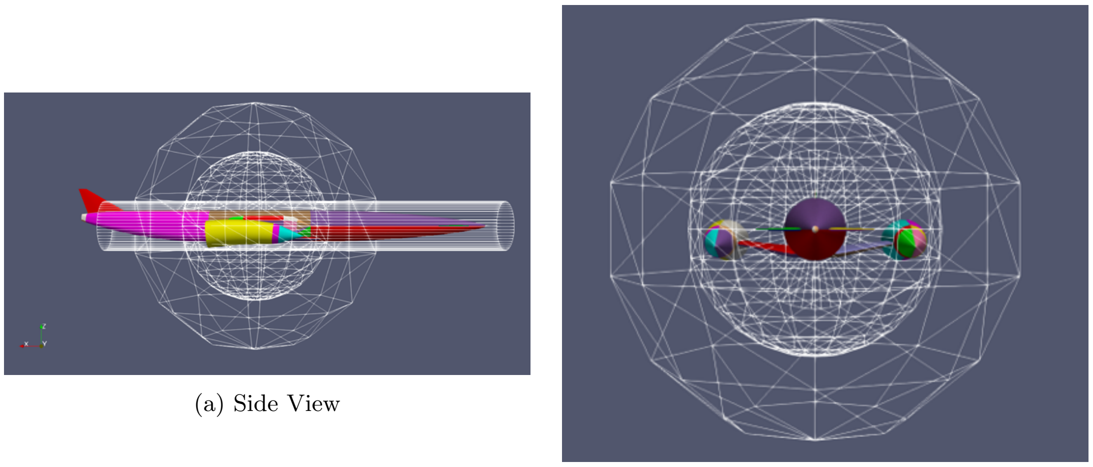
Point sources are defined by a central point in the global space $p_1 \in \mathbb{R}^3$, with two radii r_1, r_2 defined around it, and a specified spacing s_1. All mesh nodes at least a distance r_1 away from the center, $||x - p_1|| <= r_1$, will have a spacing of s_1. Nodes between r_1 and r_2 away from the center, $r_1 <= ||x - p_1|| <= r_2$ gradually decay to that of the background mesh.

Line sources extend this concept. Two point sources are defined, and the radii/spacing defined at each point is extended along the line connecting the two points. This is particularly useful for increasing spacing around long slender bodies, such as a fuselage or leading edges of wings.

Triangle sources are defined in a similar way to line sources, with one additional node defined, which comprises a triangle surrounding a region. This type of source can be useful when capturing sharp edges, such as trailing edges. One of the nodes can be defined on the sharp edge, with the other 2 extending away and along the body.

2.3.3.1 Initial Source Positions

An initial grid was setup to form a basis for the source convergence study. The background mesh was initially defined as a cube of tetrahedra that encapsulated the complete domain and had a coarse global spacing of 100m. Two point sources and a line source were defined around the geometry. The approximate size and location of these sources is shown in Fig. 2.12.


(a) Side View

(b) Front View

Figure 2.12: Approximate location and size of the two point sources and one line source over the Skylon geometry to form the baseline source placement.

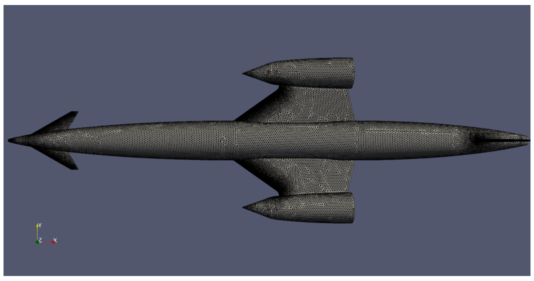
The two point sources are centred at approximately the centre point of the geometry at $[-35, 0, 0]$. The radii for the inner sphere are $[15, 30]$ and for the outer sphere they are $[25, 50]$. The line source is located along the length of the fuselage from tip to tail and has radii $[5, 10]$. This fully captures the fuselage, as the radius of the fuselage is 3.15. All volume meshes generated in this section use the baseline boundary layer defined in Section 2.3.2.4.

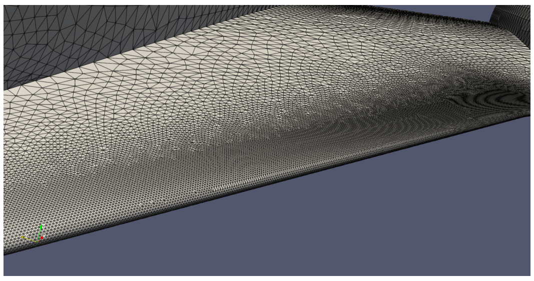
2.3.3.2 Curved Surfaces Refinement

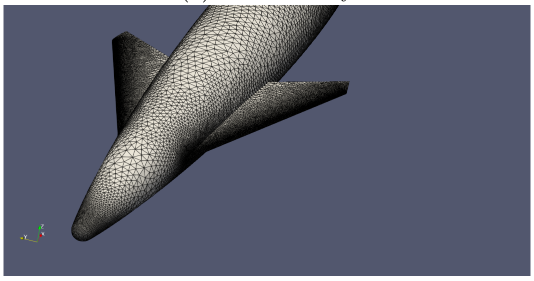
On top of the initial sources, additional sources are added in localised regions of the geometry. Attention is focused on surfaces with higher curvature. For example, the leading edges of the wings have significantly more curvature than the flat top or bottom of the wing. Surfaces with curvature require more elements to accurately capture than flat surfaces. Table 2.3 lists the locations of the additional sources used, the result of which is shown in Fig. 2.13. The spacing is calculated for each these surfaces to ensure the curve of a hemisphere at these points was discretised by approximately 12 nodes. The same line and point sources from the preceding section are used with spacing of 0.3 and 0.5 respectively.

Location	Source Type	Spacing
Canard Leading Edge	Line	0.01
Wing Leading Edge	Line	0.025
Fin Leading Edge	Line	0.008
Nose Tip	Point	0.03
Nacelle Tip	Point	0.1

Table 2.3: Additional sources used for higher fidelity mesh.


(a) Whole Body


(b) Wing Leading Edge


(c) Nose

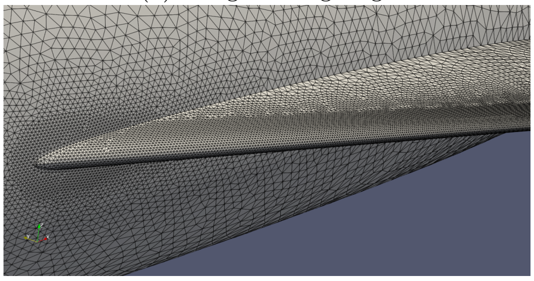
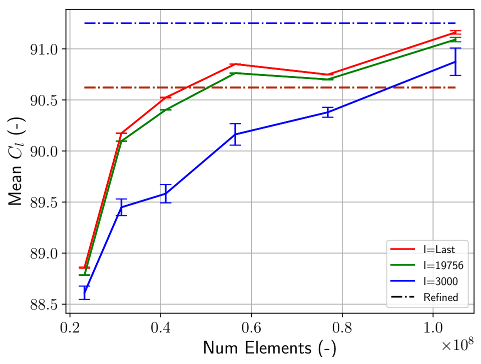

(d) Canard Leading Edge

Figure 2.13: Surface mesh for the refined Skylon mesh. Node spacing at key curved surfaces has been significantly refined.

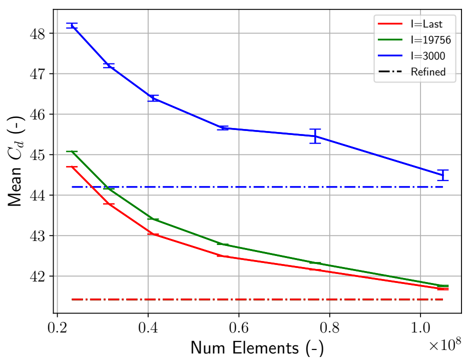
The resulting volume mesh for the geometry contained 20,474,040 nodes and 96,913,642 elements, increasing by a factor of approximately 2.4$\times$ relative to the initial source position and resolution. The results of the solution using this mesh are shown in Fig. 2.14, represented by the dot-dashed line, which are compared with those for coarser meshes. The increased mesh resolution and therefore element count results in an increase in computation time, with the solution running for 3 days and completing 19,756 iterations.

2.3.3.3 Initial Source Resolution

A range of values for the resolution of the initial non-localised sources, each differing by 10%, is explored. For the line source and inner point source, the spacing values range between 0.363 and 0.2, and the outer point source values range between 0.3 and 0.605. The relationship between overall element count in the volume mesh and the aerodynamic coefficients is shown in Fig.2.14.



(a) C_l Convergence



(b) C_d Convergence

Figure 2.14: Coefficient convergence as the total number of elements used in the volume mesh increased. Error bar represents the standard deviation of the previous 250 iterations. Values are compared at 3 different iteration numbers, the final value after 3 days runtime, the largest common iteration count across all tests (19756), and 3000 iterations. The value calculated by a significantly higher fidelity mesh is shown using dashed lines, again compared at different iterations of it's convergence history.

For each mesh, the error bar represents the standard deviation of the 250 iterations prior to the one for which the coefficient was recorded. So if the coefficient value was recorded after 3000 iterations, then the standard deviation is calculated over iterations 2750-3000. The values are recorded and compared at 3 different iteration numbers. Fig 2.14 shows coefficient values taken after 3 days runtime for each solution. Each of these solutions runs for a different total number of iterations, as a higher node count in higher fidelity meshes takes longer to calculate. The coarsest mesh explored completed 56,622 iterations in this time, while the finest mesh completed just 20,561. They are therefore also compared for an identical number of total iterations. To compare each at the same value, the largest common iteration number was used. A third comparison is also made, comparing each test after just 3000 iterations to determine if an approximation can be taken after a small number of iterations for quicker comparison between designs. After only a small number of iterations, the coefficient values are within approximately 2% of the C_l and about 7% of the C_d compared to the final values after 3 days of calculation. This demonstrates that a reduction in node count can achieve comparable results with only a slight reduction in error. This would not be an acceptable margin for operational use in an industry setting, however is suitable for a theoretical study to reduce the computational time and resource requirement.

2.3.4 Baseline Mesh

A key objective of this thesis is to explore the applicability of different optimisation methods to shape optimisation problems of this kind. In the interest of exploring various optimisation methods, given that computational resource is limited to a degree, total wall clock time is a key consideration when selecting a suitable resolution for the mesh. While much finer resolutions are typically employed in industry to ensure simulation accuracy, a coarser mesh is used in this study to reduce computational expense while illustrating the applicability of the chosen optimisation algorithms, demonstrating their feasibility and functionality in solving such problems, rather than achieving the best possible performance for a specific case. It is important to note that the function space defined by a coarse mesh differs significantly from that of a fine mesh. As such, solutions obtained using a coarse mesh may not accurately represent the true physical behavior of the system and should be interpreted with caution.

Therefore, the baseline mesh for optimisation studies throughout this thesis has; a boundary layer defined as in the summary of Section 2.3.2.4, two point sources with a mesh spacing of 0.3 and 0.5 respectively, and one line source along the length of the fuselage with a spacing of 0.3, as shown in Fig. 2.12. No leading edge refinement is used to keep the node count low.

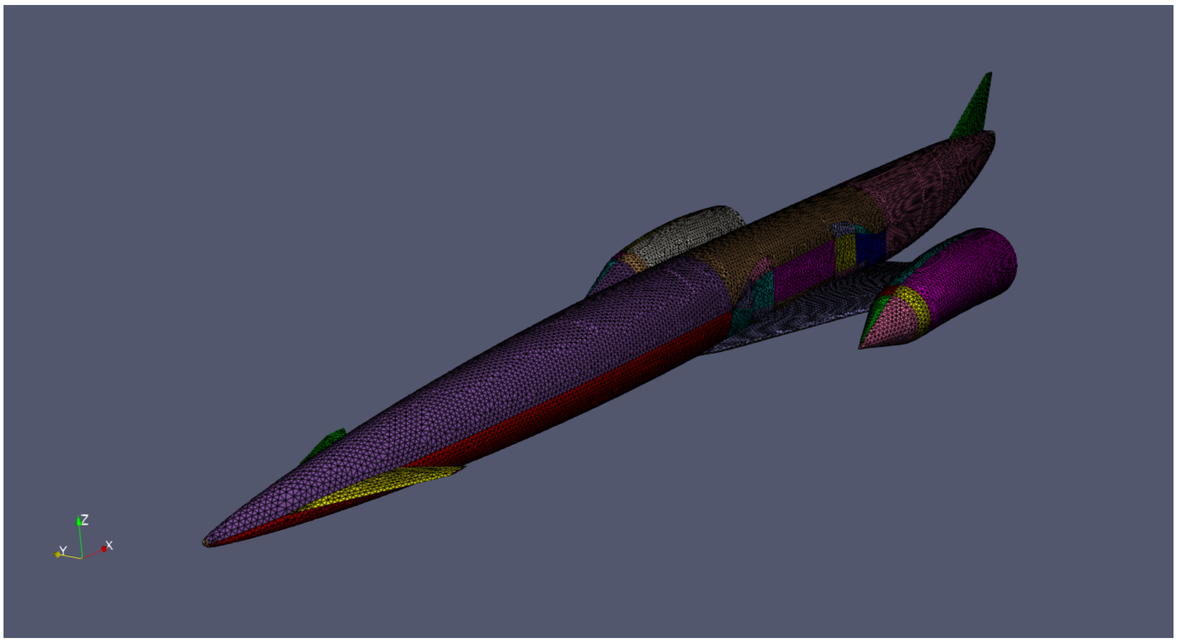
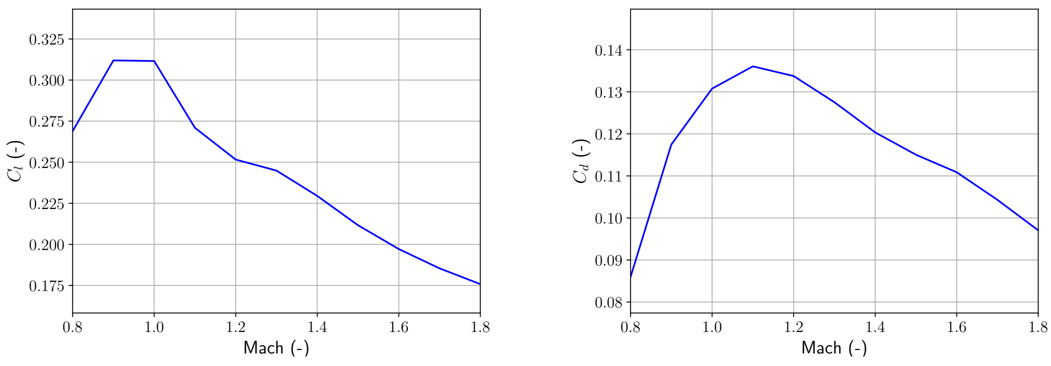


Figure 2.15: Baseline mesh for the Skylon spaceplane.

2.4 Baseline Performance

The performance of the baseline mesh needs to be analysed across a wider range of conditions. The altitude data for a given freestream Mach number shown in Fig. 1.3 is used in conjunction with the ICAO Standard Atmosphere to determine the freestream conditions for a range of Mach values through the transonic regime [74]. C_l and C_d coefficients for the geometry across the transonic regime are shown in Figure 2.16 and the Re values used in each test are given in Table 1.1.

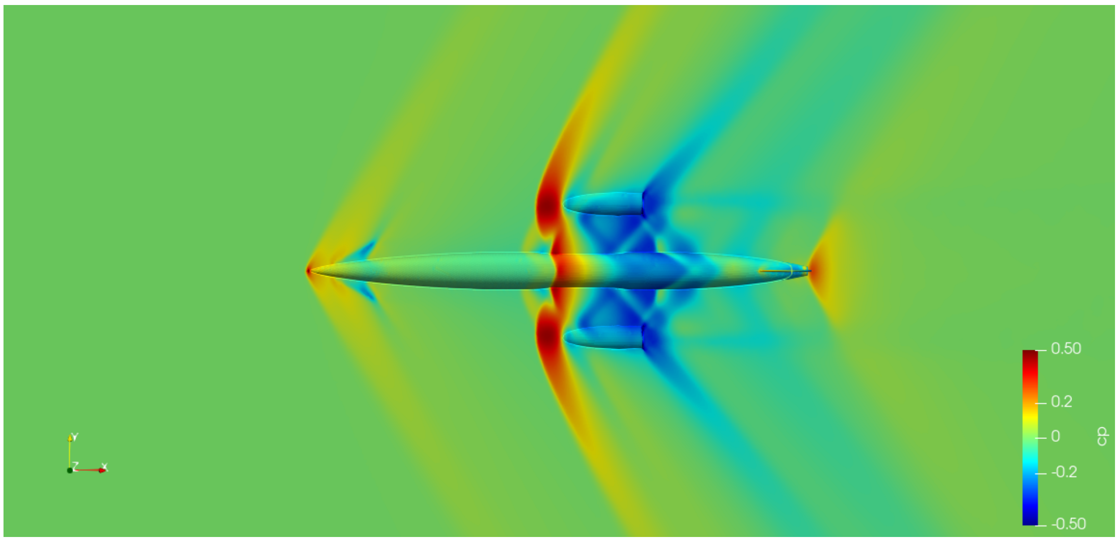
[H]



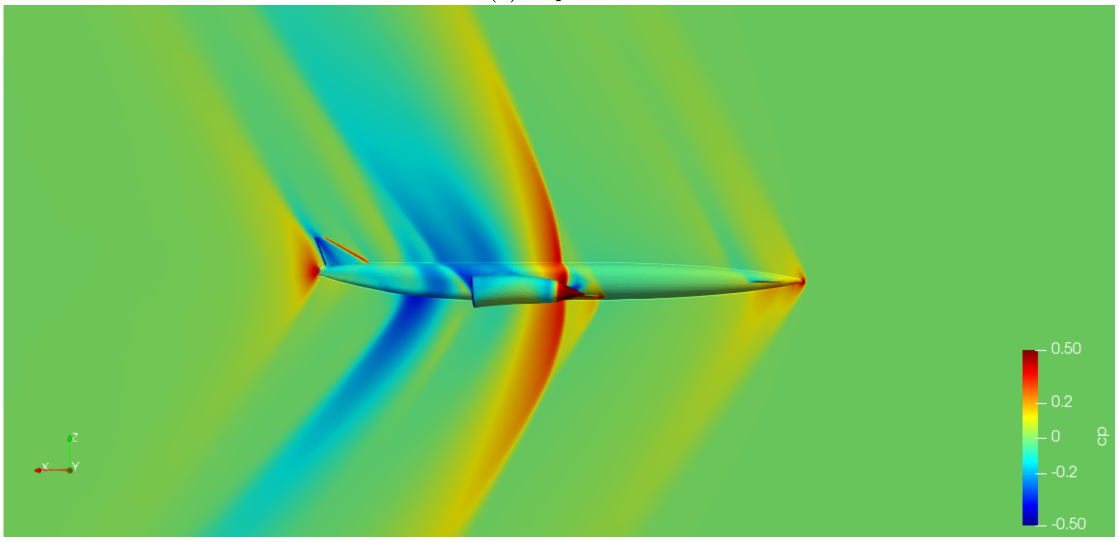
(a) C_l (b) C_d

Figure 2.16: Lift and drag coefficients of the Skylon C1 spaceplane in the transonic regime.

The maximum drag value occurs around $M = 1.1$. This correlates closely with the expected peak drag highlighted in Fig. 1.2 and based on this, the optimisation design point was selected as Mach 1.2 to capture the expected minimum specific excess power between thrust and drag.



(a) Top View



(b) Side View

Figure 2.17: Top and side profiles of the solution field for the baseline design of the Skylon C2 spaceplane run on a coarse mesh at $M = 1.2$ and incidence $\alpha = 4.0$.

Solution fields for the baseline design at Mach 1.2 are shown in Fig 2.17. The planform view in Fig. 4.7b shows that complex shock formations occur around the midsection, just above the wings. This likely contributes to a reasonable portion of the pressure drag experienced through this regime, making it an important section to target during optimisation to yield a reduction in drag.

Chapter 3

Mesh Morphing Framework

To enable optimisation of a design for a given parameterisation, a method of modifying the design is needed. In this chapter, a novel mesh morphing framework is introduced that uses Radial Basis Function (RBF) to maintain the surface continuity of surface meshes morphed from a set of user-defined parameters. The method is designed in a way that it can be decoupled from specific optimisation methods and supports the automated optimisation of a geometry based on the user-defined independent parameters and constraints.

The chapter first introduces a background of RBF, including a summary of their prior use in mesh morphing methods. Some additional case studies used to assist in the definition of the methodology are then introduced. The novel methodology is then discussed in detail including an overview of the method and how it can be used to achieve geometric constraints.

3.1 RBF Mesh Morphing

Originally developed for the representation of topological surfaces (such as height contours on a map), RBF is an approximation method that has been used for a wide range of applications since its initial introduction [41]. Initially called the Multiquadric (MQ) method, it was identified as a superior interpolation method to many others under development at similar times [32]. Also identified in this work was a similar method by Duchon using the Thin Plate Spline (TPS) method. These two functions are now considered to be two different basis functions that fall under the general method of RBF. Two decades after Hardy's development of the MQ method, a review paper summarised the range of applications for which the method had been used for [42]. The concept was extended to interpolating a combination of topological and bathymetric contours of a global map, within hydrology for mapping rain fall over an area, for improving the model of the earth's geoid, image reconstruction from a coarse image, and a range of other geological applications. These applications showed the highly effective application of Radial Basis Functions (RBF's) in a wide range of domains where interpolation of scattered data points is required. One of the key benefits of the method is that, although the distribution of scattered points has an impact on the results, it is not a limiting feature as is in some other methods [42]. For example, polynomial methods can suffer from instability leading to oscillations at the boundaries (Runge's phenomenon). In contrast, RBF's are mesh-free and adapt well to the distribution of scattered data naturally, without requiring structured grids.

A more recent application that has emerged is that of RBF mesh morphing, which is a shape optimisation method introduced in the early 2000s by multiple authors [23, 54]. In this section an overview of the mathematical description of RBF is given, parameter tuning is explored to show how their modification can affect the interpolation in different ways, and concludes by summarising current work on the application of RBF to mesh morphing for shape optimisation.

3.1.1 Definition

The mathematical definition of a RBF is $\Phi(\boldsymbol{x}) = \phi(d(\boldsymbol{x}, \boldsymbol{x_c}))$. The radial function, Φ, accepts a vector input and calculates the ordinate value at $\boldsymbol{x}$ using a univariate function ϕ. The univariate input to ϕ is determined using a metric, $d(\cdot, \cdot)$, to calculate the distance of x from the RBF centre, $\boldsymbol{x_c}$, making the function radially symmetric about this point.

Multiple RBF's can be combined as a weighted sum to form an RBF network, Equation 3.1. This is the format RBF's are most commonly used in across applications, where each known evaluation point forms a RBF centre. Often within literature, *network* is dropped from the name and the term RBF refers to this weighted sum.

This weighted sum can be used to approximate a real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ using known evaluations to construct an interpolation function $s(\boldsymbol{x})$ that approximates f. The known evaluation points form centres, and a radial function is defined over each centre. The interpolation function is then constructed as the weighted sums of the radial functions, as shown in Equation 3.1. These known locations, referred to as centres or sources, form a subset of the domain, $X_c = \{\boldsymbol{x_{c_i}} \in X : i \in [1, n] \subset \mathbb{N}\}$. The ordinates at the centres are collectively represented by the vector $\boldsymbol{f} = f(X_c)$.

$$s(\boldsymbol{x}) = \sum_{i=0}^{n} \lambda_i \phi(d(\boldsymbol{x}, \boldsymbol{x_{c_i}})) \tag{3.1}$$

Some examples of commonly used basis functions, ϕ, are presented in Section 3.1.2. For RBF's to be radially symmetric around their centre, a metric must be defined. The metric most commonly used in the implementation of RBF's is the Euclidean metric, Equation 3.2. Note in Equation 3.2 that the distance metric between a point $\boldsymbol{x}$ and a centre $\boldsymbol{x_c}$ is often defined in the literature as r.

$$d(\boldsymbol{x}, \boldsymbol{x_c}) = r = \|\boldsymbol{x} - \boldsymbol{x_c}\| = \sqrt{(\boldsymbol{x} - \boldsymbol{x_c})^2} \tag{3.2}$$

The weights within Equation 3.1, $\boldsymbol{\lambda} = \{\lambda_i : i \in [1, n] \subset \mathbb{N}\}$ are calculated for each centre. To calculate these weights, a condition is imposed that all evaluated points must be recovered by the interpolation function, and the condition is met by ensuring that Equation 3.3 is satisfied.

$$s(\boldsymbol{x_c}) = \boldsymbol{f} \tag{3.3}$$

The interpolation function in Equation 3.1 can be rewritten from its weighted summation format to a linear system of matrices, Equation 3.4. The interpolation matrix,

$$A_{ij} = \phi(\|\boldsymbol{x_i^c} - \boldsymbol{x_j^c}\|), i, j \leq n$$

, is defined element-wise by using the basis function to calculate the variance between all nodes in the domain [17]. From this, the linear system can be solved to calculate the weights, Equation 3.5.

$$\boldsymbol{A\lambda} = \boldsymbol{f} \tag{3.4}$$

$$\boldsymbol{\lambda} = \boldsymbol{A}^{-1}\boldsymbol{f} \tag{3.5}$$

Inversion of the interpolation matrix is an approximately $O(n^3)$ operation with respect to expected run time, where n is the number of centres used. This cubic relation means that RBF systems take significantly longer to calculate as the number of centres increases, and many methods of counteracting this can be found in literature [13].

The nature of the interpolation matrix can be used to determine whether it is invertible or not. If $\boldsymbol{A}$ is not invertible, then the system cannot be solved. If the matrix is strictly positive definite, then $\boldsymbol{A}$ is invertible and there is a unique solution to Equation 3.5.

These equations hold for cases where the radial function used creates a matrix that is positive definite. For some functions defined in Table 3.1, the matrix created is only conditionally positive definite [16]. If $\boldsymbol{A}$ is only conditionally positive, then if a solution exists, it is not guaranteed to be unique. The basis function used determines this property of the matrix. Some basis functions do not guarantee the positive definiteness of $\boldsymbol{A}$. A summary of the equations for which this is the case is given in Table 3.1. In cases where $\boldsymbol{A}$ is conditionally positive definitie, the inclusion of a polynomial term in Equation 3.1 can be used to guarantee uniqueness, resulting in Equation 3.6.

$$s(\boldsymbol{x}) = \sum_{i=0}^{n} \lambda_i \phi(\|\boldsymbol{x} - \boldsymbol{x_c}\|) + h(\boldsymbol{x}) \tag{3.6}$$

The degree of this polynomial depends on the basis function that is used and is also included in the summary in Table 3.1. The coefficients for the polynomial, β, also need to be calculated during the training stage of the interpolation function. This is achieved by introducing a second condition on the system. This condition is that the coefficient vector is orthogonal to the polynomial space, resulting in the condition given in Equation 3.7. This equation must hold for all polynomials, $p(x)$, of degree lower than that of $h(\boldsymbol{x})$ [17].

$$\sum_{i=1}^{n} \lambda_i p(\boldsymbol{x_{ci}}) = 0 \tag{3.7}$$

For a $d = 3$ dimensional space, this condition results in the set of constraints in Equation 3.8. To show this, it is first noted that each constraint is a polynomial itself and so these cases are all the polynomials containing a single term. More complex polynomials can be constructed as linear combinations of these single term equations.

$$\sum \lambda_i = \sum \lambda_i x_{c_i} = \sum \lambda_i y_{c_i} = \sum \lambda_i z_{c_i} = 0 \tag{3.8}$$

Consider the polynomial $p(x) = \beta_0$ where β_0 is a constant. When substituting this polynomial into Equation 3.7, the constant term can be taken out of the summation $\beta_0 \sum_{i=1}^{n} \lambda_i = 0$ and divided throughout to remove, demonstrating the derivation of the first condition. This can be done for other the other single term polynomials to show that the coefficient can be ignored in satisfying this particular part of the constraint. When considering polynomials of two terms or more, similar summation properties can be used to reduce the polynomial down to multiple single term polynomials, reducing to a subset of the same 4 constraints in Equation 3.8. This shows that these constraints are sufficient to meet Equation 3.7.

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{P} \\ \boldsymbol{P^T} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} \tag{3.9}$$

As before, the summation form of Equation 3.4 can be rewritten in matrix form to Equation 3.9. The interpolation matrix $\boldsymbol{A}$, vector of known evaluations $\boldsymbol{f}$, and the weights $\boldsymbol{\lambda}$ are identical to before. The new matrices ensure that the linear system has a unique solution [54]. The matrix $\boldsymbol{P}$ is defined as a block matrix

$$\boldsymbol{P} = \begin{pmatrix} 1 & x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & y_n & z_n \end{pmatrix}$$

where the coordinate values in each row correspond to each of the sources $\boldsymbol{x_c}$.

The first condition is enforced in Equation 3.9 by multiplying the first row to get $\boldsymbol{A\lambda} + \boldsymbol{P\beta} = \boldsymbol{f}$, where each row of the resulting matrix summation matches the interpolation function at each centre, thus recovering the known evaluations. The second condition is enforced by multiplying the second row to get $\boldsymbol{P}^T\boldsymbol{\lambda} = 0$ where the matrix multiplication results in the same calculation as the summation notation used previously.

The weights and polynomial coefficients can be determined by solving the linear system in Equation 3.9. Inversion of the block matrix for this system results in the coefficients calculated as shown in Equation 3.10.

$$\begin{aligned} \boldsymbol{\lambda} &= (A^{-1} - A^{-1}P(P^TA^{-1}P)^{-1}P^TA^{-1})\boldsymbol{f} \\ \boldsymbol{\beta} &= (P^TA^{-1}P)^{-1}P^TA^{-1}\boldsymbol{f} \end{aligned} \tag{3.10}$$

Calculating Equation 3.10 as it is would involve calculating multiple matrix inversions, adding to the computational time required to solve the linear system. Therefore, in practical implementations, the block matrix is usually calculated monolithically, Equation 3.11. The size of the square matrix without the polynomial term is N, while with a polynomial it is $N + 4$. Given that in most systems $N \gg 4$, including polynomial terms does not affect the overall computation time as much as computing multiple inversions. The interpolation function can then be used to determine the ordinate of any uncalculated locations within the domain.

$$\begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{P} \\ \boldsymbol{P^T} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} \tag{3.11}$$

3.1.2 Basis Functions

The basis functions underpinning RBF can be any positive definite radial function. However, a common selection of functions are often used, including the original multiquadric and thin plate spline functions. The basis functions can be categorised as locally or globally supported. The term compact support is also often used interchangeably with local support. Table 3.1 shows the full range of commonly used basis functions, including their categorisation.

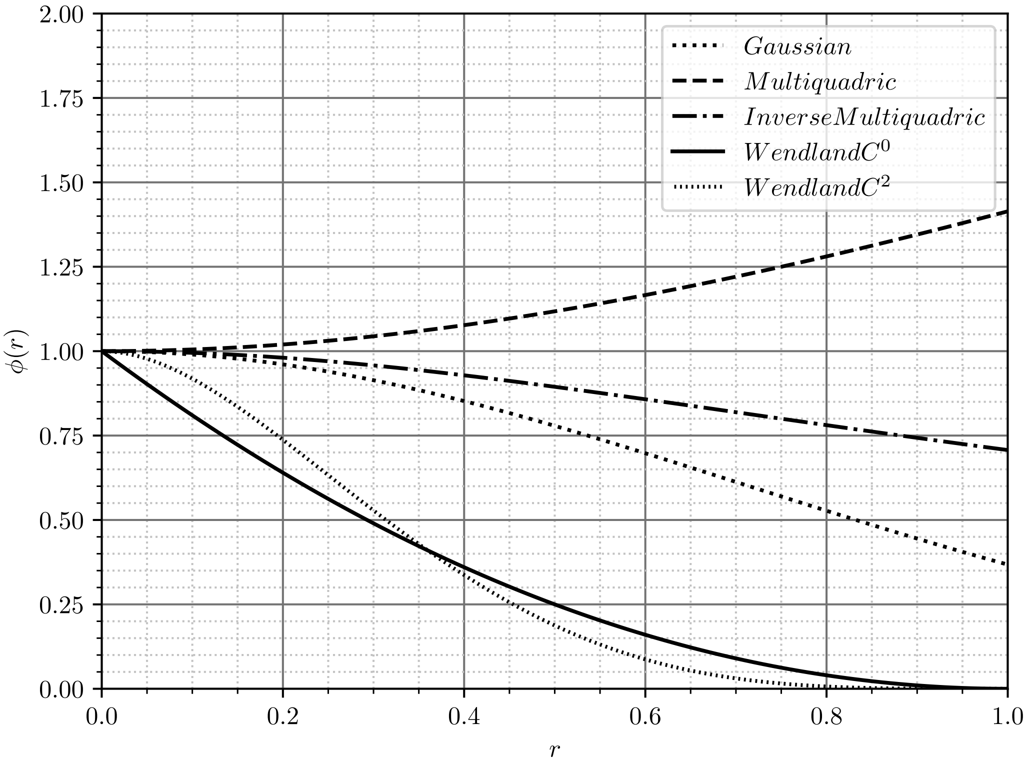


Figure 3.1: Comparison of different basis functions. The Gaussian, Multiquadric and Inverse Multiquadric functions are globally supported. The Wendland functions are locally supported. The shape parameter and support radius for the globally and locally supported functions respectfully is set to 1.

Locally supported functions reduce to zero after a certain distance from the origin. The magnitude of this distance can be controlled by changing the function hyperparameters to eliminate the effect of centres on the interpolation after a certain distance. In contrast, globally supported functions will have an effect on the entire domain; however, in practice, most functions tend to zero and so have a negligible effect after a certain distance from the centre. Figure 3.1 shows some examples of these two types of functions in the domain $r \in [0, 1] \subset \mathbb{R}$. This domain demonstrates how the locally supported functions reduce to 0 where globally supported functions do not.

Table 3.1: Basis functions $\phi(r)$ where $r = \|\boldsymbol{x} - \boldsymbol{x_c}\|$ is the distance of $\boldsymbol{x}$ from the basis function centre $\boldsymbol{x_c}$, ϵ is the shape parameter for globally supported functions and $\eta = r/R$ where R is a support radius for locally supported functions. In some cases, generalised families of functions are given for $\zeta \notin \mathbb{N}$ and $n \in \mathbb{N}$.

Function	**Equation**	**Support**	**Polynomial**
Gaussian	$e^{-(\epsilon r)^2}$	Global	None
Multiquadric (old)	$(\epsilon^2 + r^2)^\zeta$	Global	Degree max(2ζ-1,0)
Multiquadric (new)	$(1 + (\epsilon r)^2)^\zeta$	Global	Degree max(2ζ-1,0)
Inverse Multiquadric (old)	$(\epsilon^2 + r^2)^{-\zeta}$	Global	Degree max(2ζ-1,0)
Inverse Multiquadric (new)	$(1 + (\epsilon r)^2)^{-\zeta}$	Global	Degree max(2ζ-1,0)
Spline/Monomial (n odd)	r^n	Global	Degree n
Thin Plate Spline (n even)	$r^n log(r)$	Global	Degree n
Wendland C^0	$(1-\eta)^2$	Local	None
Wendland C^2	$(1-\eta)^4(4\eta+1)$	Local	None

In the remainder of this section, the global and locally supported functions are discussed in more detail, some specific functions are given their own focus, and function parameters selection is discussed.

3.1.2.1 Globally Supported Functions

Globally supported functions have a small but measurable effect on interpolation throughout the domain. They often come with a shape parameter, ϵ in Table 3.1, which affects the shape of the function.

The $\zeta \notin \mathbb{N}$ parameter in the multiquadric function can be any non-positive integer value, however, is commonly set to 0.5 throughout the literature. In particular, this function is only strictly positive definite for values $\zeta < 0$ [28]. As discussed in Section 3.1.1, this condition is necessary to ensure unique solutions of Equation 3.5. Following this condition, the function is commonly referred to throughout the literature as the inverse multiquadric with $\zeta = -1/2$.

The spline and thin plate spline types derive from the same family of functions originally defined by the French mathematician Duchon [24]. The family of functions is defined over $n \in \mathbb{N}$ where thin plate spline types are valid for even values, $n = 2k : k \in \mathbb{N}$ and spline types are valid for odd values, $n = 2k - 1 : k \in \mathbb{N}$ [17, 13].

3.1.2.2 Compact Support

Of the compactly supported functions presented throughout the literature, the Wendland family of functions is the most widely used [111]. Other families were presented prior to Wendland's proposition. The first was Euclid's Hat, however, this function is not differentiable [35]. Following from this, Wu presented a differentiable function, however, the Fourier transform contained zeros [115, 86]. Wendland's functions do not suffer from these issues and are differentiable polynomials that lead to positive definite radial functions. Some variables within this section use the same nomenclature as other sections for consistency with the literature. For example, the values for β within this section represent local variables not RBF weights.

The Wendland functions are derived from the truncated power function, Equation 3.12, and can be shown to require the conditions that the function $\psi_{\nu,k}$ is positive definite over $\mathbb{R}^d$ and C^{2k} continuous, are satisfied.

$$\phi_a(r) = (1-r)_+^a \tag{3.12}$$

The general formula for $\psi_{\nu,k}$ is given in Equation 3.13 where the coefficient $\beta_{n,k}$ is defined by a recursive formula [111, 17].

$$\psi_{\nu,k}(r) = \sum_{n=0}^{k} \beta_{n,k} r^n \phi_{\nu+2k-n}(r) \tag{3.13}$$

The function ϕ is the positively truncated power function, which is defined such that the result is 0 if $r \leq 1$. This can be scaled by using a support radius to increase the domain to $[0, R]$ by defining $\eta = \frac{r}{R}$ to give the Wendland functions in the general form given in Equation 3.14.

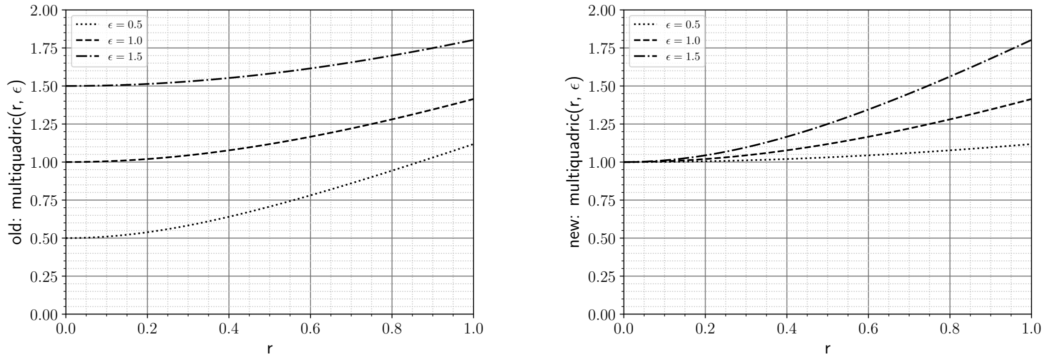
$$\phi(r) = \begin{cases} \psi_{\nu,k}(\eta) & (1-\eta) \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

To satisfy the requirement that the function is positive definite in the domain, note that Equation 3.12 is positive definite in $\mathbb{R}^d$ if $a \geq \lfloor d/2 \rfloor + 1$. Wendland also shows that functions of the form in Equation 3.13 are C^{2k} continuous. The closed form of Wendland's functions can be determined for any dimension d using the relation $\nu = \lfloor d/2 \rfloor + k + 1$, a partial list of which functions is given in [111]. For the cases $d = 2n$ and $d = 2n+1$ there is no change in the Wendland function. The use of RBF in this work is primarily for application in 2- and 3-dimensional spaces. Therefore, the Wendland functions where $d = 3$ are required. The value of $\psi_{\nu,k}(\eta)$ from Equation 3.14 for the continuous cases C^0 and C^2 are included in Table 3.1.

3.1.3 Shape Parameter

The globally supported functions listed in Table 3.1 include a shape parameter, ϵ, which, as the name suggests, controls the shape of the function. Selection of shape parameters can be challenging, and many methods have been suggested to determine appropriate values.

In the original definition of the multiquadric RBF, the constant term of the polynomial is used to control the shape of the function, as shown in row *Multiquadric (old)* from Table 3.1 [82, 79, 18]. In more recent examples, however, the constant term is fixed as 1 and the coefficient of the power term if used as the shape parameter [85, 28, 69]. Comparing the two, it can be seen from Fig 3.2 that the newer definition results in functions that all compute to 1 at the centre, whereas the older definition the evaluation at the centre changes depending on the value of the shape parameter. The newer definition may be preferable in implementations of the method, as this can be equivalently defined by scaling the distance parameter r for the basis function $\phi_c(r) = \phi(\frac{r}{c})$. This treatment can be generalised to other basis functions. By definition, this extends to the inverse multiquadric cases.



(a) Old (b) New

Figure 3.2: Comparison of old and new definitions for the multiquadric basis function.

When applying the shape parameter or support radius to any function, it is common for the parameter to be a single constant applied across all centres; however, this concept has also been extended in the literature to include a scaling parameter at each centre of the [83]. Throughout this work, attention is focused on using a constant shape parameter across all centres.

3.1.4 Shape Parameter Selection

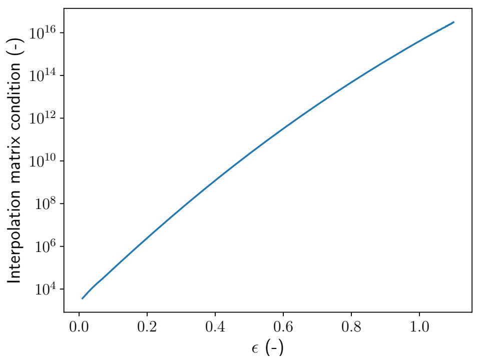
The effect of the shape parameter on the overall interpolation is affected by both the spread of the data and is problem independent [82]. Therefore, there is no single value for ϵ that can be suggested for all problems or basis functions.

One method of determining an appropriate shape parameter for a given set of data is presented by Rippa, where a method similar to Leave-One-Out-Cross-Validation (LOOCV), commonly used in machine learning methods, is introduced [82]. The method is built on the principle of LOOCV and shows that the computational time of the calculation can be reduced by approximating the difference between the actual value and the expected value in each case. It is also noted by Rippa that two previous methods of significance are Hardy's and Franke's methods [41, 32]. Hardy calculates ϵ as $\epsilon = 0.815d$ where d is the mean distance between each data point and its closest neighbour. Franke suggests calculating ϵ as $\epsilon = \frac{1.25D}{\sqrt{N}}$ where D is the diameter of the minimal enclosing circle of the set of centres. Other methods such as those presented by Scheuerer aim to build on these early selection methods, however, no algorithm stands out as significantly better or worse across different numerical applications [87]. Because of this, different methods of determining the choice of ϵ are considered for different applications when used within this thesis.One method of shape parameter selection method introduced by Jakobsson in the application of RBF to mesh morphing is to relate it to the distance between control points [54]. The control points are selected to be a uniform distance, d, apart from each other. The shape parameter is then defined as $\epsilon = \epsilon_i/d$ for a configurable value of ϵ_i. A method similar to that introduced by Jakobsson is explored in this work where the parameter is defined as a multiple of the maximum magnitude of the target value, g. It is also not uncommon for parameter values to be empirically chosen by the user based on their knowledge of the dataset. This will also therefore be considered. A more generic selection method is to use hyperparameter optimisation using bootstrapping. This involves splitting the data set into two, forming a test and train dataset. Different shape parameter values are then used with the training data set to train the network. Once trained, the model is used to predict values for the test data set. The error can be calculated from this to determine how accurate the model is. This method is particularly beneficial as it can avoid issues with over fitting and as it is independent of any other parameters related to the interpolation, can be applied to a range of different cases.

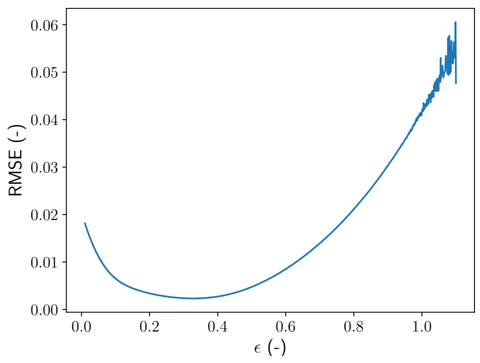
Another property that is of importance to consider when applying RBF functions to mesh morphing is the condition number of the interpolation matrix. For example, The effect of the condition number when using RBF for regression is one of the driving forces behind considerations made by Rippa in his shape parameter calculation method [82]. In matrix inversion problems, such as Equation 3.9 and 3.4, the condition number of the matrix being inverted can be interpreted as giving an indication of the sensitivity of the system. For linear systems where A has a large condition number, then a small perturbation to λ will result in a large change in g. For RBF interpolation problems, An example of the importance of the condition number can be shown by observing the change in condition number and Root Mean Square Error (RMSE) as the shape parameter increases for different basis functions. This is achieved by recreating the comparison given by Rippa, extending to include additional basis functions. The function F_1 in Equation 3.15 is interpolated using RBF using a random sample of 100 training data points. These points are taken from Frankes work [31]. A digitised copy of these values could not be located, and the scan of the publicly available text is of insufficient quality to read. The points were therefore approximated using a plot digitiser and are included in Appendix Appendix A.

$$\begin{aligned} F_1(x,y) = 0.75\text{exp}\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + 0.75\text{exp}\left(-\frac{(9x+1)^2}{49} - \frac{9y+1}{10}\right) \\ +0.5\text{exp}\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - 0.2\text{exp}\left(-(9x-4)^2 - (9y-7)^2\right) \end{aligned} \tag{3.15}$$

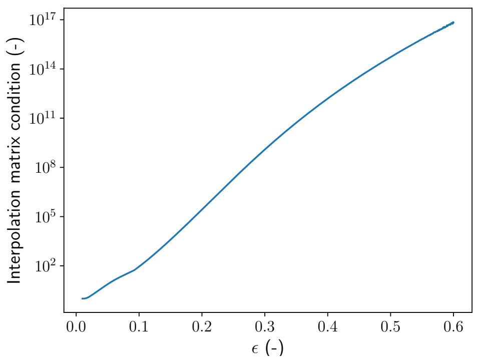
The ill-conditioning of the interpolation matrix is noticeable in the plots for ϵ vs RMSE where the smooth curve transitions into a rough, non-smooth regime. For example, this begins to occur in Fig. 3.3b at approximately $\epsilon = 0.95$, which corresponds to a matrix condition number of the order 10^{14}. The Gaussian function, also a globally supported basis function like the multiquadric, exhibits similar behaviour. For both of these functions, increasing the matrix condition number results in an exponential (or log linear) increase in the condition number. A locally supported function, WendlandC_2, demonstrates a much greater tolerance to the increase in ϵ. This is an important consideration during the selection of the shape parameter. It is not always known apriori if the value selected for ϵ is suitable, which could lead to very large condition numbers if using locally supported functions, resulting in highly ill-conditioned systems. Locally supported functions may not be affected as significantly by this. This is taken into consideration when discussing the performance of different parameters to mesh morphing applications.



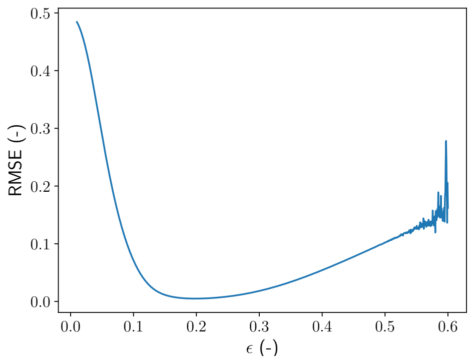
(a) Multiquadric



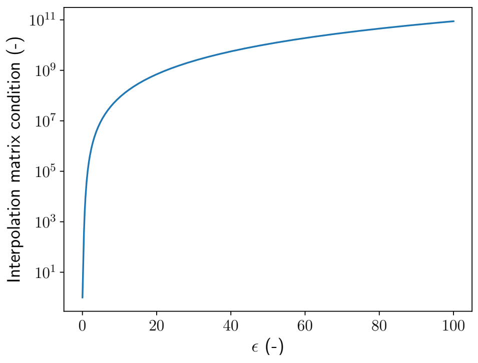
(b) Multiquadric



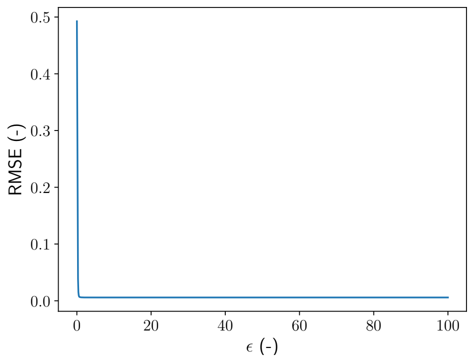
(c) Gaussian



(d) Gaussian



(e) Wendland C_2



(f) Wendland C_2

Figure 3.3: Shape parameter vs condition number and RMSE for the function F_1 using 100 training data points for three different basis functions.

3.1.5 Application to Mesh Morphing

RBF interpolation can be applied to mesh morphing for shape optimisation problems in a number of ways. One common example is through its use in domain element approaches. Here, RBF is used to interpolate the deformation of the control nodes through the domain. For example, the optimisation of a helicopter rotor blade is presented using this approach [5].

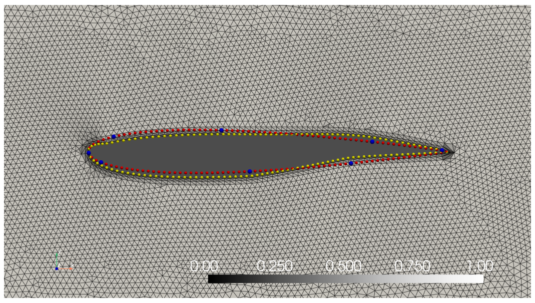
The first application of RBF to mesh morphing was introduced at a similar time in two different works [23, 54]. Both introduce methods where a set of nodes, control points, have a prescribed displacement applied, which is then propagated into the surrounding volume. The method of using RBF interpolation to propagate mesh deformation demonstrated by applying it to a simple 2D test case [23]. The resulting mesh quality is explored by using different basis functions. In particular, the authors highlight that breaking the prescribed displacement into multiple intermediary displacements can improve the quality of the mesh. In a more complex example, a 3D ONERA M6 wing optimised [54]. The wing is parameterised by defining functions for the camber, twist, and thickness of the aerofoil at a distance y along the span of the wing. Each is a function of y and are weighted sums of splines. This provides an exact displacement for the nodes on the surface of the wing. This method provides an added benefit that it reduces the number of source nodes. Time complexity of the matrix inversion is reduced by selecting the centers as a subset of the structural mesh. An algorithm is defined to select the sources from the structural mesh in an approximately uniform distribution of nodes a fixed distance d apart from each other. The analytical displacement is calculated at these points. RBF is then used to interpolate these displacements throughout the mesh, through the rest of the surface mesh and the surrounding volume. The method is compared to Laplacian smoothing and is shown to exhibit minimal difference in solution for significantly reduced computational cost. Nodes can also be constrained by prescribing a zero displacement to maintain their shapes. For example, the symmetry plane that forms a boundary at the root of an ONERA M6 wing is constrained in [54] and the outer boundary of the domain is constrained in [23].

Using RBF for mesh morphing offers many benefits, however there are some limitations that must be handled for industrial scale applications. As already highlighted, one of the biggest challenges to unlocking the full potential of RBF mesh morphing is the time complexity of the matrix inversion performed when calculating the weights, which is of O(n3). This time complexity can be reduced by using techniques such as, a preconditioned iterative solvers to approximate the matrix inversion, space partitioning and the fast multi-pole method [13]. Many of these methods are implemented in a leading commercial RBF solver, "RBF Morph" which has made significant strides in reducing the time complexity of this calculation. The solver can be deployed as a standalone mesh morphing package or as an ANSYS extension and has been used for a range of optimisation cases such as an F1 wing optimisation and motorbike windshield optimisation. A detailed summary of the exact configurations used is however commercially sensitive and not in the public domain [13].

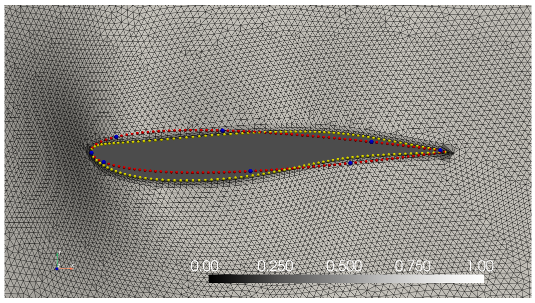
3.1.6 Shape Parameter Effect - Mesh Morphing Examples

The examples shown in Section 3.1.4 are purely academic, modelling mathematical functions. In this section, the effect of the shape parameter and different basis functions on a simple 2D mesh morphing case is explored to identify considerations that should be made when selecting appropriate basis functions and corresponding shape parameters for a problem. The NACA-0012 aerofoil is used with different transformations applied to it.

The first example presented uses the 8 control nodes shown in Fig 3.10 as sources with known translations and the outer boundary included as sources with a known translation of 0. Three different basis functions are used and the value for the shape parameter is varied between 0.1 and 1. Fig 3.4 shows the resulting meshes from 4 of these tests, and Fig 3.5 shows both the mesh quality over this range and the condition number of the interpolation matrix. The morphed boundary of the aerofoil, shown in yellow, between the meshes in Fig 3.4a and Fig 3.4b is different, and is particularly evident in the front half of the aerofoil, where the morphed boundary is further from the original boundary, shown in red. This is due to the increase in condition number for a larger value of the shape parameter. If the condition number is too low, then there is no flexibility in the interpolation matrix to propagate the translation throughout the surrounding mesh. However, it is also clear from Fig 3.5a that as the shape parameter increases, the average mesh quality decreases. Figures 3.4b and 3.4c show a large area of reduced mesh quality in front of the aerofoil that gets significantly worse as the condition number increases. Fig 3.4d represents the mesh from Fig 3.5b for which the minimum mesh quality drops to 0, where some elements are now completely degenerate. These results show the importance of controlling the shape parameter such that it is large enough to allow some flexibility in the mesh propagation but small enough that mesh quality is not significantly reduced.

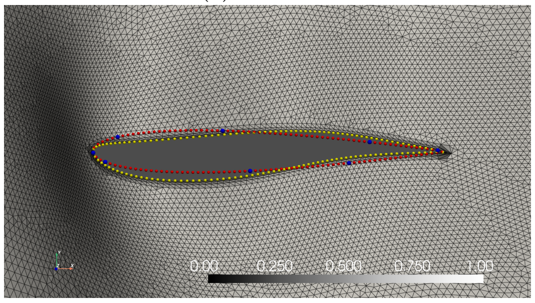


(a) $C = 0.01$

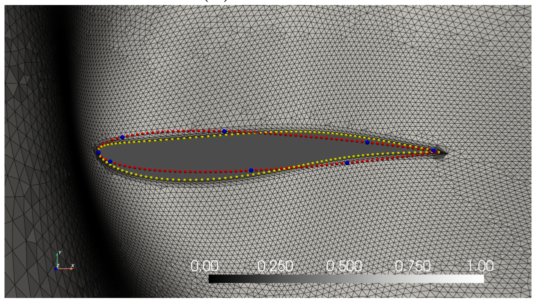


(b) $C = 0.36$

[H]



(c) $C = 0.58$

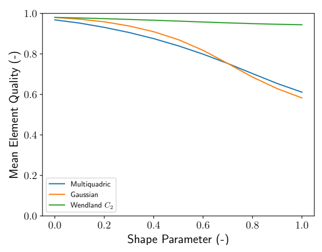


(d) $C = 0.86$

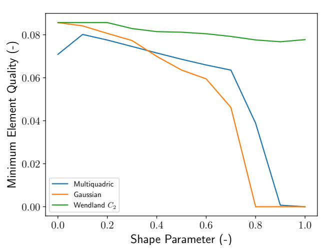
Figure 3.4: NACA-0012 morphed by 8 control nodes (blue) using the Multiquadric basis function for varying values of the shape parameter. Cells are coloured according to their size/shape metric.

Also of note in Fig 3.5 is that using the locally support Wendland C_2 basis function results in a higher mean mesh quality and the minimum mesh quality hold up. When using the globally supported functions, it was noted in Fig 3.4a that a low shape parameter, and therefore condition number resulted in the shape of the surface not being as significantly displaced by the control nodes. This is even more prominent when using the Wendland C_2 function. Fig 3.6a shows the resulting bumps that occur when a small support radius is used.

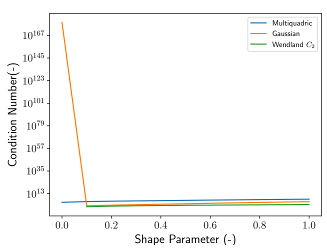
[H]



(a) Mean Quality

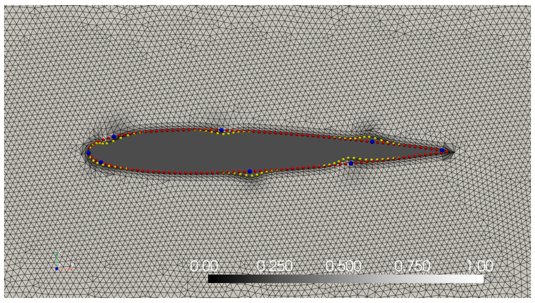


(b) Minimum Quality

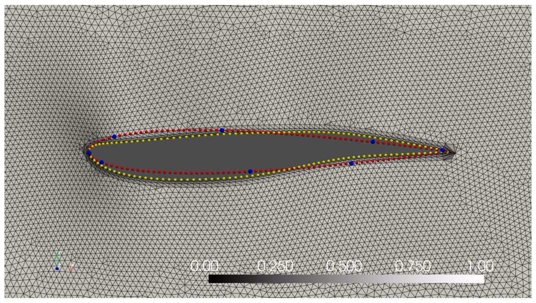


(c) Condition Number

Figure 3.5: Summary of face size/shape quality metrics and condition number of the interpolation matrix for the NACA-0012 morphed by 8 control nodes using three different basis functions and varying values of the shape parameter.

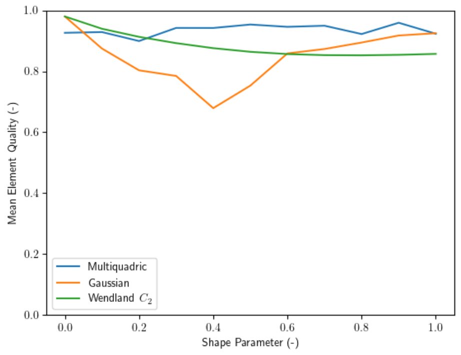


(a) $C = 0.1$

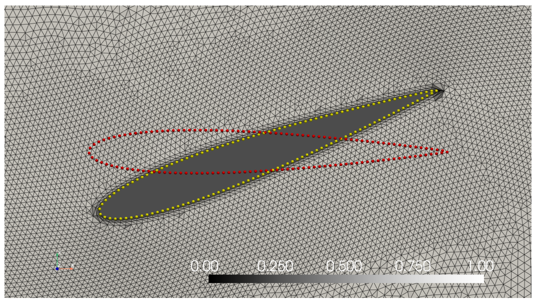


(b) $C = 1$

Figure 3.6: NACA-0012 morphed using 8 control nodes on the surface using the Wendland C_2 basis function and two different support radii. The small support radius results in the translation not spreading across the surface and small bumps appearing around the control nodes.



(a) Mean Quality

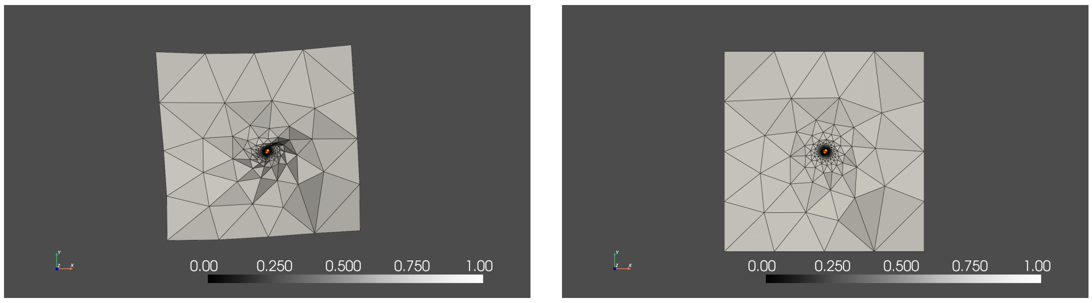


(b) Multiquadric, $C = 0.01$

Figure 3.7: Mean mesh quality for the rotated NACA aerofoil using different basis functions and an example of the resulting mesh, using the multiquadric basis function with a shape parameter of 0.1.

A second case is considered where the aerofoil is instead pitched down 20^o. The translation is propagated throughout the surrounding mesh, again using three basis functions for a range of values for the shape parameter between 0.1 and 1. In this example, the shape parameter value has little effect on the overall quality of the mesh despite significantly increasing the condition number of the interpolation matrix. This demonstrates how different morph cases respond to changes in the shape parameter differently.

Using a global basis function works well for this example when the outer boundary is included. However, if the boundary is removed and a global basis function is used, then propagation of the translation results in undesired motion of the mesh, shown in Fig 3.8. Using a locally supported function where the effect of the basis function reduces to 0 after a certain distance prevents this and offers the local control needed. With this 2D mesh, fixing the boundary to limit this is a suitable option to prevent this issue and used by de Boer et al in their introduction of the use of RBF for mesh morphing [23]. This is mesh dependent and may not always be possible for a given mesh or specific morph. Therefore, the choice of basis function needs to be casefully considered when the network is expected to extrapolate values outside of the domain of the sources used to train the network.



(a) Multiquadric (b) Wendland C_2

Figure 3.8: Comparison between using a globally or locally supported basis function when mesh boundary nodes are not used as sources.

3.1.7 Summary

In this section, mesh morphing using radial basis functions was introduced. The literature on mesh morphing is discussed and some 2D examples were presented that provide information about the effect different parameters have on interpolation. This information is used during the implementation of the mesh morphing framework introduced within this chapter.

3.2 Case Studies

Throughout this chapter, references are made to a number of parameterised case studies to assist in explaining the methodology. Some of these parameterised geometries are also referenced again in other chapters within this thesis to setup the design space for an optimisation problem. Each mesh is introduced, followed by any parameterisations made. In some cases, there are multiple parameterisation definitions.

3.2.1 NACA Aerofoil

Aerofoils are the 2D cross-sectional shapes found when cutting a wing. They are specific geometries designed to control the pressure distribution above and below them to generate a favourable amount of lift. They play a fundamental role in enabling flight and optimisation of this shape is frequently explored. The mesh morphing framework described in this chapter is mainly aimed at enabling optimisation of complex 3D geometries however a simple 2D aerofoil case is useful for analysing the effect of different RBF parameters on the quality of a morphed mesh. NACA aerofoils are a series of aerofoil shapes defined in the 20^{th} century by the precursor to NASA. The aerofoils are subdivided into n-series ($n \subset \mathbb{N}$), in particular, the 4-series has become one of the most widely adopted of these subseries. The 4 digit code for the 4-series is in the format CDPP where the digits represent

- **C** - Maximum camber as % of chord length.
- **D** - Distance of maximum camber from leading edge in increments of 1/10 of chord length.
- **PP** - Maximum thickness of the aerofoil as % of the chord length.

These values can be used along with a set of parametric equations to define a range of aerofoils. The most basic from this set is the NACA-0012, a symmetrical aerofoil with thickness 12%. This airfoil produces zero lift at $\alpha = 0$ and is one of the most well studied of the series used as a baseline design for many different practical and computational tests. A range of different transformations are applied to the geometry and are listed below.

3.2.1.1 Rotation

The airfoil is centered at the point $x = 0.5, y = 0.0$ and all nodes along the top and bottom surfaces are rotated by an angle of α relative to the x-axis about this point. The resulting transformation then needs to be propagated throughout the surrounding mesh. The rotation matrix for this transformation is given in Equation 3.16 where ϕ_x, θ_y and ψ_z are the roll, pitch and yaw angles respectively

This equation represents the full 3-D rotation matrix and is useful when applying transformations to 3-D geometries. In this example ϕ_x and ψ_z are not required and reduce to 0.0.

$$\boldsymbol{R} = \begin{bmatrix} cos(\psi_z)cos(\theta_y) & cos(\psi_z)sin(\theta_y)sin(\phi_x) - sin(\psi_z)cos(\phi_x) & cos(\psi_z)sin(\theta_y)cos(\phi_x) + sin(\psi_z)sin(\phi_x) \\ sin(\psi_z)cos(\theta_y) & sin(\psi_z)sin(\theta_y)sin(\phi_x) + cos(\psi_z)cos(\phi_x) & sin(\psi_z)sin(\theta_y)cos(\phi_x) - cos(\psi_z)sin(\phi_x) \\ -sin(\theta_y) & cos(\theta_y)sin(\phi_x) & cos(\theta_y)cos(\phi_x) \end{bmatrix} \tag{3.16}$$

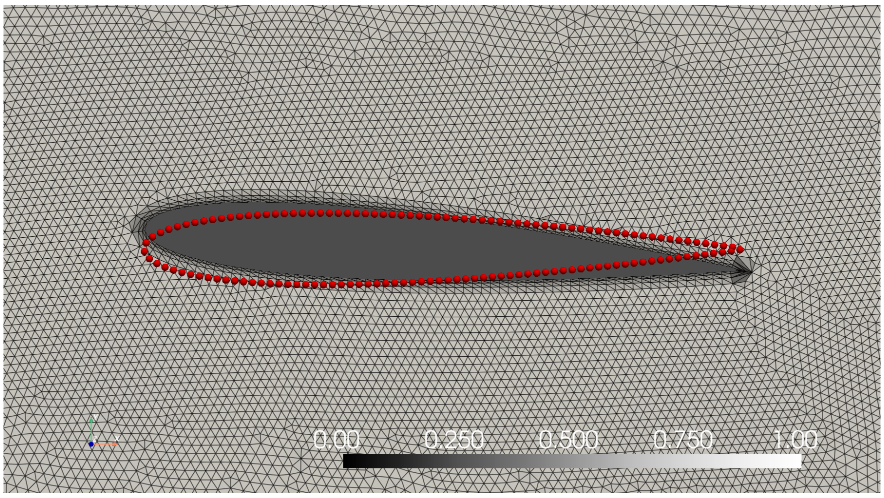


Figure 3.9: NACA-0012 pitched up by $\alpha = \theta_y = 4^o$. Elements are coloured using the $face_{size-skew}$ metric and the original airfoil shape is shown with red points.

3.2.1.2 Control Nodes

Specific nodes across the surface of the geometry can be selected, and a bounding box applied around that node. The node is free to move anywhere within that domain. This then needs to be propagated into the nodes around it. One example of how the boundary smoothness is preserved is the discrete boundary smoothing method implemented by Naumann et al where smoothing terms were introduced to the updated positions of the nodes [70]. For this NACA-0012 example, RBF is used to propagate the changes across the surface of the geometry, before propagating the overall change in the surface of the airfoil into the surrounding volume mesh.

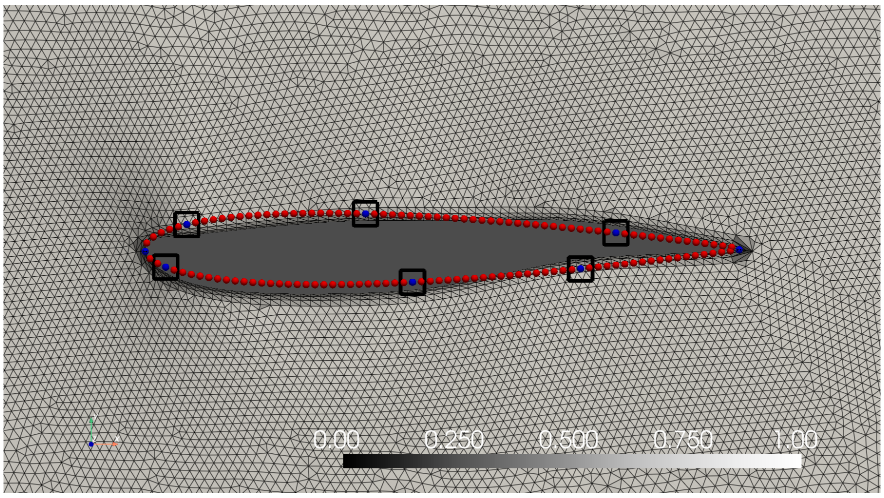


Figure 3.10: NACA-0012 morphed using control nodes. Control nodes are highlighted as blue points with their bounding boxes shown with a black wireframe. The tip and tail are constrained with 0.0 displacement. Elements are coloured using the $face_{size-skew}$ metric and the original airfoil shape is shown with red points.

3.2.2 Sphere

A unit sphere is created by triangulating a spherical CAD body. The baseline mesh, Fig. 6.4a consists of six surfaces, a total of 10,408 triangular elements and 5,206 nodes.

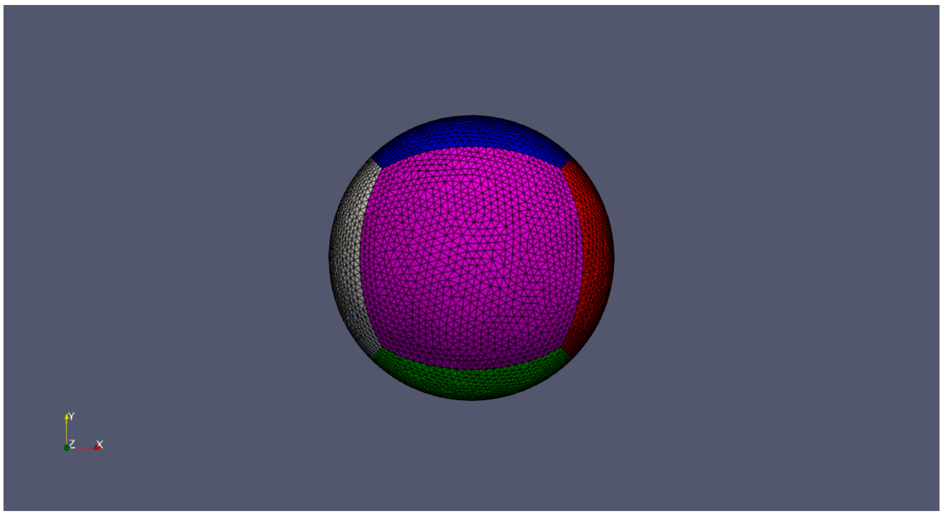


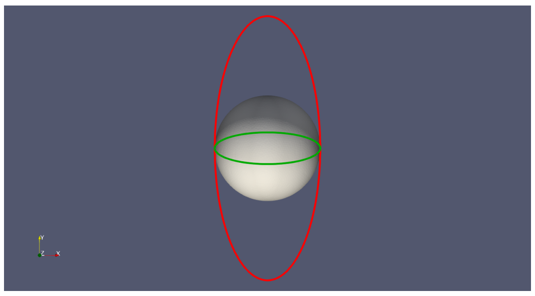
Figure 3.11: Baseline mesh for a unit sphere.

3.2.2.1 Parameterisation

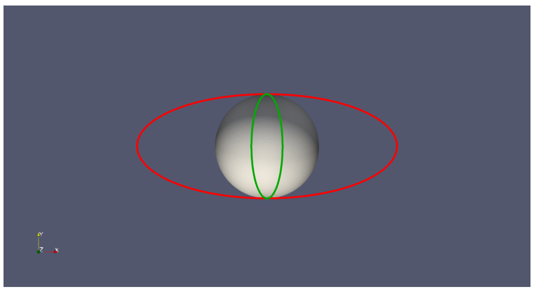
A very simple parameterisation for the sphere is defined using two parameters. The first parameter, $\hat{\boldsymbol{\phi}}_{\mathbf{1}}$, stretches the sphere radially in the y-z plane, maintaining the length of the sphere. The second, $\hat{\boldsymbol{\phi}}_{\mathbf{2}}$, compresses and elongates the length of the sphere along the x axis. A summary of these is given in Table 3.2 including the min and max values for each parameter and the mathematical definition of their transformations. The effects of the minimum and maximum bounds of each parameter is shown in Fig. 3.12.

Table 3.2: Parameterisation information for the simple sphere parameterisation. The transformation matrix, $\boldsymbol{T}$, is applied to each node in the sphere by taking its dot product with the coordinate of the node $\boldsymbol{T} \cdot \boldsymbol{x}$.

Name	Parameter	Min	Max	Transformation Matrix ($\boldsymbol{T}$)
Stretch Radius	$\hat{\phi}_1$	0.3	2.5	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \phi_1 & 0 \\ 0 & 0 & \hat{\phi}_1 \end{bmatrix}$
Stretch Length	$\hat{\phi}_2$	0.3	2.5	$\begin{bmatrix} \hat{\phi}_2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



(a) Width

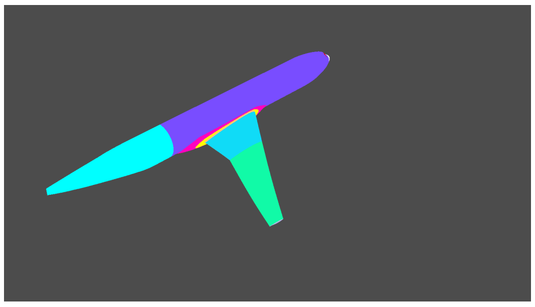


(b) Length

Figure 3.12: Min and max values for the simple sphere parameterisation.

3.2.3 NASA CRM

Benchmark cases are often important to the development of optimisation cases to ensure engineers know that their algorithm has returned optimal results. Simple functions are easy to come by in many optimisation cases, however, in cases where CFD is used in the cost function evaluation, these case studies are significantly more limited. This is due to the large number of different parameters possible (eg Mach number, angle of incidence, etc.) and also the complex nature of air flow at many regimes means no analytically known optimum exists. One commonly used reference model for bench mark cases is the NASA Common Research Model (CRM). This simple wing-body geometry has been studied in detail both numerically and experimentally. For the purposes of this chapter, this provides a basic example of a complex body. The geometry of the body is shown in Fig. 3.13, which shows the surface definitions and sections of grouped surfaces. For this geometry, the parameter to be explored is the fuselage length, compressing, and stretching within the limits shown in Fig. 3.14.


(a) Surfaces

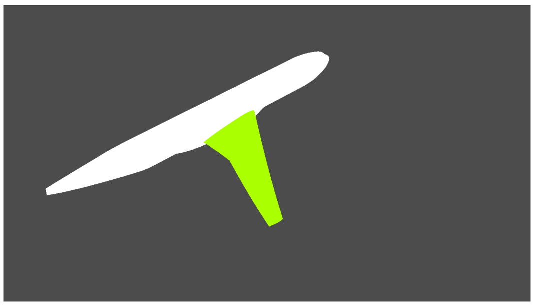

(b) Sections

Figure 3.13: Comparison between the NASA CRM surfaces and the sections these surfaces are grouped into.

The baseline mesh is a half-body that cuts the xz plane along the length of the fuselage. The farfield forms a hemisphere of radius of approximately 420 units, with the flat face forming the symmetry plane for the mesh. The hybrid mesh is made up of 22 surfaces (19 for the body, 1 for the hemispherical component of the farfield and 2 for the flat symmetry plane). Table 3.3 summarises the type and distribution of mesh elements across these surfaces.

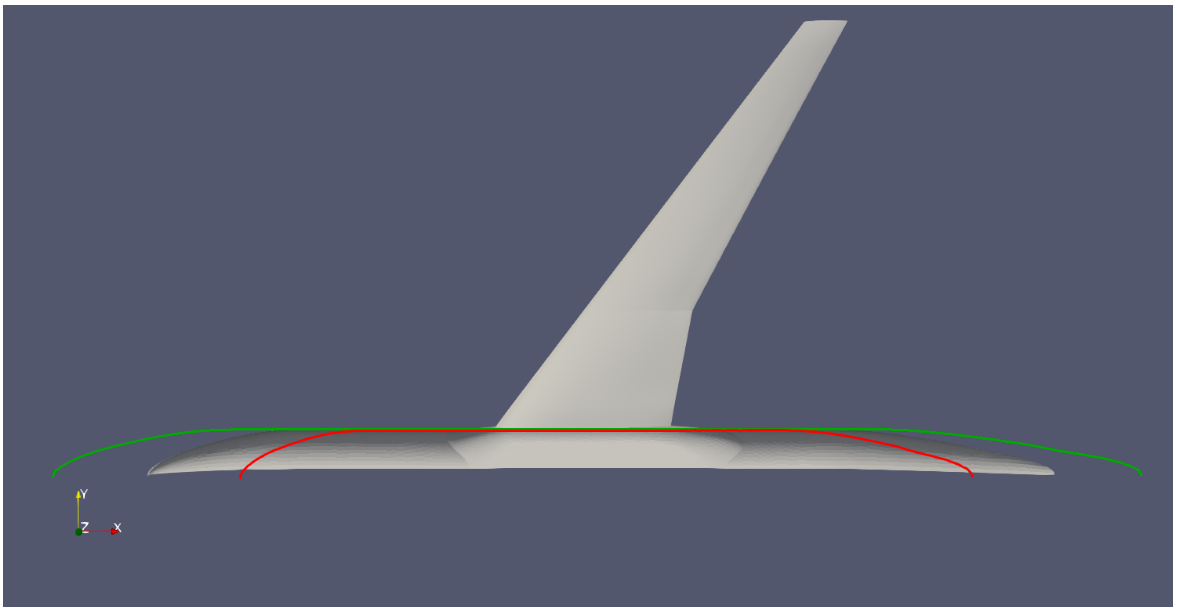


Figure 3.14: Examples of extreme fuselage stretches for the NASA CRM.

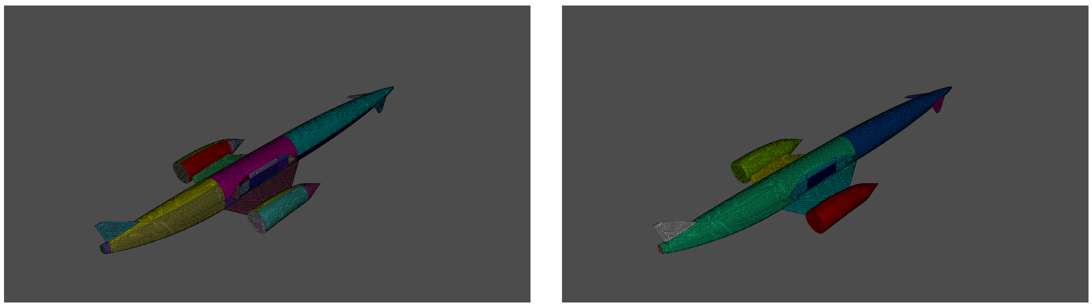
Table 3.3: Breakdown of mesh element location within the NASA CRM mesh.

Region	Nodes	Elements	
		Triangular	Quadrilateral
Total	220,759	206,232	117,641
Geometry	189,938	185,336	96,985
Farfield	142	258	0
Symmetry Plane	30,679	20,638	20,656

3.2.4 Skylon

The Skylon spaceplane introduced in Section 1.3 is the main geometry used for comparing optimisation methods throughout this work. The baseline geometry, introduced in Section 2.3.4 consists of a complete, symmetric model of the Skylon C1 geometry with a spherical farfield, radius 1000. The surface mesh of the main geometry is show in Fig 2.15. The surface mesh is a triangulation with 83 surfaces (79 on the geometry, 4 for the farfield), a total of 59,492 elements (56,200 on the geometry, 3,292 for the farfield), and 32,533 nodes (30,743 on the geometry, 1,790 for the farfield).

Some of the surfaces are grouped together to make parameterisation simpler, shown in Fig 3.15. A more detailed summary of how the surfaces have been grouped including the names used to reference each section is given in Table 3.4.



(a) Surfaces

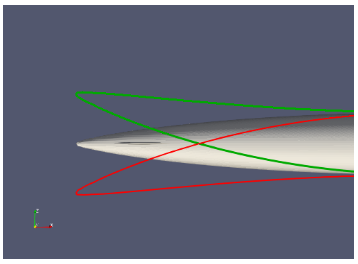
(b) Sections

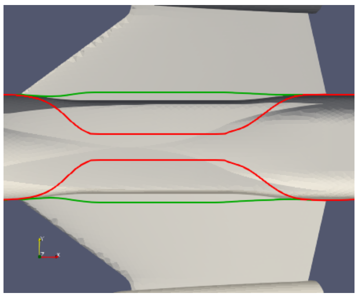
Figure 3.15: Comparison between the Skylon surfaces and sections these surfaces are grouped into.

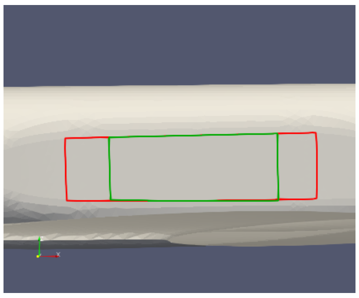
Table 3.4: Breakdown of section definitions for Skylon. Some of the sections are symmetric across the x-z plane (i.e. wing, canard, and engine) and these are treated as two independent sections. The colours of examples of each section that are visible in Fig. 3.15b are also highlighted.

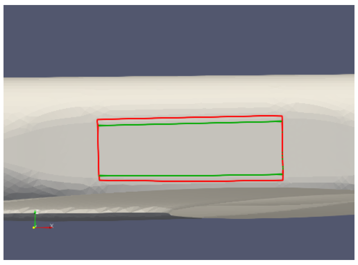
Section	Symmetric	Colours in Fig. 3.15b
Fin	No	White
Engine	Yes	Red/Light Green
Canard	Yes	Pink/Purple
Wing	Yes	Yellow/Turquoise
Central Mid Panel	Yes	Dark Blue
Mid Panel	Yes	Light Blue
Fuselage Front	No	Blue
Fuselage	No	Green
ACS	No	Orange

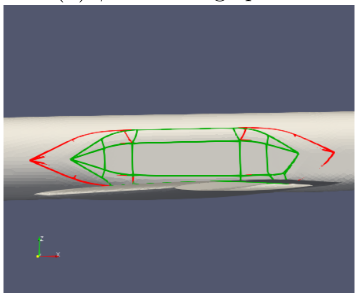
Based on the flow features of the initial aerodynamic analysis of the geometry at $M = 1.2$ performed in Section 2.3.4, it was suggested that the midsection of the fuselage was an important region to target during optimisation. The original parameterisation used to define the waisted mid section is not publicly available in any literature meaning this exact parameterisation cannot be leveraged or explored directly. However, optimisation in the region of the waisted section will likely have a positive affect on the overall performance due to the placing of the nacelles and wings relative to the mid section of the fuselage. Therefore, the geometry is parameterised as shown in Fig. 3.16 with an example of the bounds for each parameter shown. The wings and control surfaces were not explored as these have already been carefully sized to meet a range of multidisciplinary requirements by the developing company REL.


(a) ϕ_1 - Nose Droop


(b) ϕ_2 - Fuselage pinch


(c) ϕ_3 - Central panel stretch (x)


(d) ϕ_4 - Central panel stretch (z)


(e) ϕ_5 - Central section stretch (x)

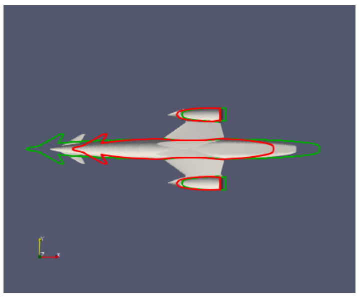

(f) ϕ_c - Fuselage stretch

Figure 3.16: Bounds for parameterised Skylon geometry. 3.16a - 3.16e shows the bounds for independent parameters ϕ_1 - ϕ_5. The dependent parameter, ϕ_c, used to constrain the volume is shown in 3.16f.

Parameter ϕ_1 - Nose Droop

The first parameter chosen was to droop the nose. To achieve this, a Bézier curve is used to define a vertical translation of the *fuselage front* section. The curve was constructed with 3 control points - one at the tip, one at the base where the front of the fuselage connects with the rear, and one directly at the midpoint between these two points. All three of these control points are shown in Fig. 3.17. The tip is then displaced by a constant value ϕ_1 and the translation acts along the z axis of each coordinate in the fuselage. Fig 3.17 provides some examples of the shapes of Bézier curve that can be constructed following this setup and an outline of the overall effect this has on the front of the fuselage given in Fig 3.16a.

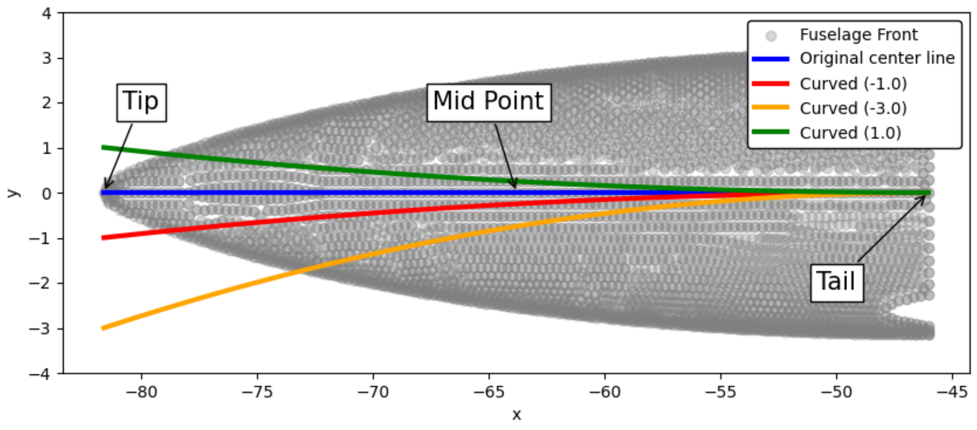


Figure 3.17: Example of the displacement from a Bézier curve along the front of Skylon's fuselage.

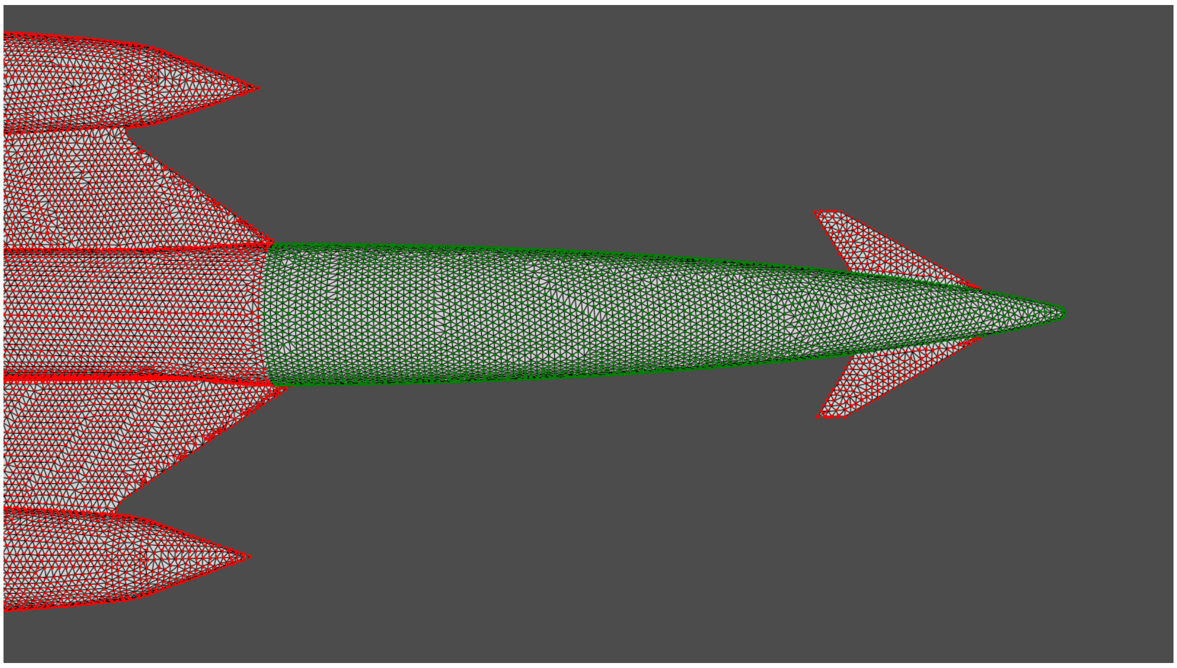


Figure 3.18: Highlighted sections of classified nodes for ϕ_1.

One important consideration here is that the canards at the front of the nose should retain their angle relative to the flow. Changing this angle would cause the aerodynamic coefficient contributions from these surfaces to change. The rotation matrix from Equation **??** needs to be applied here to ensure the canards maintain their angle as the nose droops down.

Parameter ϕ_2 - Fuselage pinch

The second parameter was to pinch the *central mid panel* in and out, normal to the internal payload bay. The Fig. 3.16b shows the extreme values this parameter could take. It is noted that from a business perspective, this parameter can have a detrimental effect on the payload bay sizing and so it's effect should be limited. It is included within the study to academically explore it's effect on performance.

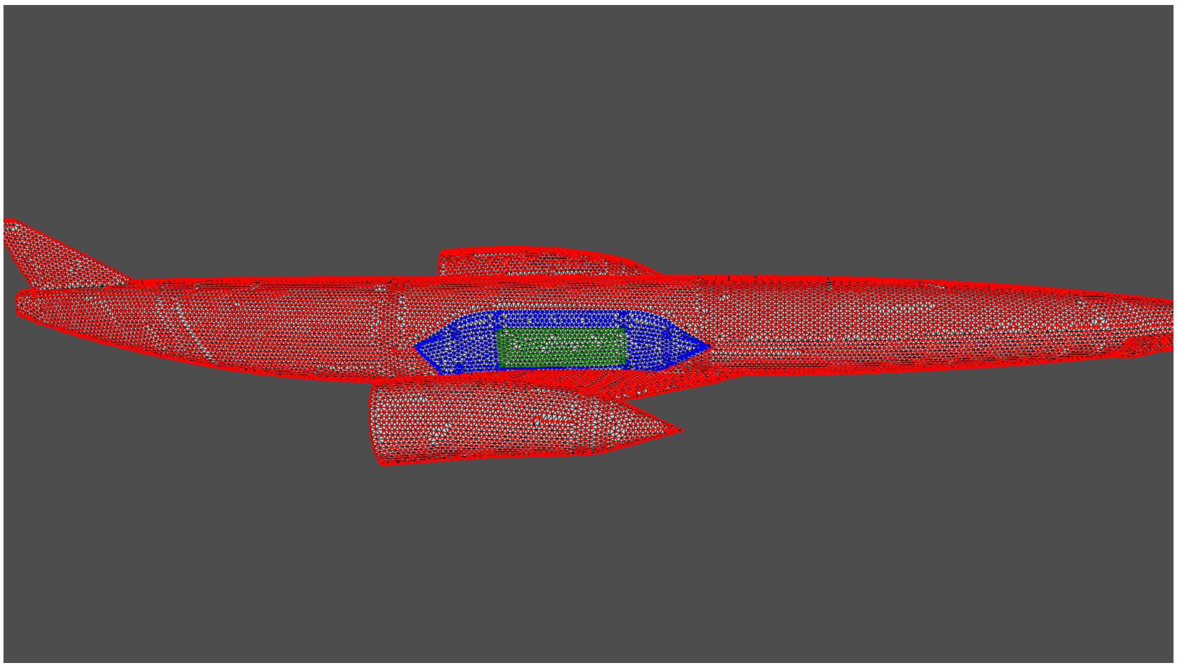


Figure 3.19: Highlighted sections of classified nodes for ϕ_2, ϕ_3 and ϕ_4. The green nodes are the target nodes to be transformed, the blue nodes may then move as a result of this transformation. The red nodes should not be affected by any transformation.

The rest of the waisted section can be translated with the central panel to avoid surface discontinuities; other sections such as wings and fuselage should remain unchanged. Fig. 3.19 highlights all of these regions.

Parameter ϕ_3 - Central panel stretch (x)

The third parameter considered is the length of this central panel along the x axis. This parameter does not affect the payload bay size and allows for increased wasting of the central section. The bounds for this parameter are shown in Fig. 3.16c and the node classification is the same as that for ϕ_2, shown in Fig. 3.19.

Parameter ϕ_4 - Central panel stretch (z)

The fourth parameter is similar to ϕ_3 but targets the height of the central panel along the z axis. Again, this has no effect on the payload bay size and increases the potential waist of the central section. The bounds for this parameter are shown in Fig. 3.16d and the node classification is the same as that for ϕ_2, shown in Fig. 3.19.

Parameter ϕ_5 - Central section stretch

The fifth parameter is the length of the wasted section surrounding the central pinch panel. This allows the effect of the wasted section to expand into the surrounding fuselage without affecting the payload bay. To ensure that the central panel shape is controlled completely by the other parameters, it's shape is constrained for this parameter. The limits for this parameter are shown in Fig. 3.16e, and node classification is shown in Fig. 3.20.

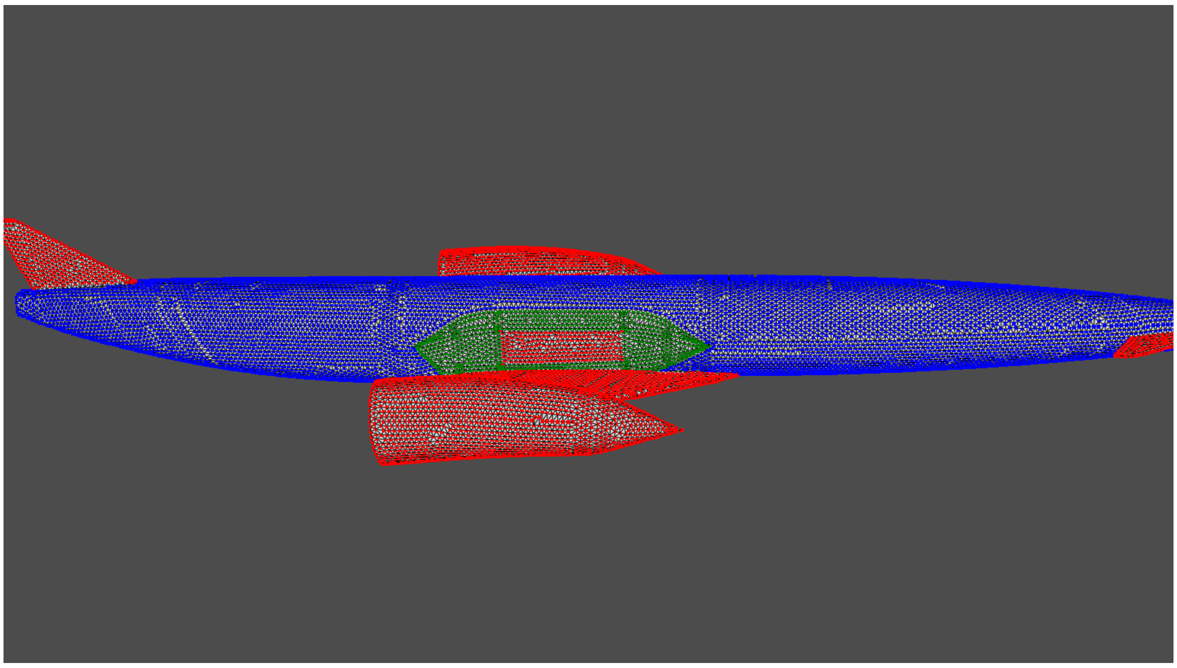


Figure 3.20: Highlighted sections of classified nodes for ϕ_5.

Parameter ϕ_c - Fuselage stretch

The final parameter defined is the length of the fuselage. The subscript c is used for this parameter because it is used as a repair operator during the optimisation cases to constrain the volume of the geometry. All sections of the spaceplane (wings, fin, canards) should remain fixed in shape, though they may move relative to one another. That is, the absolute distance between the canard and the wing may change, but their relative locations along the fuselage should not. The limits for this parameter are shown in Fig 3.16f and the node classification in Fig 3.21. Note here that, similarly to ϕ_1, nodes are only classified as target or constrained nodes in this example.

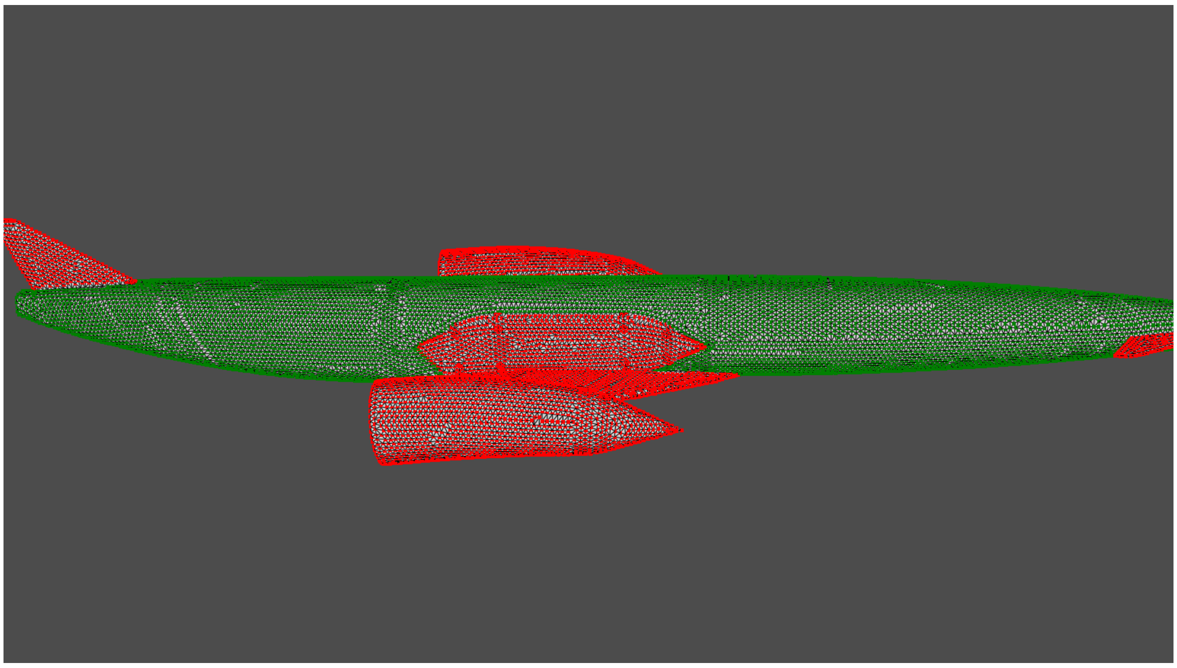


Figure 3.21: Highlighted sections of classified nodes for ϕ_6.

3.2.4.1 Parameter Bounds

Bounds for the parameters are decided by completing a mesh quality study. The volume meshing tool within FLITE3D has built-in mesh quality checks that determine if the mesh is of suitable quality to be solved over. The value of each parameter is increased until the volume mesh is not generated over a morphed surface mesh. These bounds are then qualitatively analysed to determine whether they are suitable or need further constraining. Table 3.5 shows how far each parameter can be pushed based on the quality of the mesh and the constrained values from the qualitative analysis.

Table 3.5: Skylon parameter bounds. Each parameter is increased and decreased until volume mesh generation failed, setting the absolute bounds. These bounds are then manually constricted to set the bounds for the optimisation study.

Parameter	Volume Mesh Failed		User Defined Limit	
	Min	Max	Min	Max
ϕ_1	-10	10	-5	5
ϕ_2	-2.5	1.5	-2	0.5
ϕ_3	0.5	1.3	0.8	1.2
ϕ_4	0.5	1.2	0.9	1.1
ϕ_5	0.5	1.3	0.9	1.2
ϕ_c	0.6	1.4	N/A	N/A

3.3 Mesh Morphing Methodology

The novel implementation of the mesh morphing approach is now described in more detail. The method handles boundary detection and uses the RBF centres to constrain the curves between surfaces. Hyper-parameter tuning is also discussed, as well as which basis functions provide better quality meshes between some different use cases. The methodology also includes geometric constraint handling where properties such as geometry volume can be constrained. This is an essential requirement for many multidisciplinary use cases in industry.

To set up the definition for a morph, all information about the morph is collated into a single folder defined as the *morph directory.* The geometries surface nodes are first grouped into subgroups of target nodes (T), unconstrained (free) nodes (U) and constrained nodes (C). The set of target nodes, T, are those to which a transformation function will be applied. When these nodes are morphed, the elements surrounding these nodes must not invert or intersect. To avoid this, the transformation should be propagated into the surrounding elements where appropriate. These nodes can be explicitly defined as the set of unconstrained nodes U, and after translating the set T, the transformation will propagate to U to avoid loss of element quality and maintain surface continuity. In some engineering applications, some surfaces or geometrical shapes should remain invariant during transformation. For example, during fuselage optimisation, the wing should remain unchanged. This can be achieved by specifying a set of constrained nodes, C. The spatial position of the constrained set C may vary, but the geometric shape and topology will remain consistent. For example, in the case of the aircraft fuselage optimisation case, the fuselage may thin, meaning the wings have to move inwards with the fuselage, however, the shape of the boundary will remain exactly the same.

A further constraint to be considered is that the orientation of the constrained shapes should remain invariant. Angle of attack relative to the free stream is a key simulation parameter in aerodynamic applications that has a significant impact on the simulation results. In the simple fuselage optimisation case, it may be preferable to modify the pitch of the nose or tail, but if the wings chord orientation from the free stream was changed then this would have a significant impact of the lift and drag calculations of the geometry. This highlights the importance of keeping the shape and orientation consistent. The aircraft optimisation study could be extended to consider the impact of wing shape and orientation; however, this should be handled separately as a second parameter.

The methodology can be divided into 4 main stages:

1. Node classification into T, U and C.
2. Initial translation.
3. Recover the constrained boundaries.
4. Propagate to any surrounding unconstrained nodes.

Throughout the course of this research, the implementation of each of these stages evolved, and in the remainder of this section, the implementation history of each of these stages is discussed.

3.3.1 Step 1. Node Classification

The simplest method of classifying large numbers of nodes is to group their surface id's. Examples of this can be seen in Fig. 3.15b and Fig. 3.13 where the discretised surfaces based on the original CAD definition of the Skylon and CRM geometries have been grouped into sections. These labelled sections can be grouped depending on features the morph wants to target. For example, the entire fuselage can be considered as a single group section, or the front half of the fuselage can be isolated.

The major limitation of this method is that the user is limited to applying transformations to sets of nodes based on the original surface definitions from the CAD defined geometry. This may not be the most appropriate method of selecting nodes for each application. For example, consider a wing where the wing is defined using chord wise panels but the shape of the trailing edge is under investigation; the user may want to translate a subset of the chord wise nodes while constraining others. Flexibility of node selection is one of the key benefits that is enabled using RBF as highlighted in Section 3.1.

The node selection method was therefore allows for the selection of individual nodes and faces, resulting in a flexible selection of nodes to be categorised. For example, in Fig. 3.10, individual nodes are required to be selected to select control nodes. Selection of nodes in this definition are however still limited to the selection of nodes that exist on the surface, and therefore nodes generated during the surface mesh generation. The geometry could be constructed to ensure nodes exist in areas of interest by using sources to refine certain areas, by defining specific curves in the CAD geometry.

Implementing node classification is achieved through a Python class called *CreateMorph* from within the Paraview Python python interpreter [8].

Given that all nodes must be classified into one of the sets T, U and C, it is sufficient to specify T and U, then the remaining nodes are constrained. Class methods from *CreateMorph* can be used to label surfaces, faces or individual nodes as either part of set T or U based on what is currently selected in the viewing window. These are then saved to a file in the morph directory. The sets are then constructed as the union of all nodes encompassed by the selected surfaces, faces and nodes respectively.

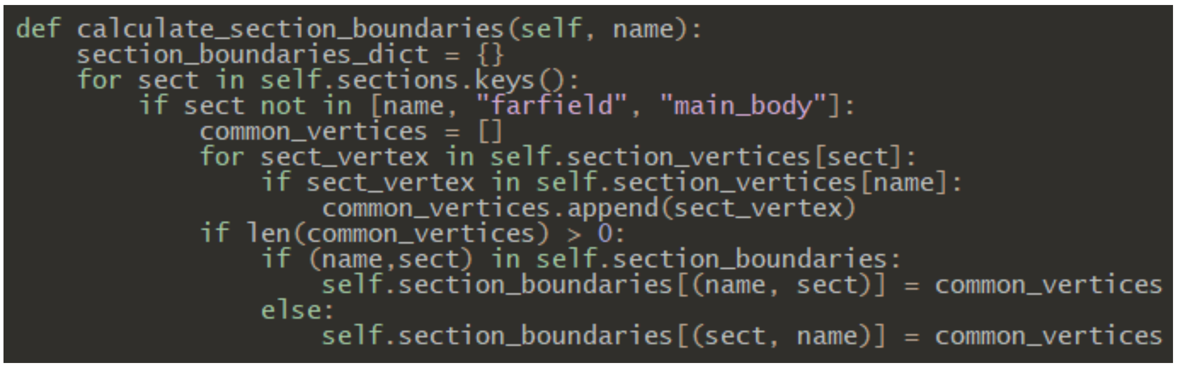
Once the nodes have been selected, the boundaries between these sets needs to be identified. *Boundary detection* refers to the process of using an algorithm to determine where the boundary between different sets of nodes is based on their classification. This ensures that after applying the initial translation, other algorithms know which nodes lay on the boundary between sets and so may need correcting.

3.3.1.1 Boundary Detection - Between Sections

The initial implementation for this determined the boundaries between the sections that are defined by sets of surfaces, for example those shown in Fig 3.15. Boundary detection was especially important here as some surface edges would be located within enclosed regions while others would be on boundaries between one or more classifications.

A single morph configuration may contain multiple closed boundaries. Consider the case of an aircraft geometry such as the Skylon spaceplane. Each canard forms a closed boundary with the fuselage. If the fuselage width decreases, bringing the canards closer to the center line, then the canards have to be pulled in towards the center line, and each boundary must be corrected by different quantities. One fuselage-canard boundary must be displaced by a negative amount while the other by a positive amount.

The constrained boundaries were detected by first classifying all nodes as explained in Section 3.3.1, classifying them by sections only, not individually or by surface. The boundaries between these sections are then identified. The section classifications are then used to determine which sets out of T,U, or C the each boundary is between.

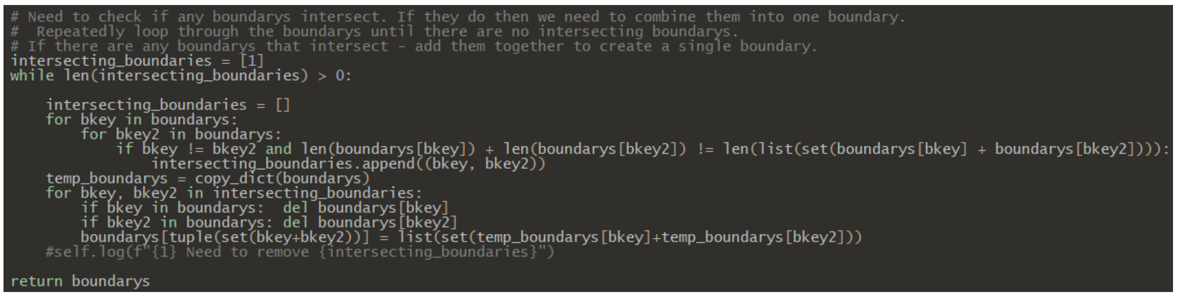


```python
def calculate_section_boundaries(self, name):
    section_boundaries_dict = {}
    for sect in self.sections.keys():
        if sect not in [name, "farfield", "main_body"]:
            common_vertices = []
            for sect_vertex in self.section_vertices[sect]:
                if sect_vertex in self.section_vertices[name]:
                    common_vertices.append(sect_vertex)
            if len(common_vertices) > 0:
                if (name,sect) in self.section_boundaries:
                    self.section_boundaries[(name, sect)] = common_vertices
                else:
                    self.section_boundaries[(sect, name)] = common_vertices
```

Figure 3.22: Code snippet for Python function used to determine interfaces between section.

Identification of boundaries between sets of nodes is fairly trivial and was implemented by looping through all nodes in a section, and checking if they were also present in another section. The implementation for this is shown in Fig 3.22. Within this code snippet, there are multiple areas for potential code optimisation in both the poor speed performance and readability of this implementation. More importantly however, a fundamental property of the resulting dictionary is that the order with which section pairs are added is dependent on the order with which the sections are defined. Not only this but the order of dictionaries is not always deterministic across all python versions. These properties will be important when discussing a subtle flaw in the overall implementation.

Once all section boundaries are been identified, the second step was to identify boundaries between different node classifications. The implementation of this step is again a trivial question of comparing if two sections have any intersection. Once these boundaries are determined, any intersections between boundaries need to be identified and combine them to close of the region. This implementation is shown in Fig 3.23. In most cases, this implementation worked without fault however it was noticed that on some rare occasions, the morphing process was hanging during this stage.



```python
# Need to check if any boundarys intersect. If they do then we need to combine them into one boundary.
#  Repeatedly loop through the boundarys until there are no intersecting boundarys.
# If there are any boundarys that intersect - add them together to create a single boundary.
intersecting_boundaries = [1]
while len(intersecting_boundaries) > 0:

    intersecting_boundaries = []
    for bkey in boundarys:
        for bkey2 in boundarys:
            if bkey != bkey2 and len(boundarys[bkey]) + len(boundarys[bkey2]) != len(list(set(boundarys[bkey] + boundarys[bkey2]))):
                intersecting_boundaries.append((bkey, bkey2))
    temp_boundarys = copy_dict(boundarys)
    for bkey, bkey2 in intersecting_boundaries:
        if bkey in boundarys:  del boundarys[bkey]
        if bkey2 in boundarys: del boundarys[bkey2]
        boundarys[tuple(set(bkey+bkey2))] = list(set(temp_boundarys[bkey]+temp_boundarys[bkey2]))
    #self.log(f"{1} Need to remove {intersecting_boundaries}")

return boundarys
```

Figure 3.23: Code snippet for python function used to identify independent boundaries.

On occasions, it was noted that the *while* loop was not exiting in edge cases where exit conditions were not possible. This bug was an edge case as a result of the non-determinism of python dictionaries previously highlighted. A different python version was used on the remote "production server" compared to the local development environment. As such, the issue only manifested in production and as the problem was non-deterministic, a simple program restart often fixed the issue.

This was occurring because the dictionary order is deterministic in later versions of Python, but not in some older versions. This does not resolve the issue as the order of *bkey* values in the variable *boundarys* is still determined by the order sections are defined in, however explained the sporadic nature of the bug. Therefore, there is always potential for certain definition orders to reproduce this bug. It was identified that this implementation was insufficient for long term code usability.

3.3.1.2 Boundary Detection - Between Sets of Nodes

The main limitations of the previous implementation in detecting boundaries were that it was reliant on the pre-computed section definitions and its outcome was dependent on the order in which sections were defined. The implementation should therefore have no dependence on the section definitions. In particular, the *set* datatype within the python standard library is a useful tool not utilized previously. This datatype can be used to find unions and intersections between sets of nodes at significantly reduced computational cost with increased code readability.

Algorithm 1 presents a function that can be used to identify the boundary between two sets of nodes within a mesh when given the connectivity matrix of the mesh. Within this function, the list of nodes connected to any given node is pre-computed and referenced to speed up the algorithm. This is used to determine which nodes from $T \cup U$ are connected to constrained nodes, C, as well as for all 3 possible combinations of these sets.

Algorithm 1 function: GetBoundaryNodes. The framework allows users to select a subset of nodes from the mesh to be morphed/constrained. This functions identifies the nodes from a given set, N, that lay on the boundary with another set E. The connectivity matrix for the mesh is used to determine which nodes are connected to a given node.

```
1: function $GetBoundaryNodes$(N, E)
2:     $allBoundaryNodes \leftarrow$ empty list
3:     for each node $n \in N$ do
4:         $cNodes \leftarrow$ set of all nodes connected to $n$
5:         if $cNodes \cap E \neq \emptyset$ then
6:             $allBoundaryNodes.insert(n)$
7:         end if
8:     end for
9:     return $allBoundaryNodes$
10: end function
```

The independent boundaries can be determined from this, along with the node connectivity by building up and linking chains of nodes. While this is trivial to visually identify by a user, it is not as trivial to programatically identify. The connectivity matrix for the mesh is used in Algorithm 2 to identify sets of nodes connected to one another.

A list of independent boundaries is initialised and the loop begins by selecting any node at random to form the initial chain.

The algorithm then loops through each node within $T \cup U$ and checks if it is connected to any of the independent boundaries. If it is, append it to the list, if not, then create a new independent boundary. As the number of independent boundaries increases, a node may appear in 2 or more chains. If this is the case then these chains are linked by their common node and can be merged. Once this process is completed for all nodes in $T \cup U$, then a list of nodes for each independent boundary will have been compiled. This method returns the same result irrespective of the starting node or the order with which nodes are investigated.

This method returns the same result as the previously described boundary detection method for the same section definition, but removes the issue of edge cases not always terminating. The only metadata about nodes that is required for this method is the connectivity and thus it can also be used when node classification is achieved through manually selecting individual nodes rather than relying

Algorithm 2 function: GetIndepBoundaries. This function takes two sets of nodes from a mesh, and identifies the sets of independent boundaries between them. The boundary nodes are a subset of one of the sets, those which are connected to the second set of nodes, which are completely excluded from the sets of independent boundaries.

```
1: function $GetIndepBoundaries$(N, E)
2:     $numBoundaries \leftarrow 0$
3:     $indepBoundaries \leftarrow$ empty map
4:     $allBoundaryNodes \leftarrow$ call $GetBoundaryNodes(N, E)$
5:     for $\forall n \in allBoundaryNodes$ do
6:         $boundariesContainingN \leftarrow$ empty list
7:         for $\forall m$ connected to $n$ do
8:             for each $key, boundary$ in $indepBoundaries$ do
9:                 if $m \in boundary$ & $n \notin boundary$ then
10:                    $boundary.insert(n)$
11:                    $boundariesContainingN.insert(key)$
12:                end if
13:            end for
14:        end for
15:        if size of $boundariesContainingN == 0$ then
16:            $numBoundaries$ += 1
17:            $indepBoundaries[numBoundaries] \leftarrow [n]$
18:        else if size of $boundariesContainingN > 1$ then
19:            $mergedBoundary \leftarrow$ nodes from $boundariesContainingN$.
20:            $indepBoundaries.remove(boundariesContainingN)$
21:            $indepBoundaries.insert(mergedBoundary)$
22:        end if
23:    end for
24:    return $indepBoundaries$
25: end function
```

on selecting nodes via their surface definitions. Thus, enabling more complex morph translations to be set up.

3.3.2 Step 2. Initial Translation

Once the problem has been configured, the first step to performing a morph is to apply the user defined transformation function to the set T. The transformation function is given the coordinates of each node and defines a vector field. This can be achieved by defining a scalar function for each dimension to determine the displacement of each coordinate of the nodes. An example of a simple function to translate a set of nodes along the x-axis of a geometry is shown in Equation 3.17.

$$\mathbf{F}(x, y, z) = a \cdot x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}} \tag{3.17}$$

This vector field is applied to the nodes in the fuselage of the DPW geometry, introduced in Section 3.2.3, to stretch the length of the fuselage. Fig. 3.24 shows the before and after comparison from applying this transformation.

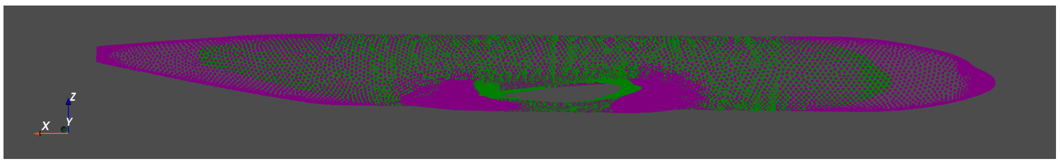


Figure 3.24: DPW fuselage following transformation using Equation 3.17 and a scale factor of 1.2. Green nodes are the baseline mesh, and purple nodes are the transformed nodes.

3.3.3 Step 3. Boundary Constraint Handling

After the initial transformation has been applied, some elements that contain nodes in sets T and C result in deformation of elements that should be constrained. A close up of the fuselage nodes from the DPW mesh are shown in Fig. 3.25. The blue nodes are the original positions of the fuselage nodes, and purple nodes are their positions following translation using Equation 3.17. The nodes on the boundary between the fuselage and wing are highlighted in yellow and red respectively. This figure clearly shows that the size and shape of the boundary has changed following translation of the blue fuselage nodes, set T.

If left unchecked this results in the quality of elements on the boundary between the fuselage and wing significantly decreasing. The wing shape could be modified to match the new boundary. However, doing this may contribute to an additional change in aerodynamic performance. To understand each parameter's individual impact on a geometry, altering multiple parameters simultaneously must be avoided. This prevents any interactive effects, allowing for precise analysis of the influence by each individual parameter. Therefore, the shape of the boundary, the yellow nodes, must be maintained. The boundary is treated as a rigid body that is free to move throughout space. The morphed purple and red nodes must be adjusted to ensure that the red nodes match the shape of the yellow rigid body.

Two different approaches are discussed in this section. The first, uses RBF trained using the displacement of the boundary and interpolates this displacement throughout the original location of nodes in T, however this proved insufficient for many examples. An improvement on this considered

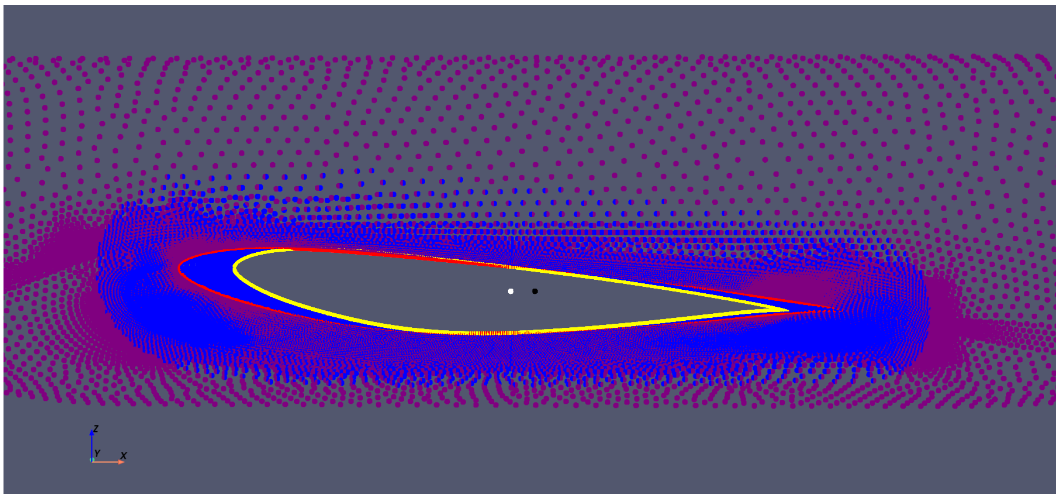


Figure 3.25: DPW before and after stretch. Blue nodes are positions from the baseline mesh of the fuselage nodes, with the nodes that lay on the boundary with the wing coloured yellow. Purple nodes are transformed positions, boundary nodes coloured red.

the constrained shapes and recovered them to their original shape, then the resulting translation is propagated into U using the known translations from the constrained nodes.

3.3.3.1 Constrained Interpolation

To determine how the constrained nodes are translated after the target set of nodes, T, had been morphed, an initial implementation defined known translations to train an RBF network. This network was then used to interpolate all T and U nodes around the constrained nodes, C, shown in Algorithm 3. This method was sufficient for most cases, however in some cases, additional sets of nodes had to be defined as sources to preserve the shape of the geometry.

Algorithm 3 Original implementation to handle boundary constraints.

```
function $ConstrainBoundary(M_0, f(\cdot), T, U)$
    $B \leftarrow$ calculate independent boundaries
    $T_0 \leftarrow$ Vertices from$M_0$in target surfaces
    $T_t \leftarrow f(T_0)$
    for each independent boundary $(b)$ in $B_c$ do
        $c_0 \leftarrow$ centroid of $b$ in $T_0$
        $c_t \leftarrow$ centroid of $b$ in $T_t$
        $b_t \leftarrow c_t - c_0$
        for each vertex $v$ in $b : v \in T_0$ do
            $\tau$ append $v$
            $\delta_k$ append $b_t$
        end for
    end for
    $\tau, \delta_k \leftarrow$ Add nodes from bounding box of $T_t$
    $\tau, \delta_k \leftarrow$ Add nodes from axis lines
    for $i$ in range $1, d$ do
        $\lambda = RBFTrain(\tau, \delta_{k;i})$
        $\delta_i = RBFInterpolate(\tau, \lambda, T_0)$
    end for
    $M_{new} \leftarrow M_0 + \delta^T$ return $M_{new}$
end function
```

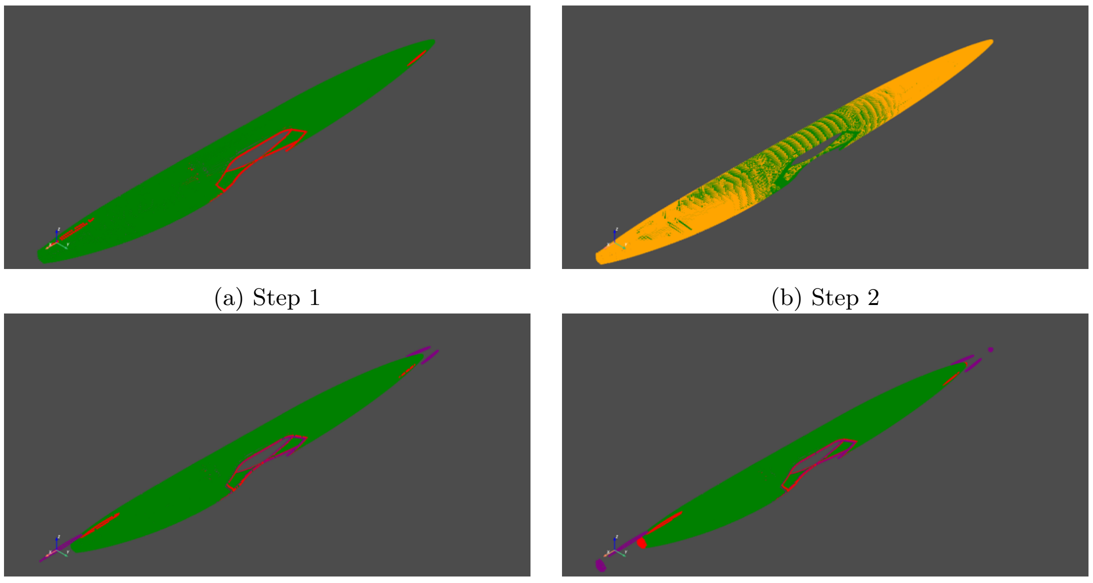

(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4

Figure 3.26: Visualisation of Algorithm 3 applied to stretching the Skylon fuselage. In Step 1, the boundaries are identified (red nodes). In Step 2, the fuselage is stretched (yellow nodes). In Step 3, the new locations of the boundary nodes in the stretched fuselage are identified (purple nodes). In Step 4 the nodes from the bounding box along the x-axis are then added to the set of source nodes.

Consider the constraint function ϕ_c (fuselage stretch) from Fig 3.16f. The steps from Algorithm 3 for this function are shown in Fig 3.26. To ensure that the full length of the fuselage stretch was captured by the RBF interpolation, the nodes from the bounding box had to be included in the set of source nodes, as shown in Fig 3.26d. This worked well on the coarse mesh with approximately even mesh spacing on each node in the fuselage. However when applying the method to higher fidelity meshes with mesh refinement applied along some sections that formed part of the boundaries, the interpolation scheme was not uniformly applied, and regions of higher node density had a higher contributing effect to the interpolation scheme. Fig 3.27 shows the impact this has on the resulting mesh. The leading edge of the canard was refined in both a medium and fine mesh, which resulted in an undesired pinch of the nose as the refinement increased.

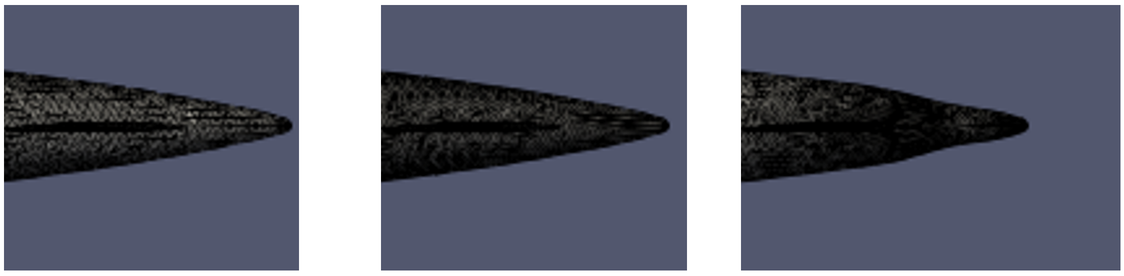

(a) Coarse (b) Medium (c) Fine

Figure 3.27: Close up view of the nose of Skylon after the fuselage has been stretched using RBF mesh morphing for 3 meshes of differing fidelity. The uneven spacing in the finer mesh results in unwanted deformation of the shape of the nose.

Another Skylon parameterisation that highlighted the issues with Algorithm 3 was for the nose droop, ϕ_1 from Fig 3.16a. When using only the boundary nodes to train the RBF interpolation, shown in Fig 3.28a, not enough information about the translation is retained and the resulting morph does not maintain its required shape, Fig 3.28b.

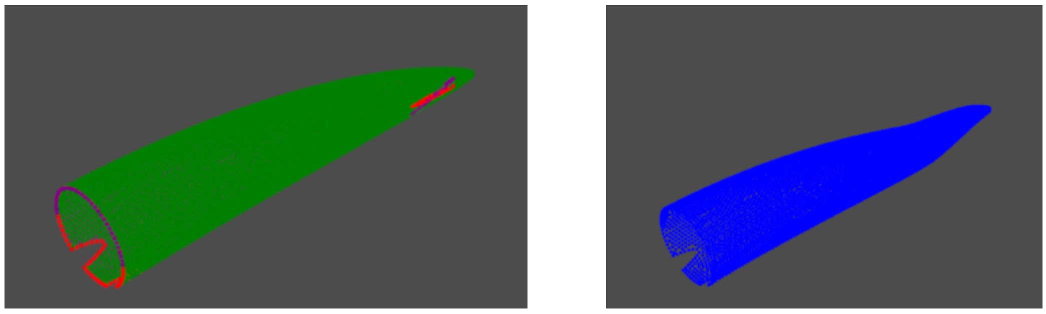

(a) Original and Sources (b) Resulting Mesh

Figure 3.28: The original method for defining sources used to train the RBF interpolation scheme for ϕ_1, drooping the nose of the Skylon fuselage. Only the boundary nodes (red and purple) were used as sources. Fig 3.28b shows that this was insufficient as the end result did not hold the drooped shape.

More source nodes can be added to capture the shape in the interpolation. Fig 3.29a shows an examples of how this can be achieved where all nodes that are within a distance δ from the $x - axis$ are included. The implementation for this step included the option to select a value for δ and specify which axis was included. This effectively captured the shape however for large translations, the shape still began to distort as shown in Fig 3.29b.

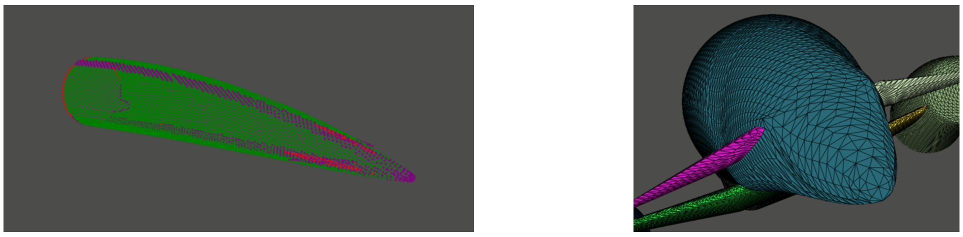

(a) Axis Line (b) Distance From Boundary

Figure 3.29: Examples of different sets of sources used to train the RBF interpolation scheme for ϕ_1, drooping the nose of the Skylon fuselage. Using the axis line was effective for very small displacements, however became distorted around the canards for large displacements.

The method was further extended to allow the option to include more nodes in the set of sources. All nodes that are more than a certain distance from a boundary are included, Fig 3.30. This was very effective at capturing the shape even for large displacements. The drawback of this method is that the total source count also increased, significantly increasing the computation time required to train the RBF network for refined meshes. Another issue with this method was that selection of the distance required was manual, increasing the level of user input required to configure the method.

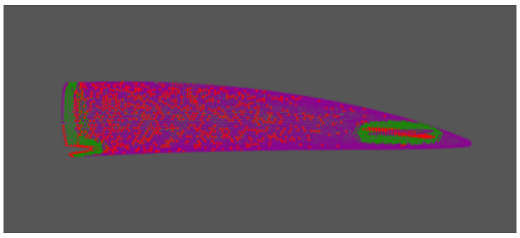


Figure 3.30: Examples of different sets of sources used to train the RBF interpolation scheme for ϕ_1, drooping the nose of the Skylon fuselage. Including all nodes after a certain distance from the boundary proved effective for coarse meshes however significantly increased the total source count for finer meshes so was not scalable.

While Algorithm 3 was sufficient for some cases, examples have been presented where too much information about the translation was lost during this process and the resulting interpolation did not recover enough of the initial translation. Resolving these edge cases required changes to the algorithm that were case specific, significantly increasing the number of configurable options that were needed. To avoid the issues with this method, a method that could be more generically applied to a range of different examples is required.

3.3.3.2 Boundary Recovery

To improve upon the initial implementation, a second method is introduced to handle the boundary constraints. The initial implementation used the translations of the corrected boundary nodes from their initial locations to their translated locations to train the RBF model, and interpolates the translated locations of all nodes from their original locations. Instead, this different method does the opposite of this, and uses the vector between the translated locations and a corrected translated location to train the RBF model and then propagates this correction throughout the translated nodes.

To begin with, the translation is performed on the target nodes, T. Any boundaries between T and C are then recovered, resulting in the constrained nodes in C maintaining their shape and translated nodes from T being affected by the minimum required to constrain C. This avoids the issue with the previous implementation of the translation information being lost. The known translations of the constrained nodes are then used to train an RBF network, along with fixing any nodes in U connected to C to propagate the translation into U. As only the boundary nodes are used to train the RBF network, the node count, and therefore computation time, is kept low.

Algorithm 4 Step 3 - Corrects the boundaries between unconstrained nodes and constrained nodes after initial morph. It is important to note unconstrained = T AND U. This is because when moving U nodes, nodes that need to maintain their position need to be identified and any nodes in C that have moved by morphing T are corrected.

```
1: function $FixConstBounds(\boldsymbol{T}, \boldsymbol{T_0}, \boldsymbol{U}, \boldsymbol{C})$
2:     $cpDiff \leftarrow CenterOf(\boldsymbol{T}) - CenterOf(\boldsymbol{T_0})$
3:     $indepBoundaries \leftarrow$ call $GetIndepBoundaries(\boldsymbol{T} \cup \boldsymbol{U}, \boldsymbol{C})$
4:     $knownTranslations \leftarrow$ empty map
5:     $sources \leftarrow$ empty list
6:     $sourcesDx \leftarrow$ empty list
7:     for each $key, boundary \in indepBoundaries$ do
8:         $TInit \leftarrow boundary \cap \boldsymbol{T_0}$
9:         $TMorphed \leftarrow boundary \cap \boldsymbol{T}$
10:        $UCNodes \leftarrow boundary \cap (\boldsymbol{U} \cup \boldsymbol{C})$
11:        if size of $TMorphed = 0$ then
12:            $knownTranslations[key] \leftarrow \mathbf{0}$
13:        else if size of $UCNodes > 0$ then
14:            for each $M, N_0 \in TMorphed, TInit$ do
15:                if size of $UCNodes > 0$ then
16:                    $boundCorrect = cpDiff$
17:                else
18:                    $boundCorrect = Center(TMorphed) - Center(TInit)$
19:                end if
20:                $knownTranslations[key] \leftarrow boundCorrect$
21:                $sources.insert(N_0)$
22:                $sourcesDx.insert(N_0 - M + boundCorrect)$
23:            end for
24:        end if
25:    end for
26:    if size of $sources > 0$ then
27:        $interpolatedNodeDx \leftarrow RBF(sources, sourcesDx, TMorphed)$
28:        $TMorphed \leftarrow TMorphed + interpolatedNodeDx$
29:    end if
30:    return $TMorphed, knownTranslations$
31: end function
```

A corrective vector *boundCorrect* is introduced to ensure that the boundaries capture any translation in the morph function. The purpose of this algorithm is to move the transformed boundary nodes back to their original shape. Consider the nose droop example, ϕ_1, shown in Fig 3.30. The translation has moved the boundary away from its original location where the center point of the boundary originally lay on the plane $z = 0$ and has moved to a different plane defined by a constant value for z. Without including the *boundCorrect* term, the vector calculated for the known translation of the boundary nodes will translate the boundary straight back to its original position centred on $z = 0$ and the information about the morph function will have been lost.

It is important to note that the *indepBoundaries* are calculated between the set $\boldsymbol{T} \cup \boldsymbol{U}$ and the constrained set C. Because of this, *indepBoundaries* will also include the boundaries between U and C. If the boundary is not connected to T then it must be a boundary between U and C. These boundaries do not need correcting and the known translation for them is set to 0. If the boundary that needs correcting is connected to the set T, then the *boundCorrect* term needs to be calculated. The boundary may be between T and one of either $U \cup C$ or C. For example, consider Fig 3.20 where the outer boundary of the set T is connected to both U and C. In this case, the translation should be captured by calculating the difference between the center point of all nodes in T before and after the morph function is applied. This ensures that any inherent translation in the morph function is captured, for example in the function $f(x) = x + \delta$. Alternatively, if the boundary is only between T and C then it can be handled using the center points of the original and translated boundaries. First, the centre point of the boundary nodes in their original and transformed positions are calculated. These are highlighted in Fig. 3.25 as the black and white points respectively. The original positions of each node in the boundary is then translated by *boundCorrect* so that the shape of the original boundary holds, but is centred at the same point as the center point of the translated nodes. The vector from the nodes transformed position to their original position (after *boundCorrect* has been applied to each node) is then calculated. This vector is used as the known values to train the RBF network, where these boundary nodes are the sources. Once the RBF field has been trained, it is applied to the remaining nodes in T to calculate their displacements relative to the boundary nodes.

This method reduces the deformation of the elements immediately connected to the boundary and spreads the deformation out amongst all elements consisting of elements in T. While this correction ensures the boundaries with constrained nodes remains fixed, it does result in losing some of the desired transformation applied to T. This method loses a significantly smaller amount of information than the constrained interpolation method from Section 3.3.3.1 did. The shape lost is necessary to ensure that the geometry remains in a state suitable for CFD computation.

3.3.4 Step 4. Propagating into U

After the nodes in T have been morphed, and the boundaries between T and C corrected, the boundaries between T and U need to be considered. This boundaries are much simpler to handle in this case than the boundary constraint handling. The nodes in U are unconstrained, so rather than the translation being corrected at the boundaries, it must be propagated into U. RBF interpolation is used to achieve this.

In this case the nodes in boundaries of U are either connected to T or C. The nodes connected to C are constrained to not move, so have a known displacement of $\mathbf{0}$. Alternatively, the nodes on the boundary with T have their known displacement set as the displacement of that node between its original position and morphed position. If a node is connected to all 3 sets, the constrained case takes precedence and the known displacement is $\mathbf{0}$. From here, the displacements for all other nodes in U can be interpolated from these sources after training an RBF network.

3.3.5 Step 5. Updating The Mesh

Once the morph function has been applied and propagated through the unconstrained nodes, the mesh needs to be updated with all the new node locations. During the boundary constraint handling in Section 3.3.3, each independent boundary is stored and returned as a map, which is a collection of key-list pairs. The key is an id tag for the boundary and the list contains all the nodes in that independent boundary. This map contains the list of the boundaries that have moved, and all nodes in C that are connected to that boundary now need to be updated. As different boundaries are moved by different displacements, each independent set of nodes connected to the independent boundaries needs to be identified. For example, consider a case where the fuselage of an aircraft is stretched and the tail and wings are constrained. After stretching the fuselage, the position of the wings and tail will change by different amounts due to the difference in their positions relative to the reference point used to stretch the fuselage.

In this section, two methods are compared - an initial implementation that worked well for coarse simple cases, and an improved method that improved on the slow aspects of the original method.

3.3.5.1 Original Implementation

The original implementation worked by looping through each boundary and checked each node to add any nodes in C that were connected to it to the rigid body connected to the boundary. A queue was used to track all nodes that needed checking and the node connectivity was recomputed to reduce run times.

Algorithm 6 Step 5 - Recover boundaries.

```
1: function $RecoverBoundaries(\boldsymbol{T}, \boldsymbol{U}, \boldsymbol{C}, boundaryTranslations)$
2:     $correctedNodes \leftarrow$ empty map
3:     $indepBoundaries \leftarrow$ call $GetIndepBoundaries(\boldsymbol{T} \cup \boldsymbol{U}, \boldsymbol{C})$
4:     for each $key, boundary \in indepBoundaries$ do
5:         $bt \leftarrow boundaryTranslations[key]$
6:         if $bt = 0$ then
7:             Boundary not moved, skip and check next.
8:         else
9:             $queue \leftarrow boundary$
10:            $nodesConnectedToBoundary \leftarrow$ empty list
11:            while size of $queue > 0$ do
12:                $n \leftarrow queue.pop()$
13:                for $m \in$ nodes connected to $n$ do
14:                    if $m$ not already checked AND $m \notin C$ AND $m \notin queue$ then
15:                        $queue.push(m)$
16:                    end if
17:                end for
18:                $nodesConnectedToBoundary.insert(n)$
19:            end while
20:            for $n \in nodesConnectedToBoundary$ do
21:                $correctedNodes.insert(n + bt)$
22:            end for
23:        end if
24:    end for
25:    return $correctedNodes$
26: end function
```

This method was exhaustive however was very slow and did not scale well. For every node that was checked, the algorithm has to check if the node has already been checked and if it is already in the queue or not. As the number of nodes in the mesh increases, these comparisons will only continue to slow the function down. Appropriate use of data structures could offer some time saving, however as the whole list needs to be checked this will not eliminate the increased runtime.

3.3.5.2 Chain Implementation

Algorithm 7 improves on this by constructing smaller chains of nodes and combining them into the rigid bodies. This method is similar to Algorithm 2, used for determining independent boundaries by comparing sets of nodes and constructing the boundary by building chains of nodes and combining them when it is identified they are part of the same boundary. The difference in this case is that all nodes in C are considered rather than just those on the boundary. This offers significant time savings over the original method introduced in the previous section as each node is compared against smaller chains rather than against a single monolithic set that grows with each loop.

Algorithm 7 function: Recover rigid bodies. chain method

```
1: function RecoverRigidBodies(T,U,C)
2:     $numBodies \leftarrow 0$
3:     $rigidBodies \leftarrow$ empty map
4:     $indepBoundaries \leftarrow$ call $GetIndepBoundaries(\boldsymbol{T} \cup \boldsymbol{U}, \boldsymbol{C})$
5:     for $\forall n \in (C \cup indepBoundaries)$ do
6:         $bodiesContainingN \leftarrow$ empty list
7:         for $\forall m$ connected to $n$ do
8:             for each $key, body$ in $rigidBodies$ do
9:                 if $m \in body$ & $n \notin body$ then
10:                    $body.insert(n)$
11:                    $bodiesContainingN.insert(key)$
12:                end if
13:            end for
14:        end for
15:        if size of $bodiesContainingN == 0$ then
16:            $numBodies$ += 1
17:            $rigidBodies[numBodies] \leftarrow [n]$
18:        else if size of $bodiesContainingN > 1$ then
19:            $mergedBody \leftarrow$ nodes from $bodiesContainingN$.
20:            $rigidBodies.remove(bodiesContainingN)$
21:            $rigidBodies.insert(mergedBody)$
22:        end if
23:    end for
24:    return $rigidBodies$
25: end function
```

Table 3.6 shows the wall clock run-times for each of the two algorithms presented here that are used to determine which nodes are part of the rigid body. The DPW example presents the most striking case for why the updated method is essential for scaling the algorithm to highly refined meshes.

Geometry	Original Method (s)	Chained Method (s)
Skylon ϕ_1 - Nose Droop	9.00	0.52
Skylon ϕ_2 - Fuselage pinch	14.71	0.65
Skylon ϕ_3 - Central panel stretch (x)	14.26	0.65
Skylon ϕ_4 - Central panel stretch (z)	14.15	0.65
Skylon ϕ_5 - Central section stretch	3.78	2.09
Skylon ϕ_6 - Fuselage stretch	4.00	1.74
DPW ϕ_1	1946.97	5.73

Table 3.6: Runtime comparison of function used to determine the nodes that make up each rigid body. Values are given for each of the Skylon parameterisations and the DPW parameterisation. Time only includes functional call time, not any additional time spent morphing the mesh or any other algorithm.

3.3.6 Complete Algorithm

The complete algorithm using all previous steps is summarised in Algorithm 8. The morph begins by applying a translation function to the desired nodes. The correction required to ensure the boundary between the morphed nodes and constrained nodes is then calculated. Boundaries between morphed nodes and unconstrained nodes are then used to propagate the morph into unconstrained sections. Once the mesh nodes along the boundaries has been updated, the position of independent rigid bodies connected to the boundaries in C is determined and recovered.

Algorithm 8 Mesh morphing algorithm

```
1: function $Morph(\boldsymbol{T}, \boldsymbol{U}, \boldsymbol{C}, f(\cdot))$
2:     $T_m \leftarrow f(T)$
3:     $T_c, boundCorrections \leftarrow FixConstBounds(T_m, T, U, C)$
4:     $U_m \leftarrow PropIntoU(T, T_c, U, C)$
5:     $C_c \leftarrow RecoverBoundaries(T_c, U_m, \boldsymbol{C}, boundCorrections)$
6: end function
```

3.3.7 Configurable Settings

Some of the parameters to be used throughout Algorithm 8 require user-defined configurations. Some of these configurable properties are discussed here. In this section, these parameters and their implementations are discussed.

3.3.7.1 Maintain Center Point

When first implementing the scaling translation, the reference point was taken to be the origin. In some transformations, this shifted the center of the body of the translatable set of nodes. Moving the geometry in some meshes away from the datum point can cause issues with comparing calculations, and therefore should be avoided. The main body of the geometry could move too far towards the farfield in extreme morphs. The translation must therefore be performed relative to the center of the translatable nodes T.

The *center point* is defined for a set of nodes as the mid point of the bounding box over the set. A separate definition commonly used to determine the center of a set of nodes is the *centroid*. The *centroid* is defined as the average position of the set. The *center point* calculation is preferable in this application for determining the center of a set of nodes. The *centroid* calculation can be affected by node distribution within the set. This means the center calculation may not be the same location for two different meshes of the same geometry when mesh refinement differs between meshes.

The center point constraint can be handled by either performing the scaling about the center or by translating the body back to its original center after the translation. The decision to maintain the center point under translation is left up to the user. The implementation is however assisted by including python functions to automatically handle this constraint, but the decision to perform this must be made during the problem setup. The constraint should be handled within the function definition itself and so can be modified freely by the user.

3.3.7.2 Basis Function Selection

The choice of basis function has a significant effect on the resulting mesh morph. Basis functions with local support can restrict the effect of the morph to specific locations, while those with global support can affect the shape of the resulting interpolation depending on which function is used. The basis function is therefore a configurable parameter that can be modified or included as a hyperparameter within the optimisation process. In general it is recommended that the default settings are used for these parameters however the effect of selection is studied in more detail in Section 3.1.6

The mathematical definition of each basis function is given in Section 3.1. From this list, the functions implemented and used throughout this section are *multiquadric*, *gaussian*, *inverse multiquadric*, both old and new implementations, and *Wendland* C_0 & C_2. For the multiquadric and inverse multiquadric functions, $n = 1$. None of these functions require the use of polynomial support however the implementation has functionality built in to handle this if required.

3.3.7.3 Shape Parameter Selection

Globally supported RBF's include a shape parameter and locally supported functions include a similar parameter that determines the support radius of the function. There are several ways to calculate this value. Some of the different methods are described in Section 3.1.4. The methods implemented and explored within this chapter are *radius* and *one*. They are implemented by allowing users to select the method of choice, and a scaling constant ϵ_s. In the *radius* case,

$$\epsilon = \epsilon_s \cdot max(d)$$

where d is the vector of known displacements for a set of nodes and ϵ_s is a constant value used to scale this value. This method allows for a radius around the translated nodes to be controlled. For the *one* method, this is simply a constant value of 1 which when scaled by ϵ_s allows the use to fix a particular value for ϵ.

3.3.7.4 Parameter Modification

This parameter simply scales the shape parameter by a fixed quantity. If set to 1, the shape parameter will not change. Combined with the shape parameter selection method of *one*, this allows the user to fix the shape parameter to a selected value. Combined with other methods it allows for control over the radial effect of the method to be modified where appropriate.

3.3.8 Morph Directory

The parameterisation setup defined in Section 3.3.1 and the methodology used to morph a mesh for a given parameterisation requires some settings to be configured by the user. All of this results in the method being setup and configured in a single folder for a used case. This folder should contain the baseline mesh, translation function definitions and a configuration file. In this section, the resulting implementation is discussed and explained in detail. An example folder structure for what the Skylon case study described in Section 3.2.4 could look like is shown in Fig. 3.31. The exact structure can be easily adapted. For example, the modified surface mesh could be output to a folder in a separate location

```
folder_name
 |
 |-- functions.py
 |
 |-- indep_params.csv
 |
 |-- MorphInfo.json
 |
 |-- morph.log
 |
 |-- surface_mesh.fro
 |
 |-- surface_mesh_modified.fro
```

Figure 3.31: Example data structure for a morph directory.

3.3.8.1 Functions

This methodology aims to keep the translation function definition as generic as possible to allow as much freedom as possible during shape optimisation. The function definition is therefore a user able parameter. In the python implementation of this method, a *functions.py* file is dynamically loaded at run time however alternative implementation methods may be required for other languages.

3.3.8.2 Morph Info

All of the customisable parameters that need to be defined are stored in the *MorphInfo.json* file. This file can be generated using *CreateMorph* Paraview Python script. When the user has defined all nodes, the selection is saved in this file and the additional parameters can be selected. Table 3.7 provides an overview of all parameters and their default values.

Property	Enum
None	0
Volume	1
Surface Area	2
Center Mass	3
Centroid	4
X Planform	5
Y Planform	6
Z Planform	7
Min x	8
Max X	9
Min Y	10
Max Y	11
Min Z	12
Max Z	13

Table 3.7: Geometric property enumerations for property constraint configuration. The properties listed are calculated over the whole surface mesh. For example, the volume for the mesh of a unit sphere would return approximately $4\pi r^2$ with a small margin for discretization error.

Geometric property constraints can also be configured within this file. A complete overview of what geometry constraints are is introduced in Section 3.4 including details of the implementations used to handle them. The required property to be used is configured using the enumerations in Table 3.7. To select a property, set the value of *const_prop* in *MorphInfo.json* to the corresponding integer value from this table. The implementation used throughout this thesis calculates this constraint over the whole geometry but it would be trivial to extend the implementation to target specific regions/surfaces.

3.3.8.3 Independent Parameters

The independent parameter values can be controlled by an optimiser, however the user may want to setup some test cases with specific values to explore. A configuration file is therefore added to handle this. The file *indep_params.csv* contains a comma separated table of tests. The columns represent the ϕ_i values respectively for all n parameters and each row represents a different test.

3.3.8.4 Running A Test

Once the functions and all configurable parameters have been defined, a command line program provides functionality for performing individual morphs, or the directory can be read and used by an optimisation method.

3.4 Geometric Constraint Handling

Many optimisation problems include constraints on geometric properties of a geometry. For example, the volume of an aircraft is a very commonly constrained property in aerodynamic optimisation cases [68]. This is often handled by adjusting the cost function, adjusting a designs performance based on the constraint to be applied. This allows an engineer to control the importance of a particular constraint however in cases where function evaluations are expensive, this may not be the best approach to handle the constraint. In equality constraints, the feasibility region is so small that most designs would not be valid. In cases such as this where the evaluation function could take in the order of hours to complete, it is not worth the physical resource to explore invalid designs. Even when constraints can be applied prior to evaluation to filter out designs that do not meet constraints, this could still lead to algorithms taking a long time to identify valid geometries. The methodology needs to be able to handle enforcement of equality constraints on geometric properties. In this section, the implementation for this is discussed.

Geometric constraints can be handled by defining a dependent morph. The variable parameter for this morph is not set by the user or optimiser, instead, the required value to constrain the required property is determined by calculation or iteratively using an algorithm. In the case of complex geometries, it is not known apriori what the exact relationship between the constraint function and the mesh property is, making exact calculation can be challenging or not possible.

As an example, consider a generic wing body geometry for which the objective is to minimise drag by changing the shape of the wing, but with the constraint that the overall volume of the geometry must remain constant. Let V_0 be the volume of the baseline design. After a set of morphs have been applied to the wing, the volume of the morphed geometry V_m would then be different from that of the baseline design such that $V_m \neq V_0$. The aim is then to recover the required volume. For this theoretical example, define a constraint function (or repair operator) to stretch the length of the fuselage by a certain scalar quantity, x. A value of $x < 1$ results in the fuselage shrinking (and therefore the overall volume of the geometry), whereas a value of $x > 1$ has the opposite effect. The shape of the wings under this transformation should remain the same to avoid the constraint function affecting the independent parameters. Due to the complex shape of the geometry, it is not certain what the rate of change will be on the overall volume relative to x will be, $\frac{dV}{dx}$, without performing the stretch and re-calculating the new volume. It is therefore not immediately clear what value x should take to constrain the volume. This makes an iterative approach the preferred method, which is achieved through exploration and refinement. In this section, the performance of multiple different implementations of iteratively determining the required value for the dependent parameter

x are explored. A value of x is trailed, the effect this has on the volume is determined, and this information is used to determine what the next value of x to apply should be. Each iteration, the geometry is updated and the process is repeated, refining the magnitude of x as appropriate until the overall volume V_m converges towards V_0. The general overview of the iterative geometrical constraint handling method is described in Algorithm 9.

Algorithm 9 In the constrained morph function, the baseline mesh (M_0) is modified by the independent parameters ($\boldsymbol{f}$), then the property ($P(\cdot)$) is constrained to that of the original mesh using the function ($f_c(\cdot)$) by iteratively changing x to converge P_m towards P_0. δ is an arbitrary small value used as an initial guess for x

```
1:  function $ConstMorph(M_0, \boldsymbol{f}, P(\cdot), f_c(\cdot))$
2:      $P_0 \leftarrow P(M_0)$
3:      $M_m \leftarrow f(M_0)$
4:      $P_m \leftarrow P(M_m)$
5:      $x \leftarrow \delta^+$
6:      if $P_m < P_0$ then
7:          $x \leftarrow \delta^+$
8:      else if $P_m > P_0$ then
9:          $x \leftarrow \delta^-$
10:     end if
11:     while $P_m \neq P_0$ do
12:         $M_m = f_c(M_m, 1 + x)$
13:         Refine $x$
14:         $P_m \leftarrow P(M_m)$
15:     end while
16: end function
```

It is assumed during the implementation of this method that the constraint function to be applied uses a scalar multiplication on a section of the geometry, as in the wing-body example. A null operation on the geometric property would therefore be a scalar translation where $x = 1$. The constraint parameter x is therefore as a small value $\delta \approx 0$ and applied as an offset from the null transformation as this makes the refinement of x simpler to implement. In the remainder of this section different methods of achieving the iterative approach are explored, each method describes a more sophisticated refinement of x in line 11 from Algorithm 9 than the previous.

3.4.1 Simple Refinement

An initial attempt to perform the geometric constraint was to simply iterate by a small value for x towards the required value P_0 until either P_0 was reached or it was overshot. If the value was overshot, the value of x was decreased and the iteration continued back towards P_0. Algorithm 10 shows this step by step process. The small value of x was initialised to 10% or 0.1, line 4. If the property value for the new mesh after applying the independent morphs was larger than that of the baseline then the value needs to be negated so that the constraint morph drives the value of P_m down towards P_0 instead of up, line 6. In each loop, the mesh is updated, the new value of the new meshes property is calculated and x is updated as required.

Algorithm 10 Pseudo-code defining the iterative simple refinement approach taken to determine the dependent parameter.

```
1:  function $SimpleRefinement(\boldsymbol{M}_0, \boldsymbol{M}_m, P(\cdot))$
2:      $P_0 \leftarrow P(\boldsymbol{M}_0)$
3:      $P_m \leftarrow P(\boldsymbol{M}_m)$
4:      $x_0 \leftarrow 0.1$
5:      if $P_m > P_0$ then
6:          $x_0 = -x_0$
7:      end if
8:      while $P_m \neq P_0$ do
9:          $P_{m_0} \leftarrow P_m$
10:         $M_m \leftarrow f_c(M_m, 1 + x)$
11:         $P_m \leftarrow P(\boldsymbol{M}_m)$
12:         if $((P_{m_0} < P_0) \wedge (P_m > P_0)) \vee ((P_{m_0} > P_0) \wedge (P_m < P_0))$ then
13:             $x \leftarrow -0.1x$
14:         end if
15:     end while
16: end function
```

This method is applied to the volume constrained sphere example described in Section 3.2.2. The convergence history for the volume of the geometry is given in Fig. 3.32. The graphs in this figure show how the absolute value of x decreases over time while its sign changes when the volume overshoots the target volume until the target volume is reached. After each iteration of this method, the only check made is whether the target property is above or below the target. No indication of the magnitude of the error is used to assist the selection of x, presenting a clear opportunity for improvement.

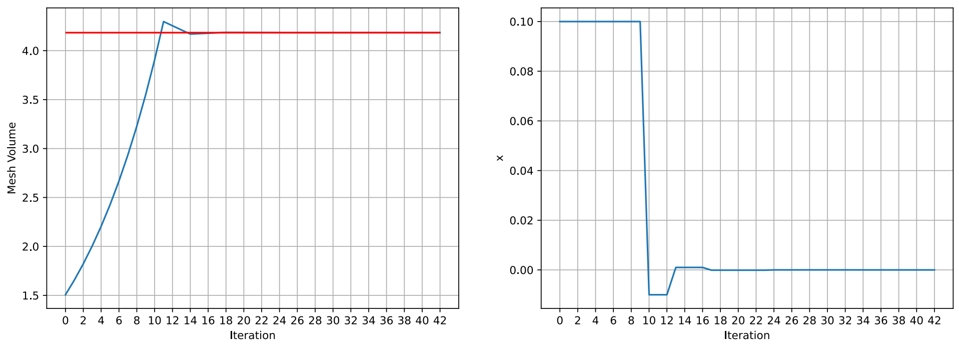


Figure 3.32: The convergence history for the volume of the geometry during volume constraint of a morphed sphere with the baseline target volume given by the horizontal red line. The right hand graph shows how the value for x changes over each iteration.

3.4.2 Scaled Refinement

To build on the simple method, it is noted that some values of x are used repeatedly in sequence as the target property is converged on. This slows the overall algorithm down a complete morph of the geometry must be made each iteration. It would make more sense to achieve the same outcome from multiple morphs in one motion. This is achieve by scaling the value of x dependent on the distance from the target. The exact relation between x and the mesh property is still not known, however, the change in volume in Fig. 3.32 is a quasi-linear relationship for constant x. That is to say that if x remains unchanged the rate of change of x each iteration stays fairly similar. This can be leveraged to calculate a scale factor, C, to apply to x to make a more significant jump towards the target value in one morph, significantly reducing the total number of iterations required to reach the target thus speeding up the over all constrained morph process. The updated algorithm including this calculation is given in Algorithm 11. Most of the algorithm remains largely unchanged from Algorithm 10, the main difference that x is scaled with a variable C prior to the morph being applied, line 15. This constant is calculated by dividing the current distance from the target by the previous δ_x. To calculate this value at least one iteration needs to be performed with the current value of x to determine how much $P(M)$ is changed by and thus calculate δ_x.

Algorithm 11 Pseudo-code defining the iterative scaled refinement approach taken to determine the dependent parameter.

```
1:  function $ScaledRefinement(\boldsymbol{M}_0, \boldsymbol{M}_m, P(\cdot))$
2:      $P_0 \leftarrow P(\boldsymbol{M}_0)$
3:      $P_m \leftarrow P(\boldsymbol{M}_m)$
4:      $x_0 \leftarrow 0.1$
5:      $\delta_x \leftarrow 0.0$
6:      if $P_m > P_0$ then
7:          $x_0 = -x_0$
8:      end if
9:      while $P_m \neq P_0$ do
10:         $P_{m_0} \leftarrow P_m$
11:         $C \leftarrow 1.0$
12:         if $\delta_x \neq 0.0$ then
13:             $C \leftarrow \lfloor \frac{|P_0 - P_m|}{|\delta_x|} \rfloor$
14:         end if
15:         $M_m \leftarrow f_c(M_m, 1 + (Cx))$
16:         $P_m \leftarrow P(\boldsymbol{M}_m)$
17:         if $((P_{m_0} < P_0) \wedge (P_m > P_0)) \vee ((P_{m_0} > P_0) \wedge (P_m < P_0))$ then
18:             $x \leftarrow -0.1x$
19:             $\delta_x \leftarrow 0.0$
20:         end if
21:     end while
22: end function
```

The convergence history for the volume constraint of the sphere under the same conditions as applied to the simple method is shown in Fig. 3.33 where it can be seen that the volume is constrained in significantly fewer iterations than the simple iterative approach.

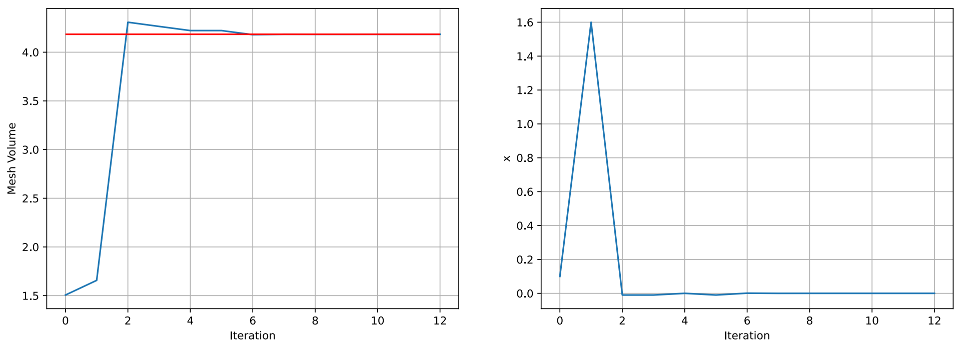
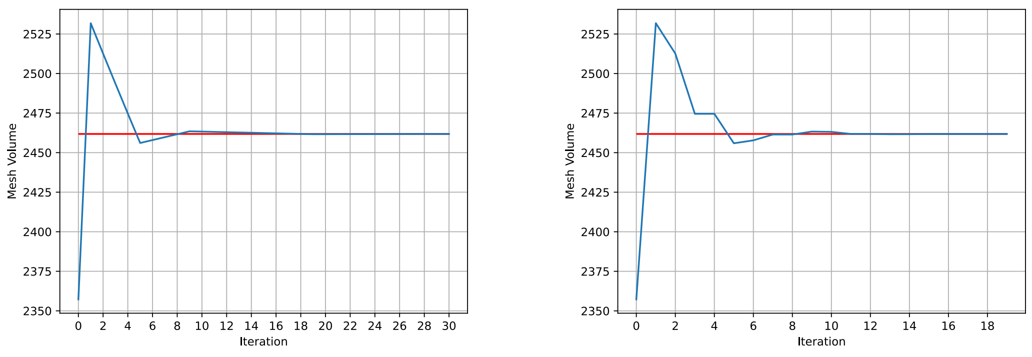


Figure 3.33: The convergence history for the volume of the geometry during volume constraint of a morphed sphere with the baseline target volume given by the horizontal red line. The right hand graph shows how the value for x changes over each iteration.

3.4.3 Simple vs Scaled Comparison

So far, each method has been applied to the volume constrained morphing of the simple sphere parameterisation described in Section 3.2. The performance of each approach is also compared on a more complex geometry. The parameterisation of the Skylon spaceplane defined in Chapter 4 is used. Again, the goal is the volume constraint of the overall geometry where ϕ_1-ϕ_5 are independent parameters and ϕ_c, the fuselage stretch, is used to constrain the volume. Both of the above techniques are applied and the iteration count is compared for each method. The convergence histories for both the simple and scaled methods are compared in Fig 3.34. Comparing the two methods, it can be seen that using the scaled refinement method results in convergence on the volume 33% quicker than the simple method.



(a) (b)

Figure 3.34: Volume constraint example of Skylon using the simple method, 3.34a, and the scaled method 3.34b with an initial step size of 0.1.

There is a very large jump in volume in the first iteration in both cases. The volume increase far exceeds the required amount. This would suggest that a better method could be to reduce the size of the first jump, and make a larger step in the second iteration where some knowledge of the relation between x and P_m is known, thus bringing V_M much closer to V_0 after the second iteration than is currently seen. Setting the initial step size to $x_0 = 0.01$, reducing it from 10% to 1%, and repeat the same algorithm, the convergence history for the scaled refinement is quicker again, reducing the original iteration number to 33% over the simple method, Fig 3.35.

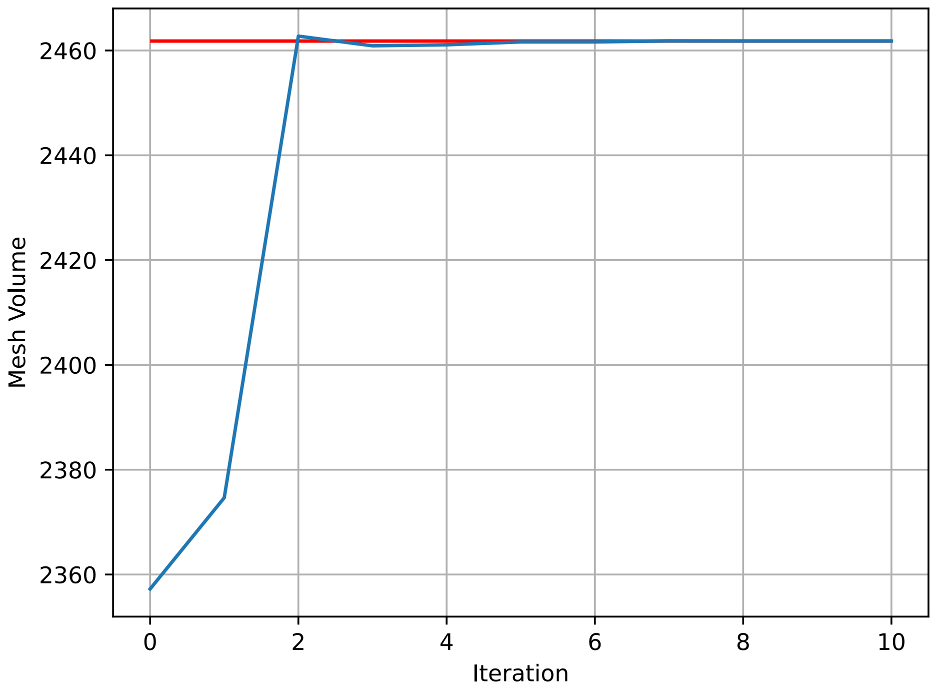


Figure 3.35: Convergence history for volume constraint of Skylon geometry using the scaled method with $x_0 = 0.01$.

3.5 Source Correction

One side effect of morphing the surface mesh and not the volume with it is that the source distributions and background spacing of the baseline geometry may not correspond with the appropriate targeted regions in the new mesh.

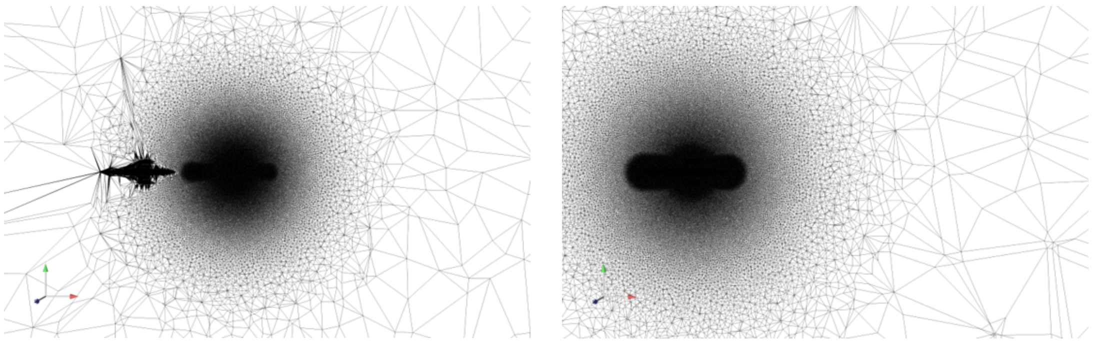

(a) Uncorrected Sources (b) Corrected Sources

Figure 3.36: Example of the source correction following a translation of the surface mesh along the x-axis.

An example of when this may be required is shown in Fig. 3.36. Here, two different volume meshes are generated on the translated geometry, Fig. 3.36a where the sources are in their original locations, and Fig. 3.36b where the sources have been realigned to the new geometry location. In this dummy example, the x coordinate of each mesh on the surface is translated along the x axis by 100 m. The line source that passes through the length of the fuselage and two point sources around the centre of the mesh need to also be translated. A simple translation of the location of the source is applied to handle this, as shown in Algorithm 12. This allows the source locations to be updated without user intervention or definition.

Algorithm 12 Source translation method.

```
1: for each source center: c do
2:     $p_{id} \leftarrow$ Node id of closest node to c
3:     $p_{original} \leftarrow$ Original coordinate of $p_{id}$
4:     $p_{translated} \leftarrow$ Translated coordinate of $p_{id}$
5:     translation $\leftarrow p_{translated}$ - $p_{original}$
6:     c $\leftarrow$ c + translation
7: end for
```

If the domain of the mesh morph is known during the problem set up, the sources can be configured accordingly to ensure that this does not become a problem. This could also be extended to include using the RBF displacement fields to act on the background spacing nodes or source centres as appropriate, however, this is not considered within the scope of this work.

3.6 Summary

In this chapter, a mesh morphing framework was introduced that allows users to define a set of parameters for a given geometry that morph the shape. The parameterisation can be used to morph a geometry while following geometric constraints such as geometry volume. The method was demonstrated by setting up a volume constrained parameterisation of the Skylon spaceplane and the evolution of the implementation was also described. Once defined, the parameter values can be changed and a new mesh generated automatically, enabling optimisation methods to explore the design space. This will be investigated in the subsequent chapters.

Chapter 4

Design of Experiments Based Optimisation

The morphing framework defined in Chapter 3 enables engineers to define a set of parameters for a particular geometry. This results in a space of valid designs, referred to as a design space. Once a design space has been defined, it needs to be explored to identify where the optimal solutions are within it for a given evaluation function.

In this chapter, a Design of Experiments (DOE)-based approach to optimisation of a geometry is used to explore the design space. DOE is a very simple optimisation procedure that is still widely used in industry. It is a systematic experimental approach to optimisation where the design space is randomly sampled, and the information gained from the samples is used to locate an optimal solution. DOE provides a simple example of how the mesh morphing framework can be applied to a use case representative of those used in industry.

The use case that the DOE method is applied to is the design space of the Skylon spaceplane constructed by the parameterisation in Section 1.4.2. The aim is to explore the aerodynamic performance of the spaceplane and minimise the drag coefficient of the design. In this chapter, the results of this design space exploration and the optimal design are analysed.

4.1 Methodology

The general steps taken to perform a DOE based optimisation procedure can be summarised in the following steps:

- Define the design space.
- Define the objective and constraints.
- Sample design space.
- Train response surface.

- Optimise the objective function using the performance from the response surface.
- Test predicted optimum.

First, the problem must be parameterised, and the design space constructed from this. Bounds for each parameter can be set to unbounded or set to a specific range based on the engineering requirements of the problem. For optimisation problems, the objective function and any necessary constraints need to be defined to specify how the geometries are evaluated. Any additional computation parameters also require definition to complete construction of the space. For example, CFD-based optimisation studies require flow conditions such as the freestream Mach, Reynolds number, and attitude to be specified. The design space is sampled to produce a set of training data for which a response surface is fit according to the results from the objective function and the extrema for this model can be identified. In the remainder of this section, each step is described in detail and some visualisation techniques for multi-dimensional data sets is presented.

4.1.1 Defining Design Space

The first step in any optimisation problem is to construct the design space. For the Skylon case presented in this chapter, the geometry is parameterised such that the framework introduced in Chapter 3 can be applied to generate new designs. Upper and lower bounds for each parameter are then determined according to engineering requirements, geometric constraints (such as avoiding surface contact), or mesh quality limitations where appropriate. Engineering constraints, such as structurally driven limitations, can be applied during parameter definition by the engineer. For example, if the length of a wing was the design parameter being explored, there may be an upper limit on the length that existing materials could structurally support. Other constraints such as a maximum length to allow the aircraft to fit in the hangar could be affect the bounds. The Skylon optimisation case explored in this chapter is evaluated using CFD, for which the mesh quality is particularly important. As highlighted in Section 2.2, the quality of a mesh can affect the performance of CFD and the quality of its results. RBF based mesh morphing methods often introduce mesh degradation within the volume mesh in particular. Using the framework described in Chapter 3, only the surface mesh is morphed, and the volume mesh is regenerated for each new geometry, which reduces the level of degradation in the volume mesh. However, some degradation of the surface mesh will still occur. An absolute upper and lower bound for each parameter can therefore be determined by pushing the limits of the bounds for each parameter to the point where the volume mesh no longer generates successfully. These bounds can be further constrained depending on the engineering requirements as necessary.

Geometric constraints such as those described in Section 3.4 can be defined dependent on the given parameters and baseline geometry where appropriate. This also contributes to the construction of the design space, limiting it to a subset of designs.

With the parameter bounds defined, other parameters such as simulation conditions and optimisation parameters must be defined. Some of the parameters that require defining for CFD evaluation are: Mach number, Reynolds number, angle of attack, convergence criteria, turbulence model and its specific parameters, and dissipation schemes, where appropriate.

4.1.2 Sampling Design Space

In DOE optimisation, after the design space has been specified an initial sample needs to be taken to explore the space. For the Skylon study in this chapter, the design space is sampled using Latin Hypercube Sampling (LHS) [66].

In LHS the hypercube of the design space is divided into a grid of n equal intervals where n is the number of samples required. Samples are then spread amongst the cells in this grid, ensuring that there is only one sample in each hyperplane. In the most basic implementation of LHS, the samples are drawn from a uniform distribution defined across each cell. An example of how a 2D space is divided and the samples drawn is given in Fig. 4.1.

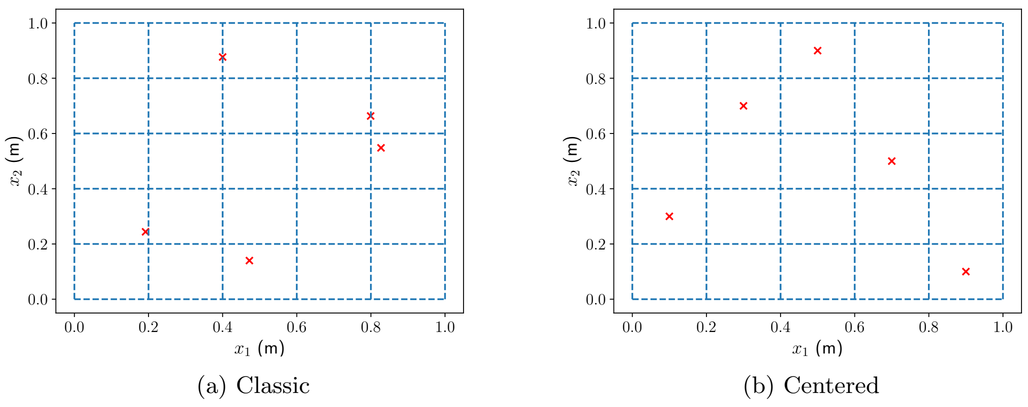


Figure 4.1: Examples of variations of LHS methods over the domain $[0, 1]^2 \subset \mathbb{R}^2$ for 5 sample points. Sample locations are shown with red crosses, and the uniform grid that divides the domain is shown with dashed lines.

In Fig. 4.1a there are two nodes with $\boldsymbol{x} \approx [0.8, 0.6]$ that in close proximity to each other despite not being located within the same hyperplane. This colocation of samples can be avoided by using a variation of the classic LHS method called the centred approach [1]. In centred LHS, samples are placed in the centre of each cell of the grid, ensuring the samples maintain at least a minimum distance of 1 grid square apart. Fig. 4.1b presents the same design space sampled using this method where a more uniform spread over the domain is achieved.

While it is generally recommended to use approximately $n = 10d$ samples where d is the number of dimensions, to generate a sample dataset, this may not be sufficient to capture the representativeness of the design space. A number of factors can impact the effectiveness of using LHS for a one-shot sampling approach, such as the complexity of the design space, non-linear relationships, interactions between variables, and high dimensionality. Instead of relying on a fixed sample size, one should assess and potentially adjust the sampling strategy accordingly.

Once generated, the parameters are used to morph the new geometries. Geometric constraints can then be applied, such as fixing the volume or surface area of all or part of the geometry using constraint functions.

4.1.3 Regression

When the design space has been sampled and new geometries generated, the evaluation function can be calculated for each geometry to generate the response for each geometry in the dataset. These training data points are then used to fit a response surface, with the design parameters as independent parameters, and the response as dependent parameters.

In this chapter, RBF interpolation is used as the model for the response surface. RBF interpolation is particularly well-suited to model problems with approximately 20 or less parameter dimensions, making it ideal for handling the 5-dimensional parameter space of the Skylon example. They are also inexpensive to compute and scale well when the size of the data set increases. To train the RBF network, each design parameter is standardised so that the values have a mean distribution of 0 and unit variance, such that each design parameter is now normally distributed. The percentage change of the response of each sampled geometries from the baseline response is then calculated, and this is used to fit the regression. This is done to more clearly distinguish whether a design performs better or worse than the original.

4.1.4 Optimised Design

Once a response surface has been fitted to the sampled dataset, an optimiser can be used to identify the extrema of the response surface and suggest an optimal design. Throughout this work, unless otherwise specified, the *differential_evolution* function from the Python module *numpy* is used to locate the minima of the surface [43]. The default parameters are used for the configuration of this function. The suggested optimal design is then morphed and evaluated to determine the actual response.

4.1.5 Visualisation

Multidimensional data sets can be difficult to visualise. Three techniques are discussed throughout this chapter to assist with this: parallel plots, t-Distributed Stochastic Neighbour Embedding (t-SNE) visualisations, and Kernel Density Estimation (KDE) smoothing. To help with the description of these methods, a commonly used called the *Iris* data set is used. The *Iris* dataset consists of a set of 50 Iris flowers of three different species and is frequently used for simple data analysis examples [30]. Four features of each sample were recorded; the lengths and widths of the petals and sepals of each sample.

4.1.5.1 Parallel Plots

Parallel plots can be used to plot raw inputs and responses and qualitatively identify any correlations within the data. Each parameter and response has its own parallel axis, usually vertical, and the values for each different design are plotted as a line across all axis. Each line can then be coloured to highlight a particular response. From this, groupings of similar colours can be used to identify areas of a design parameter that have a particularly influential effect on the response. An example of a parallel plot is shown in Fig. 4.2 which shows the *Iris* dataset displayed as a parallel plot [30]. The Python module *plotly* is used for a parallel plot visualisation [53].

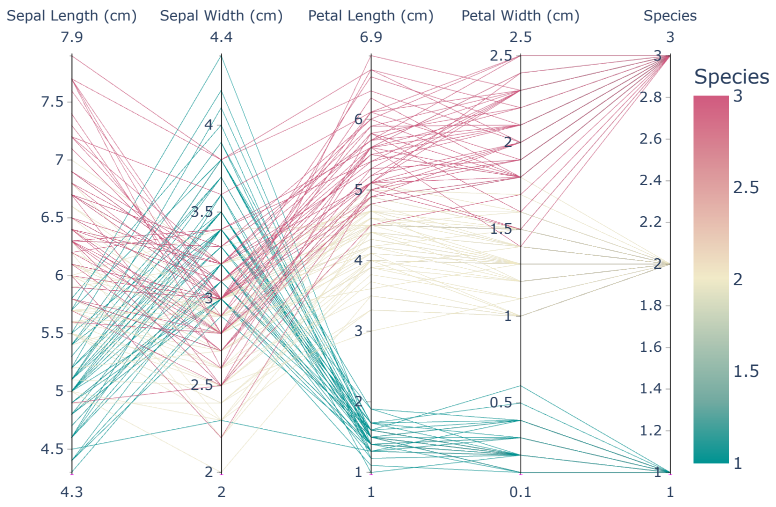


Figure 4.2: Example parallel plot of the *Iris* dataset [30]. This benchmark dataset is commonly used to demonstrate and visualise methods in data analysis.

However, it is not always clear from parallel plots how effectively a new untested design would perform. Given the complexity of handling high-dimensional data sets, it can be difficult to qualitatively locate optimal values for any of the design parameters.

4.1.5.2 t-SNE Dimensionality Reduction

Dimensionality reduction can be used as an effective method of visualising data for high-dimensional data. t-SNE is one such method and is used here to reduce the number of dimensions to 2 while maintaining the classification of the higher-dimensional data points. This results in designs with similar parameterisations being grouped together in the t-SNE space. Dimensional reduction is applied to the design space and is calculated independently of the design response. The plot can then be coloured accordingly by each design response. If a new design for which the response has not been tested were included within this dimensionality reduction calculation, it would provide an indication of its performance based on its proximity to other designs that have been tested. However, this analysis is not able to quantitatively determine the optimum design; it is simply used a useful data visualisation method that can visualise the grouping of the sampled dataset and provide early information about how a new designs performance would fit in that group. The calculation of the t-SNE reduction is performed using the Python module *scikit-learn* [76].

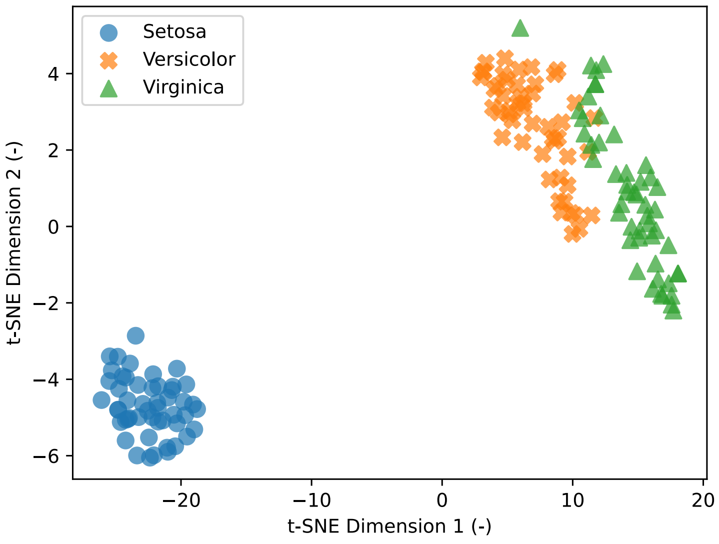


Figure 4.3: Example t-SNE visualisation of the *Iris* dataset [30]. Data points are coloured according to their species. The values along each axis are dimensionless and have no physical representation. This benchmark dataset is commonly used to demonstrate and visualise methods in data analysis.

Fig 4.3 shows an example of a 4-D data set reduced to 2-D. The 4-D data set comprises the widths and lengths of the sepal and petal widths and lengths from the *Iris* dataset. t-SNE makes it possible to visualise the multidimensional data set in a human-comprehensible format and provides some insight into the clustering of the data sets. If a new unknown iris petal was introduced to the data set and was found to cluster in the same region as all other Setosa petals, then this would suggest that the unknown new petal is also of Setosa species.

4.1.5.3 Kernel Density Estimation

To visualise multidimensional response surfaces, 1-D slices are taken through each design point of the trained model to view the model. To create the slices, a large number of random sample points are taken within the design space, for which the predicted response from the surface is calculated, and a KDE model is produced for this data set. KDE is a non-parametric statistical method that is used to estimate the continuous probability density function of a random variable by creating a smooth representation of the underlying data distribution from a subset of known values from the distribution without assuming a specific functional form. The widely used Python visualisation module *matplotlib* contains a function that calculates the KDE using the Gaussian kernel [51].

4.2 Skylon Optimisation

The Skylon geometry introduced in Chapter 1.3, along with the parameterisation defined in Section 1.4.2 is used as a test case to demonstrate the DOE methodology from Chapter 3.

The design space was constructed as described in Section 1.4.2. A sample size of 60 was selected to create 10 geometries for every parameter (5 dependent and 1 independent). An RBF network was trained to fit the trained data using the Wendland C_2 basis function and the shape parameter is calculated as half of the maximum Euclidean distance between the design parameters in $\mathbb{R}^5$. The sensitivity of the RBF parameters is also discussed.

It is not known apriori what the exact angle required to constrain the lift of a new geometry. Therefore, each geometry is solved at a range of angles and the required angle is interpolated from this. There is generally a linear or quasi-linear correlation between incidence and lift coefficient, leading to interpolation of this value being a reliable method of determining the required angle. We confirm this in Fig. 4.4 where the interpolated angle of attack is tested and the calculated C_d is shown to closely match the predicted C_d.

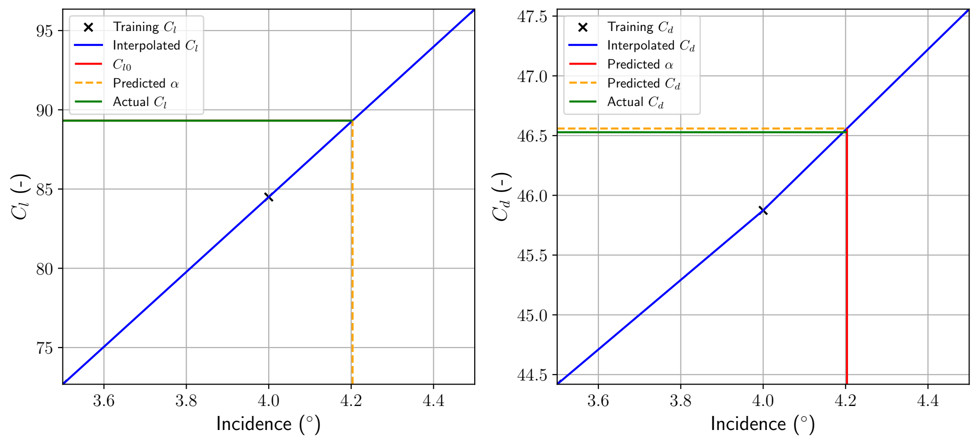


Figure 4.4: Example of lift constraint method for a morphed geometry. The lift and drag values are interpolated (blue solid lines) between calculated incidences (black crosses). C_{l_0} is then used to interpolate the expected incidence at which this new design will produce this C_l. This predicted angle is then used to predict the C_d value at this point, which is confirmed with the calculated C_d at this angle (green solid line)

4.2.1 Training Data Set

Fig. 4.5 shows a parallel plot of the data set with 60 samples, calculating 3 solution fields for each geometry and constraining the lift for each design. The plot is coloured according to its percentage change in drag from the baseline, which is also included in the final column. This graph highlights the variation of the performance of these designs from marginally better than that of the baseline (0.4% lower) to significantly worse (23.8% higher). It is to be expected at this stage that the majority of randomly generated designs will perform worse than the baseline, as the Skylon C2 design has already undergone manual optimisation based on the expert knowledge of the original designers.

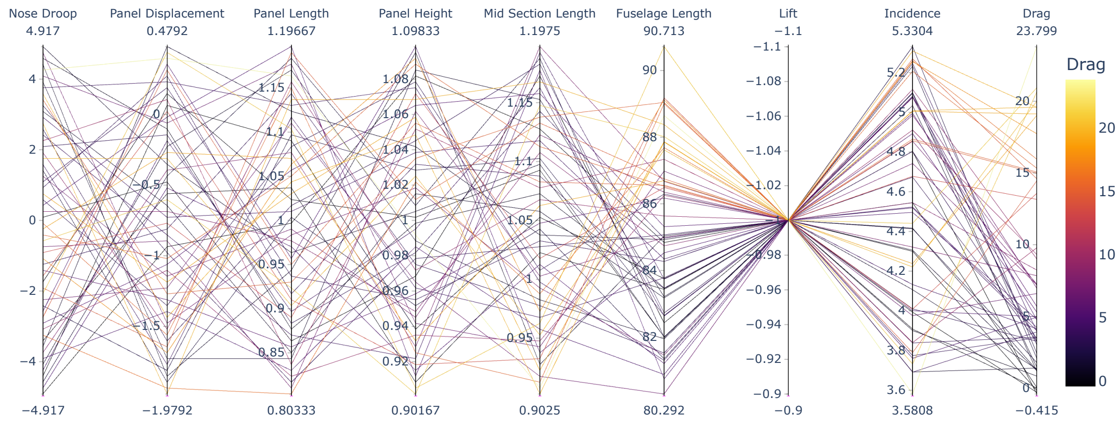
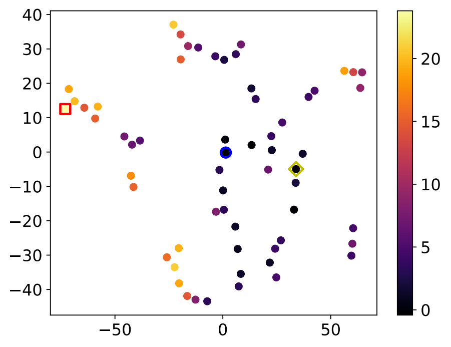
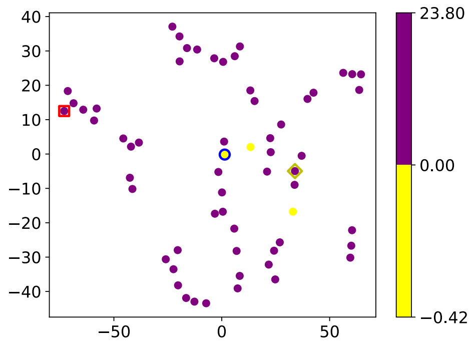


Figure 4.5: Parallel plot of the training data set for $n = 60$.

Fig 4.6a shows a t-SNE plot for the same dataset, where again the percentage change in the drag coefficient is used to colour each point. Fig 4.6b shows the same data set, except a discrete colour scale is used depending on whether the percentage change is positive or negative. The discrete colouring clearly highlights groupings of designs with a lower C_d than the baseline.



(a) Continuous colour scale



(b) Discrete colour scale

Figure 4.6: t-SNE plots for the training data set for $n = 60$. The baseline design is highlighted with a yellow diamond, the worst design with a red square and the best with a blue circle. Both plots have the same coordinates coloured using different scales. The continuous scale is the percentage difference in C_d relative to the baseline. The discrete scale is yellow for designs with lower drag than the baseline, and purple for those greater or equal.

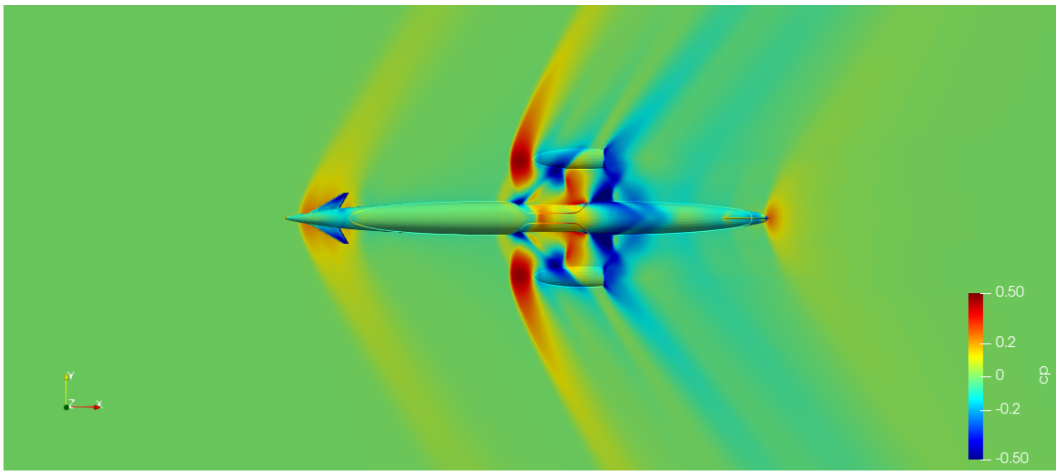
4.2.2 Dataset Aerodynamic Analysis

Before exploring the optimisation procedure, the solution fields and shapes of the best and worst performing designs from the data set are briefly analysed to assess why they may have performed as such. Both geometries were solved again at the angle of attack predicted to constrain the lift. Table

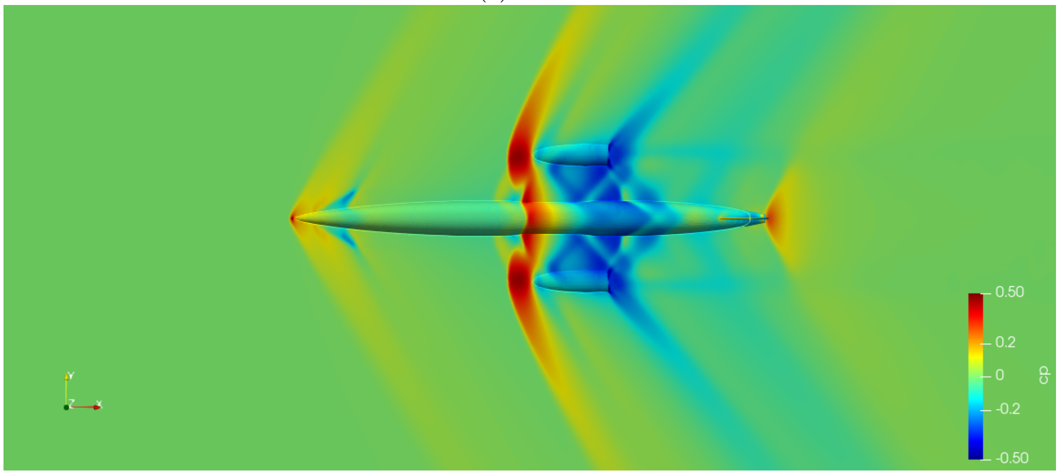
2 in Appendix Appendix B shows the design parameter values for each design. From this data, it can be qualitatively summarised that the worst performing design:

- Significant downward bend of the nose.
- Significantly brings the central side panels in.

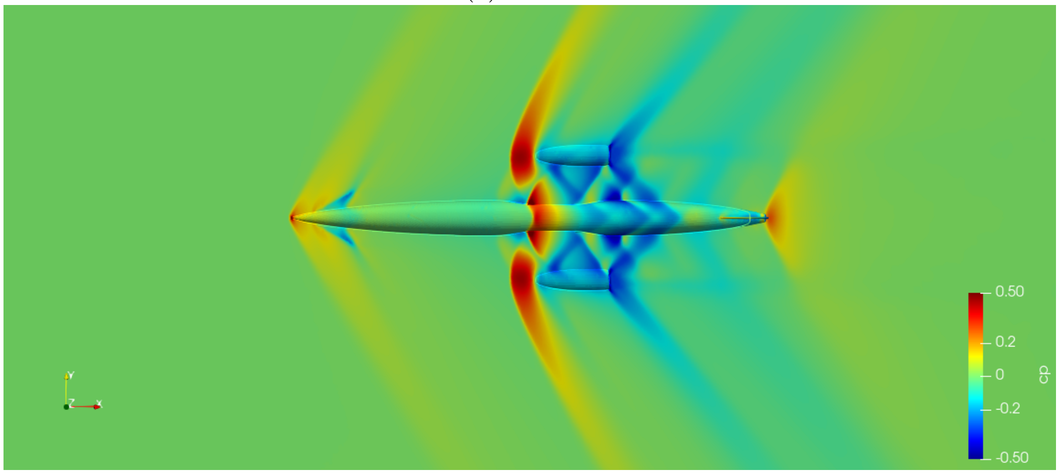
The ultimate effect of the side panel impression is that the complex shock system over the wing changes dramatically. Fig. 4.7 shows a top down view of the worst geometry from the doe set, the baseline geometry and the best geometry from the doe training set. The worst geometry constructs a very different shock system over the wings. The low pressure regions have completely broken down and a high pressure region appears on top of the wing, including a strong low pressure region directly behind them. The poor performance of this geometry is further demonstrated in Fig 4.8a which shows that impressing the panels as significantly as they have in the worst design results in a vortex forming over the wing.



(a) Worst

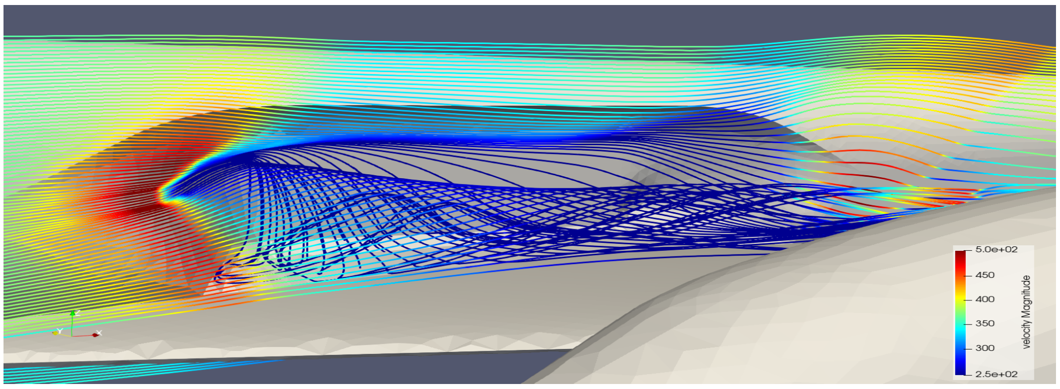


(b) Baseline

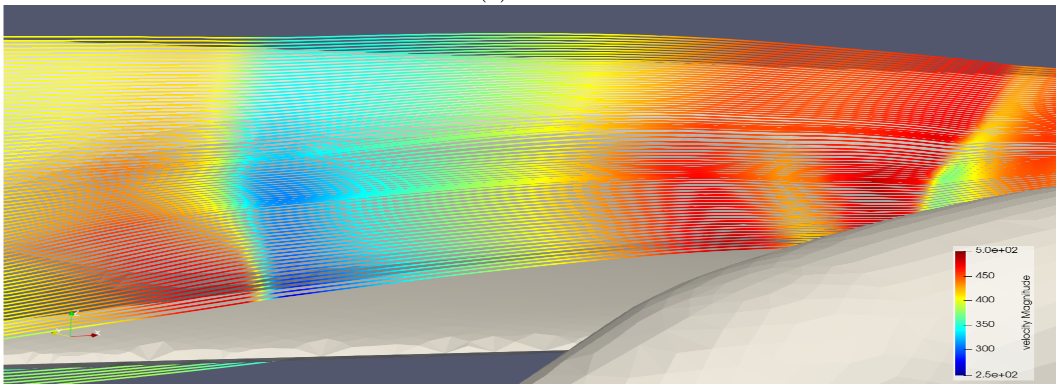


(c) Best

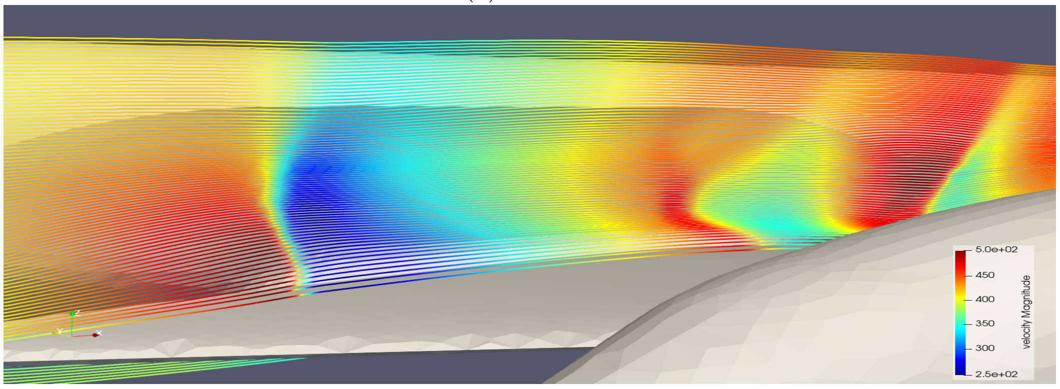
Figure 4.7: Top view of the slice through the plane $z = 0$ across the solution fields for the baseline Skylon design, the best design from the DOE data set and the worst design from the DOE dataset.



(a) Worst



(b) Baseline



(c) Best

Figure 4.8: Streamlines past the side of the geometry for the solution fields of the baseline Skylon design, the best design from the DOE data set and the worst design from the DOE dataset.

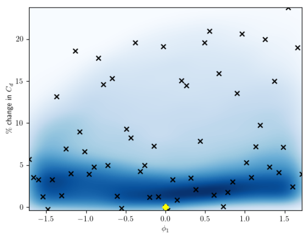
In comparison, the best design from the dataset mainly affects the size and strength of the pressure regions and has little effect on the shock formation positions and pressure regions compared to the baseline design. The streamlines over the wing in Fig 4.8c also show that like the baseline design, no vortexes emerge above the wing caused by the pinched section. In contrast to the worst performing design, the two main qualitative features that stand out for the best design are:

- Slight downward bend of the nose.
- Slightly brings the central side panels in.

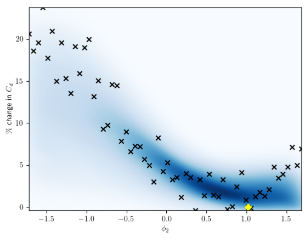
4.2.3 Optimisation

The training data set is used to train an RBF network which is minimised to identify an optimised geometry. A CFD solution is then generated for the optimised geometry to determine the lift constrained drag for the optimised design.

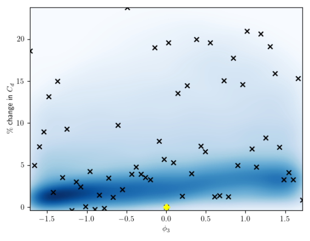
The RBF network was trained using the Wendland C_2 basis function and a support radius of 3.84. This value was determined through empirical analysis by comparing the KDE plot of the fit with the training dataset. These plots show areas where the RBF function is more heavily weighted for each individual parameter, independently of the other design parameters. Consider the second design parameter, ϕ_2, the fuselage pinch. Within the scaled domain, regardless of the value of all other design parameters, any value $\phi_2 > 0$ has a high chance of producing a design with a drag coefficient no greater than 5% higher than that of the baseline. Alternatively, for $\phi_2 < 0$, it makes little difference what the other design parameters are - the drag coefficient is likely not to perform better than the baseline. If the same comparison is made for each of the other parameters, no strong correlation can be made, suggesting that ϕ_2 has the most consequential effect on the drag coefficient.



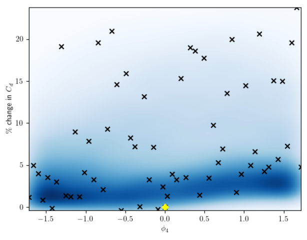
(a) ϕ_1- Nose droop



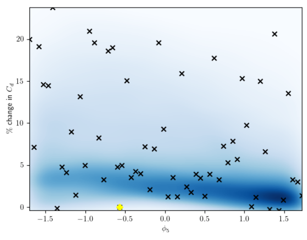
(b) ϕ_2- Fuselage pinch



(c) ϕ_3- Central panel stretch (x)



(d) ϕ_4- Central panel stretch (z)



(e) ϕ_5- Central section stretch

Figure 4.9: KDE plot visualising 1D slices through the RBF response surface fit to the training data set. Training data points are included as black crosses. The x-axis is plotted using the scaled design parameter space, and the y-axis the percentage change in drag relative to the baseline for that design.

The optima for this RBF model was then identified using *differential_evolution* function as described in Section 4.1.4.

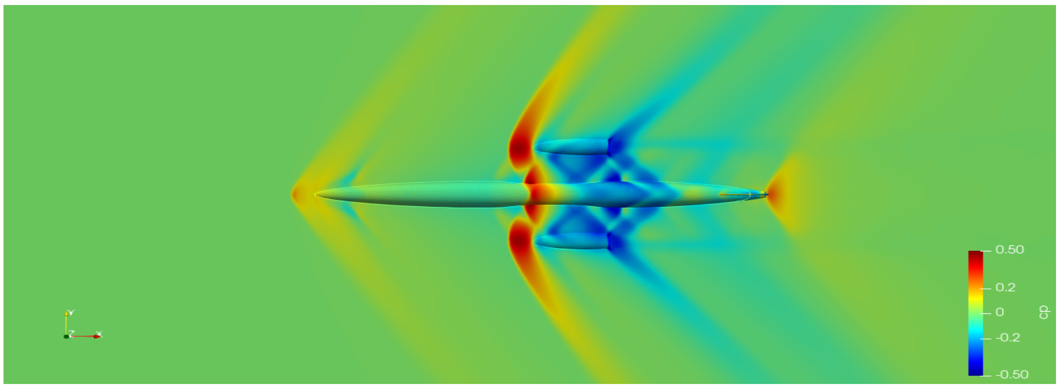
Once the hyperparameter values for the RBF regression model produced a design that potentially had a lower C_d than the baseline, a complete CFD analysis of the geometry was carried out. CFD analysis of this design confirms its expected performance. The design parameters resulting from this are listed in Table 4.1. The predicted drag coefficient for this design based from the RBF model is 1.05%, and the interpolated lift constrained C_d following the CFD tests is 1.62% at an angle of 4.143^o. To confirm the accuracy of the interpolation scheme used to calculate the lift constrained drag, the flow field is calculated at this angle, which resulted in a percentage reduction in C_d of 1.66%. Fig. 4.10 shows the flow field solutions for this geometry where it is evident that this optimised design makes similar changes to the pressure field as the best design from the sampled dataset did.

Table 4.1: Design parameter values for suggested optimised design from minimisation of the RBF response surface trained using 60 sample points.

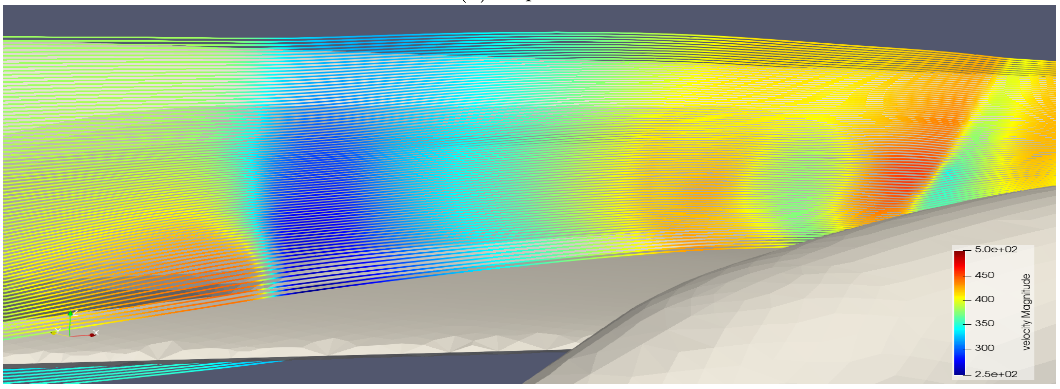
Parameter	Value
ϕ_1	-0.970
ϕ_2	-0.268
ϕ_3	0.851
ϕ_4	0.985
ϕ_5	1.197

Table 4.2: Percentage change in drag coefficient from the baseline value of the optimised design from the minimised RBF response surface. The predicted value is the predicted response from the model and the actual value is the CFD calculated value.

Name	Value
Predicted %	1.05
Actual %	1.62
α	4.1436



(a) Top View



(b) Streamlines

Figure 4.10: Solution field of the DOE optimised geometry showing a top down view of the C_p and the streamlines past the pinched section of the geometry.

4.2.4 Off-Design Performance

The performance across Skylon's flight regime should be checked to ensure the performance gained on design at a specific Mach value does not result in a reduction in performance at other Mach numbers. To check this, the solutions for the optimised design at each Mach range tested in Fig. 4.11 were generated. The same FLITE3D solver is used, however no work is done to explore how different turbulence models or dissipation scheme parameters effect the solutions within the subsonic or supersonic regimes.

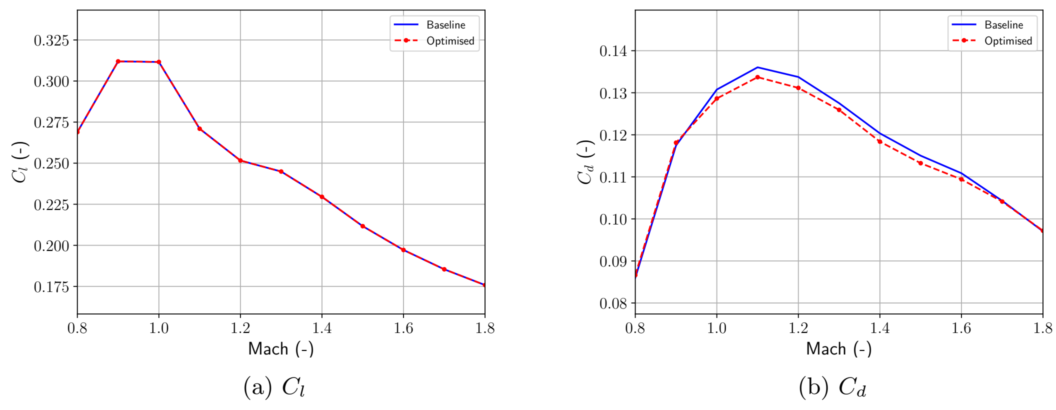


Figure 4.11: Lift and drag coefficients of the Skylon C1 spaceplane in the transonic regime. Optimised design compared against the baseline design.

Similar drag reduction is seen at other transonic values. As the subsonic regime is approached, the drag performance falls closer to the base line design. However, it is worth noting within this regime that the residual for these tests is not fully converged. This is due to the instability of the regime. As a result, the error bars at the Mach values ≤ 1.0 are significantly larger to capture the peaks and trophs of this fluctuation. As the graph enters the supersonic regime, the C_d again approaches the baseline design, and eventually increases above that of the baseline. Across the transonic regime, a reduction in drag is shown and outside of this regime there is very little detriment to the drag. As already discussed, no work was performed to configure the solver for these different regimes, and the same volume mesh was used for each test. As a result, there is likely some error introduced to the supersonic test cases from the discretisation scheme used. Overall, despite the level of approximation in the results, this check provides sufficient information to give an indication that the performance gain has not been severely impacted by this optimisation.

4.3 Summary

In this chapter, the mesh morphing framework introduced in Chapter 3 was used to enable the design optimisation of a spaceplane, including geometric constraints. The DOE method is a simple effective way of exploring the design space, however, there were two key issues with it's effectiveness for this case study. The first is the large number of design points to construct the training data set and second, the hands on approach to ensuring an appropriate fit to the design space. However, these limitations do not prevent an optimal design being determined. Overall, a design with a 1.6% reduction in drag coefficient at Mach 1.2 was located.

Chapter 5

Bayesian Optimisation

Bayesian optimisation is a sequential global optimisation method that guides the selection of evaluation points by considering both the predicted optimal sampling point and the uncertainty within the surrogate model [55]. The algorithm aims to balance exploration and exploitation to quickly converge on the global extrema (minimum or maximum, depending on the definition of the objective function) and is effective when the cost function is expensive to evaluate.

The only property of the objective function that is necessary is that it must be calculable at any point within the domain. No knowledge of the gradient or smoothness of the function is required; in fact, no analytical function is required. This type of function is defined as a Black-Box Optimisation (BBO) function. Using CFD as an analysis tool can be considered as a black-box function that takes a geometry and flow parameters as inputs and returns coefficient values (eg lift and drag). Often, CFD analysis can be very computationally expensive, in the order of hours or days. In particular, the viscous compressible equations that need to be solved when the freestream is in the transonic regime, fall into this category. To even capture simple flows within this regime, the mesh density required close to the surface to capture the boundary layer and in the free stream to capture the shock formation can be large, resulting in high computational cost. This makes Bayesian Optimisation (BO) a good candidate for optimisation when using CFD within the evaluation function to solve compressible viscous flows.

In this chapter, the background and mathematics of BO is covered. Then, the methodology for how the mesh morphing framework in Chapter 3 is integrated with a third-party Bayesian optimiser to allow the optimisation of a geometry, followed by a test case. Although the third-party optimiser is a good implementation, it is not sufficient for the scope of this project. As a result, a custom implementation is then presented and compared with the results from the third-party code.

5.1 Background

In Bayesian optimisation, the objective function is modelled using a stochastic process as the surrogate model [81]. Given a function that has been evaluated at a set of test points, the goal is to find an appropriate model to describe the data.

Gaussian Processes are one of the most commonly used stochastic processes used for BO to form the surrogate model. Once the model is built, an acquisition function is calculated on the model to determine where the next sample should be taken. In the remainder of this section, the mathematical background of both concepts is introduced.

5.1.1 Gaussian Process Regression

One of the most commonly used stochastic methods used within BO are Gaussian Processes. A Gaussian Process (GP) is a probability distribution of functions over a function space. A function space is the set of all functions within a domain $\mathbb{R}^d$. For example, consider the domain $x \in [0, 2\pi]$, Fig. 5.1 shows three possible functions that could exist within this domain. There are of course an infinite number of possible functions that could exist within this domain. In this section, a summary of how GP's can be used to model a black-box function, including background of the mathematical tools and assumptions made throughout this process.

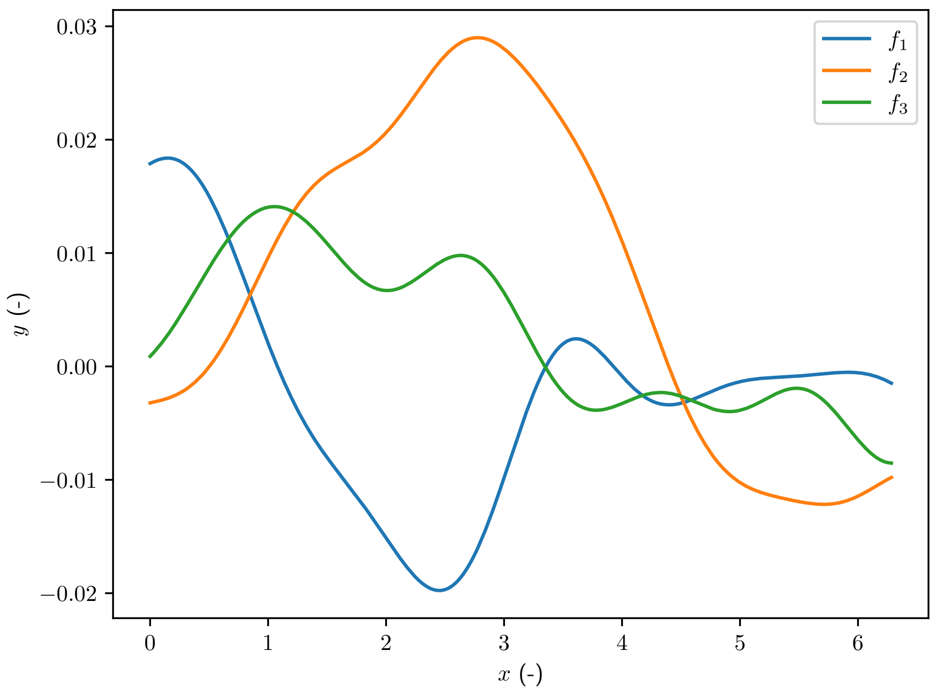


Figure 5.1: Three candidate functions, $\{f_1, f_2, f_3\} \subset \mathbb{F}$, from a function space over a domain $(0, 2\pi) \subseteq \mathbb{R}$. The complete function space is infinite in size, this is simply a snapshot over the domain.

Given a black-box function $f : \mathbb{R}^d \to \mathbb{R}$ over a domain of d dimensions, $\boldsymbol{X} \subset \mathbb{R}^d$ such that $\boldsymbol{y} = f(\boldsymbol{x}) : \boldsymbol{x} \in \boldsymbol{X}$. This is the function that is desired to be minimised. The function f can be used to generate a set of training data points $\boldsymbol{Y_k} = f(\boldsymbol{x_k}) : \boldsymbol{x_k} \in \boldsymbol{X}$. GP regression can be used to fit a predictor function, $\tilde{f}$, to the observed values that can be used to determine a prediction at each unknown point within the domain. The purpose of this is to create a function that is computationally inexpensive to analyse relative to the original black-box function f. This is what leads BO to be an effective optimisation method for computationally expensive functions.

5.1.1.1 Prior

Initially, the function space is constructed without any knowledge of the known values, $\boldsymbol{x_k}$. Each point in $\boldsymbol{X}$ is defined by a random variable that follows a Gaussian distribution. This distribution can be multivariate for cases where $d > 1$. The key concept is that every coordinate in the domain is defined by a random variable. The Gaussian distribution is used to define the random variable here. This leads to the random variable at a value of $x_i \in \boldsymbol{X}$ being defined by a normal distribution such that

$$\tilde{y}_i = \tilde{f}(x_i) \sim \mathcal{N}(\mu_i, \sigma_i)$$

This is to say that the predicted value $\tilde{y}$ at x is drawn from a normal distribution. The distributions for all $x \in X$ can then be combined into a joint distribution. This distribution is defined as the prior over a given domain, Equation 5.1.

$$\boldsymbol{Y}_{prior}|\boldsymbol{X}, \boldsymbol{\theta} = \boldsymbol{Y_0} \sim \mathcal{GP}(\boldsymbol{\mu_0}, \boldsymbol{\Sigma_0}) \tag{5.1}$$

Here, the prior distribution is defined over a domain $\boldsymbol{X}$ and by the parameters $\boldsymbol{\theta}$, which represents the vector for the kernel function hyperparameters used to calculate the covariance matrix $\boldsymbol{\Sigma_0}$. This distribution is completely defined by the mean function such that $\boldsymbol{\mu_0} = \boldsymbol{\mu}(\boldsymbol{X})$ and the covariance $\boldsymbol{\Sigma_0}$.

The mean function at any particular value of x returns the mean of the normal distribution for that value,

$$\boldsymbol{\mu}(x_i) = \mu_i$$

. The covariance matrix is defined by a kernel function k and is a Gram matrix with elements defined as shown in Equation 5.2. Each element of the matrix is the covariance between two points in the domain. Covariance matrices hold information on how variables within the domain relate to each other. Kernel functions can often be controlled by a set of hyperparameters, $\boldsymbol{\theta}$. The significance of these values is discussed in more detail in Section 5.1.1.8. For now, it is important to note that the covariance of the distribution is defined as a function of these hyperparameters. In simpler terms, changing the hyperparameters changes the variance of the distribution.

$$\Sigma_{ij} = cov(\boldsymbol{x_i}, \boldsymbol{x_j}) = k(\boldsymbol{x_i}, \boldsymbol{x_j}, \boldsymbol{\theta}) \tag{5.2}$$

The diagonal of this matrix is the covariance of a point with itself, which is just the variance at that point. This is equivalent to the variance at that value of x,

$$\Sigma_{ii} = \sigma_i$$

. The choice of kernel function can have a significant impact on how well the model fits to the data. The range of suitable covariance functions is discussed in more detail in Section 5.1.1.8.

5.1.1.2 Joint Distributions

The prior has been introduced as a joint distribution of normal distributions, and this type of distribution has some important properties that need to be introduced. In this section, joint distributions are introduced, followed by some of their important properties.

Multiple probability distributions of the same type can be combined to form a joint distribution. Joint distributions can be formed from groups of normal distributions; for example, in Equation 5.3, the random variables y_i are drawn from normal distributions $y_i \sim \mathcal{N}(\mu_i, \sigma_i)$ and form a joint distribution, $\boldsymbol{Y}$. Note that the notation for the column vector $[y_1, \cdots, y_n]^T$ is often written in shorthand as $(y_1, \cdot, y_n)$ when included within the probability to simplify equation layouts.

$$\boldsymbol{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \sim \mathcal{GP}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \left(\begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix}, \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{n1} & \cdots & \sigma_{nn} \end{bmatrix} \right) \tag{5.3}$$

Joint distributions have the property that subsets of the distribution form joint distributions themselves. In the example shown in Equation 5.3 this can be taken to mean that any subset of $\boldsymbol{Y}$ can form a joint distribution itself. Gaussian Process regression leverages two key properties of such distributions: marginalisation and conditionalisation, to fit a surrogate model to a set of known data points.

5.1.1.3 Marginalisation

One of the key properties of joint distributions employed by GP's is marginalisation. The distribution can be separated into subsets of variables such that a subset of the distribution, defined as the marginalised variables, is independent of another subset of variables, defined as the marginal variables. Consider the example in Equation 5.3, define two subsets $\boldsymbol{Y_1} = \{y_1, \cdots, y_m\} \subset \boldsymbol{Y}$ and $\boldsymbol{Y_2} = \{y_{m+1}, \cdots, y_n\} \subset \boldsymbol{Y}$. The joint distribution in Equation 5.3 can be re-expressed as shown in Equation 5.4.

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y_1} \\ \boldsymbol{Y_2} \end{bmatrix} \sim \mathcal{GP}\left(\begin{bmatrix} \boldsymbol{\mu_1} \\ \boldsymbol{\mu_2} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma_{11}} & \boldsymbol{\Sigma_{12}} \\ \boldsymbol{\Sigma_{21}} & \boldsymbol{\Sigma_{22}} \end{bmatrix} \right) \tag{5.4}$$

Each of these subsets is a joint distribution. By setting Y_2 as the marginal variable, it can be integrated out of the distribution, as shown in Equation 5.5, to calculate the marginalised variables, $\boldsymbol{Y_1} \sim \mathcal{N}(\boldsymbol{\mu_1}, \boldsymbol{\Sigma_{11}})$. This distribution is also known as the marginal distribution. The marginalised distributions for the example distribution in Equation 5.4 are shown in Equation 5.6.

$$p(\boldsymbol{Y_1}) = \int_{y \in \boldsymbol{Y_2}} p(\boldsymbol{Y_1}, y) p(y) d\boldsymbol{Y_2} \tag{5.5}$$

$$\begin{aligned} \boldsymbol{Y_1} &= \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \sim \mathcal{GP}(\boldsymbol{\mu_1}, \boldsymbol{\Sigma_{11}}) = \left(\begin{bmatrix} \mu_1 \\ \vdots \\ \mu_m \end{bmatrix}, \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1m} \\ \vdots & \ddots & \vdots \\ \sigma_{m1} & \cdots & \sigma_{mm} \end{bmatrix} \right) \\ \boldsymbol{Y_1} &= \begin{bmatrix} y_{m+1} \\ \vdots \\ y_n \end{bmatrix} \sim \mathcal{GP}(\boldsymbol{\mu_2}, \boldsymbol{\Sigma_{22}}) = \left(\begin{bmatrix} \mu_{m+1} \\ \vdots \\ \mu_n \end{bmatrix}, \begin{bmatrix} \sigma_{m+1,m+1} & \cdots & \sigma_{m+1,n} \\ \vdots & \ddots & \vdots \\ \sigma_{n,m+1} & \cdots & \sigma_{nn} \end{bmatrix} \right) \end{aligned} \tag{5.6}$$

This marginal distribution, $\boldsymbol{Y_1}$ is a joint distribution that ignores any contribution from $\boldsymbol{Y_2}$. This property makes the distribution focus on a particular set of variables while integrating out the influence of other variables.

5.1.1.4 Conditionalisation

Where marginalisation is the probability distribution in the absence of one or more variables, conditioning is the probability distribution given the occurrence of another variable. Marginalisation can be intuitively thought of as the removal of data from the distribution, where conditionalisation can be thought of as the extension of data to a distribution.

A conditional probability can be defined with respect to either the joint distribution or Bayes' theorem can be recovered, both shown in Equation 5.7, for the conditional distribution of $\boldsymbol{Y_1}$ given the known sample points $\boldsymbol{Y_2}$. In this equation, the conditional distribution is a restriction of the distribution over $\boldsymbol{Y_1}$ given a set of realisations from $\boldsymbol{Y_2}$ were observed.

$$p(\boldsymbol{Y_1}|\boldsymbol{Y_2}) = \frac{p(\boldsymbol{Y_1}, \boldsymbol{Y_2})}{p(\boldsymbol{Y_2})} = \frac{p(\boldsymbol{Y_2}|\boldsymbol{Y_1})p(\boldsymbol{Y_1})}{p(\boldsymbol{Y_2})} \tag{5.7}$$

For the joint distribution in Equation 5.4, the conditional distribution of $\boldsymbol{Y_1}$, given the observed values $\boldsymbol{Y_2}$, is given by Equation 5.8. This can be thought of as "the probability Y_1 occurs, given that Y_2 has occurred".

$$\begin{aligned} \boldsymbol{Y_c} &= \boldsymbol{Y_1}|\boldsymbol{Y_2} \sim \mathcal{GP}(\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c}) \\ \boldsymbol{\mu_c} &= \boldsymbol{\mu_1} + \boldsymbol{\Sigma_{12}}\boldsymbol{\Sigma_{22}^{-1}}(\boldsymbol{Y_2} - \boldsymbol{\mu_2}) \\ \boldsymbol{\Sigma_c} &= \boldsymbol{\Sigma_{11}} - \boldsymbol{\Sigma_{12}}\boldsymbol{\Sigma_{22}^{-1}}\boldsymbol{\Sigma_{21}^{T}} \end{aligned} \tag{5.8}$$

The conditional mean, $\boldsymbol{\mu_c}$, can intuitively be considered to be the mean function of the unsampled points, offset by the weighted difference between the observed values and the expected values at the observed locations based on the mean function, $\boldsymbol{\mu}$. This weight is calculated using the cross-covariance matrix and the inverse covariance matrices $\boldsymbol{\Sigma_{12}}$ and $\boldsymbol{\Sigma_{22}^{-1}}$ respectively. Also of note is that the conditional variance, $\boldsymbol{\Sigma_c}$, is not dependent on the mean functions or the calculated values at the observed points. It is simply given as the Schur complement of the known variables in the covariance matrix of the joint distribution.

Functions drawn from a conditioned distribution will be weighted towards the observed dataset $\boldsymbol{Y_2}$. An example of this is shown in Fig. 5.2. The prior distribution used to generate the functions in Fig. 5.1 is conditioned by a single observed value. The resulting functions drawn from this distribution all pass through the sampled point.

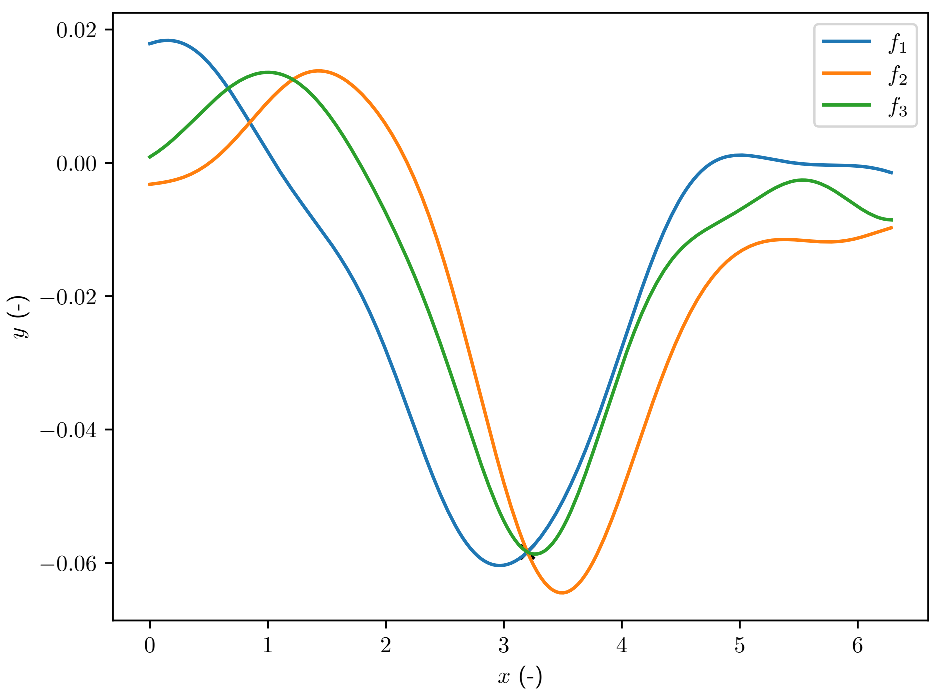


Figure 5.2: Sampled functions that have a high probability of passing through the sampled data point, denoted with a black cross.

5.1.1.5 Posterior

When some points within the domain are known, known as realisations, a posterior over the domain can be established. The points sampled are defined by the vector $\boldsymbol{Y_k}$ and determined by calculating the values of the expensive function $f(x)$. These points are still modelled as random variables such that

$$\boldsymbol{Y_k} \sim \mathcal{N}(\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})$$

In this distribution, the mean is defined by the same mean function as the prior, and the covariance is defined by the same kernel function, however, unlike the prior, the sample of the random variables $\tilde{y}_i$ are replaced by the realised values y at x. The joint distribution formed from this is given in Equation 5.9.

$$\begin{bmatrix} \boldsymbol{Y_0} \\ \boldsymbol{Y_k} \end{bmatrix} \sim \mathcal{GP}\left(\begin{bmatrix} \boldsymbol{\mu_0} \\ \boldsymbol{\mu_k} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma_0} & \boldsymbol{\Sigma_{0k}} \\ \boldsymbol{\Sigma_{k0}} & \boldsymbol{\Sigma_k} \end{bmatrix} \right) \tag{5.9}$$

The posterior is then calculated by conditioning the prior distribution over the known observations using Equation 5.8. Let $\boldsymbol{Y_1} = \boldsymbol{Y_0}$, and $\boldsymbol{Y_2} = \boldsymbol{Y_k}$, then the posterior function, given in Equation 5.10 comes directly from Equation 5.8.

$$\begin{aligned} \boldsymbol{Y}_{post}|\boldsymbol{X},\boldsymbol{\theta} = \boldsymbol{Y_0}|\boldsymbol{X},\boldsymbol{\theta},\boldsymbol{Y_k} &\sim \mathcal{GP}(\boldsymbol{\mu_c},\boldsymbol{\Sigma_c}) \\ \boldsymbol{\mu_c} &= \boldsymbol{\mu_0} + \boldsymbol{\Sigma_{0k}}\boldsymbol{\Sigma_k^{-1}}(\boldsymbol{Y_k}-\boldsymbol{\mu_k}) \\ \boldsymbol{\Sigma_c} &= \boldsymbol{\Sigma_0} - \boldsymbol{\Sigma_{0k}}\boldsymbol{\Sigma_k^{-1}}\boldsymbol{\Sigma_{k0}^T} \end{aligned} \tag{5.10}$$

The mean from this posterior distribution can be used as a predictor function for the given dataset, and the variance at any given point provides an indication of the certainty of that prediction. This function models the original expensive function however is significantly cheaper to compute at any given point within the domain. Conditionalisation ensures that random samples drawn from the distribution pass through the values at known points, as in Fig. 5.2, and as the distance from these points increases, the variance weighting increases to that of the prior distribution.

5.1.1.6 Realisations With Noise

All examples until this point have assumed that the known values of $f(x)$ have been noiseless. When sampling real-world functions, this is not always an assumption that can be made. Noise can be captured in the system as an additional distribution. Let the realisations

$$\boldsymbol{Y_k} = f(X) + \xi\boldsymbol{I}$$

where $\boldsymbol{I}$ is the identity matrix, and ξ is a random variable $\xi \sim \mathcal{N}(0,\theta_n)$. The noise variance, θ_n can be captured as an additional hyperparameter in the vector $\boldsymbol{\theta}$. The addition of this changes the distribution of the random variable over the known sample points to

$$\boldsymbol{Y_k} \sim \mathcal{N}(\boldsymbol{\mu_k},\boldsymbol{\Sigma_k}+\xi\boldsymbol{I})$$

, which modifies the conditional distribution in Equation 5.8 to Equation 5.11.

$$\begin{aligned} \boldsymbol{Y_c} = \boldsymbol{Y_0}|\boldsymbol{Y_k} &\sim \mathcal{GP}(\boldsymbol{\mu_c},\boldsymbol{\Sigma_c}) \\ \boldsymbol{\mu_c} &= \boldsymbol{\mu_0} + \boldsymbol{\Sigma_{0k}}(\boldsymbol{\Sigma_k}+\boldsymbol{\xi I})^{-1}(\boldsymbol{Y_k}-\boldsymbol{\mu_k}) \\ \boldsymbol{\Sigma_c} &= \boldsymbol{\Sigma_0} - \boldsymbol{\Sigma_{0k}}(\boldsymbol{\Sigma_k}+\boldsymbol{\xi I})^{-1}\boldsymbol{\Sigma_{k0}^T} \end{aligned} \tag{5.11}$$

5.1.1.7 Sampling Multivariate Normal Distributions

A sample, $\boldsymbol{X_S}$ from a multivariate normal distribution of n dimensions can be taken using the mean and covariance matrix, μ and Σ, to transform an uncorrelated sample. This is achieved computationally by taking the Cholesky decomposition of the covariance matrix, $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$, and calculating a correlated sample using Equation 5.12. Decomposition is commonly required in computational matrix calculations due to increased speed and stability [81]. Here $\boldsymbol{X_R}$ represents an uncorrelated set of random samples taken from a multivariate normal distribution with zero mean and unit variance.

$$\boldsymbol{X_S} = \boldsymbol{\mu} + \boldsymbol{L}\cdot\boldsymbol{X_R} \tag{5.12}$$

This is the method used to generate the functions in Fig. 5.1, where the mean function and covariance were taken from the prior distribution, Equation 5.1. The same can be generated from the posterior distribution, Equation 5.10 where each point sampled will pass through a known data points, Fig. 5.2. In this example, a single realisation is taken to compute the posterior.

5.1.1.8 Covariance Matrix Kernels

Throughout this section, heavy reference has been made to the covariance matrix $\mathbf{\Sigma}$, however a definitive explanation of the kernel functions used to calculate it has not been given. In this section, some of the different kernel functions are discussed, including their properties.

Kernel functions are bi-variate functions that map the two input values in X to a scalar output [81]. This scalar value represents the variance between the two input vectors. A summary of some of the common functions is given in Table 5.1. Kernels can be categorised using a number of definitions. In the following definitions, K refers to functions that can be expressed as univariate variations of the functions k, whose input is derived from the two points x_i and x_j;

- **Stationary** - k is stationary iff it is dependent on $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ only by their difference. $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = K(\boldsymbol{x_i} - \boldsymbol{x_j})$
- **Isotropic** - k is isotropic iff it is dependent on $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ only by the absolute value of their difference. $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = K(|\boldsymbol{x_i} - \boldsymbol{x_j}|)$.
- **Dot Product** - k is a dot product kernel iff it is dependent on $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ only by their dot product. $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = K(\boldsymbol{x_i} \cdot \boldsymbol{x_j})$
- **Non-Degenerate** - k is non-degenerate if it has an infinite number of non-zero eigenvalues.

The properties; stationary, isotropic, and dot product, all relate the way that the input variables are handled. The fourth property, non-degenerate, is a key property the functions should exhibit to be used as kernel functions. The matrix constructed by a degenerate function has a finite rank, and as such is not positive definite. This results in matrices that do not fully capture the variability between variables in the space, which can lead to poorly trained models. As a result of this, the majority of kernels are non-degenerate. There are a limited number of degenerate kernels often found in the literature and these are often included to be used within compound kernels. All the stationary kernels presented in Table 5.1 are also isotropic. It is therefore assumed throughout this work that all stationary kernels are also isotropic. Note that this list is not exhaustive, for example the periodic kernel and the neural network class of functions are not included [25, 81, 19].

Table 5.1: Summary of some key GP kernel covariance functions considered. Within the kernel functions, $\boldsymbol{\tau} =$ and $\boldsymbol{r^2} = \sum_{i=0}^{d}(\boldsymbol{x} - \boldsymbol{x'})^2/l_d$, where the vector $\boldsymbol{l}$ is a length scale parameter and can be defined separately for each dimension, σ_k is the noise variance parameter and σ_p is the periodicity parameter. Properties of each function are also given: S - Stationary, I - Isotropic, Dt - Dot Product, ND - Non-Degenerate.

Kernel name	**Kernel Function**	**Properties**			
		S	I	Dt	ND
Constant [81]	σ_k^2	×	×	×	×
Linear [25, 19, 81]	$\sigma_k^2 \boldsymbol{x}^T \boldsymbol{x'}$	×	×	✓	×
Polynomial [81]	$(\sigma_k^2 + \boldsymbol{x}^T \boldsymbol{x'})^p$	×	×	✓	×
Squared Exponential [25, 91, 99]	$\sigma_k^2 exp(-\frac{r^2}{2})$	✓	✓	×	✓
RBF [19, 81]	$\sigma_k^2 exp(-\frac{r^2}{2})$	✓	✓	×	✓
Mat 1/2 [91]	$\sigma_k^2 exp(-r)$	✓	✓	×	✓
Mat 3/2 [91, 81]	$\sigma_k^2(1 + \sqrt{3}r)exp(-\sqrt{3}r)$	✓	✓	×	✓
Mat 5/2 [91, 81, 19, 99]	$\sigma_k^2(1 + \sqrt{5}r + \frac{5}{3}r^2)exp(-\sqrt{5}r)$	✓	✓	×	✓
Exponential [19]	$\sigma_k^2 exp(-r)$	✓	✓	×	✓
Gamma Exponential [81]	$\sigma_k^2 exp(-r^\gamma)$	✓	✓	×	✓
Rational Quadratic [81]	$\sigma_k^2(1 + \frac{r^2}{2\alpha})^{-\alpha}$	✓	✓	×	✓

Gamma Exponential The mathematical statement of the gamma exponential function is not commonly found among literature, however it provides a generalised form for multiple functions [81]. When $\gamma = 1$, the function reduces to the commonly presented exponential function and Matérn 1/2 function. When $\gamma = 2$, the function reduces to the squared exponential and RBF functions. These two kernels are identical and the term is used interchangeably in the literature [81]. This form of the gamma exponential function is not immediately obvious as the squared exponential and RBF due to an additional scalar value of 1/2 often included in the exponent, as has been done in Table 5.1 [19, 99, 81]. This does not change the end result as the scaling is counterbalanced by the length-scale parameter during hyper-parameter optimisation. As such, the functions are presented in this work in their more commonly used format, as opposed to the general form of the gamma exponential.

Matérn One commonly used class of functions is the Matérn class, given in their base form in Equation 5.13. The first term includes the Gamma function, a widely used and studied mathematical function. The final term $K_\nu(r\sqrt{2\nu})$ is the modified Bessel function.

$$k(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}(r\sqrt{2\nu})^\nu K_\nu(r\sqrt{2\nu}) \tag{5.13}$$

$$\Gamma\left(x+\frac{1}{2}\right) = 2^{1-2x}\sqrt{\pi}\frac{\Gamma(2x)}{\Gamma(x)} \tag{5.14}$$

$$K_{n+\frac{1}{2}}(z) = \sqrt{\frac{\pi}{2z}}e^{-z}\sum_{k=0}^{n}\frac{(n+k)!}{k!(n-k)!}(2z)^{-k} \tag{5.15}$$

Of particular interest are the positive half-integer cases where $\nu = p + 1/2$ for $p \in \mathbb{Z}^+$. Note that the superscript + indicates non-negative. The three most commonly used functions are Matérn 1/2, 3/2, and 5/2. The derivative of these functions from Equation 5.13 requires some complex mathematical identities. The first useful identity is the Legendre duplication formula, or the doubling formula, Equation 5.14 [36]. The explicit formula for the modified Bessel function, Equation 5.15, is also pivotal to the derivation [36, 2].

For the function Matérn 1/2, set $p = 0$, resulting in $\nu = \frac{1}{2}$. The value for $\Gamma\left(\frac{1}{2}\right)$ can be found using Equation 5.14. Let $x = p$, and cancelling the emergent $\Gamma(1)$ term from each side, leaving $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$. This result is a commonly used identity when using the Gamma function. Taking $n = p$ in Equation 5.15 results in the summation returning 1. Setting $z = r\sqrt{2\nu}$, the Matérn 1/2 function from Table 5.1 emerges from Equation 5.13. This particular case is identical to the exponential kernel, also listed in Table 5.1, itself a specific case of the γ-exponential class.

The second and third functions are derived in a similar way. The Matérn 3/2 kernel is derived by setting $p = 1$, resulting in $\nu = \frac{3}{2}$. $\Gamma\left(\frac{3}{2}\right)$ can again be found using Equation 5.14, and the formula for the Gamma function of a positive integer value, $\Gamma(n) = (n-1)!$, to get $\Gamma(3/2) = \sqrt{\pi}/2$. Again taking $n = p$ in Equation 5.15, the summation now reduces to the rational function $1 + z^{-1}$. Again, the derivation is complete by setting $z = r\sqrt{2\nu}$, substituting all appropriate terms into Equation 5.13 and simplifying the result. The final function of interest, The Matérn 5/2 is derived similarly with $p = 2$. In this case, $\Gamma(5/2) = 3\sqrt{\pi}/4$ and the summation in the Bessel function is $1 + 3z^{-1} + 3z^{-2}$

The interim variables of interest during these derivations is presented in Table 5.2. This class, in particular the Matérn 5/2 function, is a kernel commonly recommended for practical applications [99].

Table 5.2: Interim variables used in the derivation of the important Matérn kernels from the general Matérn class.

Function	p	ν	$\Gamma(\nu)$	$K_v(z)$
Matérn 1/2	0	1/2	$\sqrt{\pi}$	$\sqrt{\frac{\pi z}{2}}e^{-z}$
Matérn 3/2	1	3/2	$\frac{1}{2}\sqrt{\pi}$	$\sqrt{\frac{\pi z}{2}}e^{-z}(1+z^{-1})$
Matérn 5/2	2	5/2	$\frac{3}{4}\sqrt{\pi}$	$\sqrt{\frac{\pi z}{2}}e^{-z}(1+3z^{-1}+3z^{-2})$

5.1.1.9 Automatic Relevance Determination

The stationary kernels in Table 5.1 include a length-scale vector. A single constant value can be used for all dimensions, or different values can be used for each dimension. Taking the inverse of the length-scale exhibits this property that as $l \to \infty$, the influence of that dimension tends to 0. When using different length-scales in each dimension, this process is called Automatic Relevance Determination (ARD) and allows the function to determine the relevancy of each input [72]. Large length scales will effectively remove the variable from the inference [81]. Equally, parameters with small length scales indicate that the parameter has a large impact on the output.

5.1.1.10 Likelihood

An important property of Bayesian optimisation that allows us to determine how well the model fits the data is the likelihood. The exact responses at a set of point is given by the vector $\boldsymbol{Y_k}$. The surrogate model used to fit the data can be used to predict a value at this point, and the **likelihood** of the model is the probability of the model predicting the realised values, given the models current parameters. Notationally, the likelihood is defined as $p(\boldsymbol{Y_k}|\boldsymbol{Y_0}, X, \theta)$. That is to say "what is the probability the value y was recorded, given that the model predicts values in the set $\boldsymbol{Y_0}$ over the given domain X for a given set of hyperparameters θ".

The **marginal likelihood** is calculated from this by marginalising the weights out of the likelihood. Using Equation 5.5, this results in Equation 5.16. This can also be interpreted as integrating the likelihood $\times$ the prior.

$$p(\boldsymbol{Y_k}|\theta) = \int_{y \in \boldsymbol{Y_0}} p(\boldsymbol{Y_k}|y, X, \theta)p(y)d\boldsymbol{Y_0} \tag{5.16}$$

In practise, the log of this probability is taken and maximised, or equivalently, the negative log is minimised. The equation for the log marginal likelihood is given in Equation 5.17 [81]. Note here that the covariance matrix $\boldsymbol{\Sigma_k}$ is the covariance matrix between the known evaluations that includes the noise term.

$$log(p(\boldsymbol{Y_k}|\theta)) = -\frac{1}{2}\boldsymbol{Y_k}^T\boldsymbol{\Sigma_k}^{-1}\boldsymbol{Y_k} - \frac{1}{2}log|\boldsymbol{\Sigma_k}| - \frac{n}{2}log(2\pi) \tag{5.17}$$

Training the model to minimise the negative of the log marginal likelihood is an essential step to using Bayesian optimisation. The surrogate model is built as a probability and maximising the log marginal likelihood ensures that the model fit to the data is the parameters used to formulate the model result in predicting the most probable outcome. Without this step, the model is not appropriately fit to the data. During this process, the length scales for each dimension are determined, which is where the relevance of each variable can be determined using ARD.

5.1.2 Acquisition Functions

Once a GP regression model has been trained to the dataset, BO uses this model to determine the next sample point. Acquisition functions, sometimes referred to as infill criteria, are a function of the posterior, $\tilde{f}$ and use it's properties such as mean, variance, probability distribution function, and cumulative density function, to calculate a single objective function based on the expected values and variance at each point [33, 99] . They are used to balance exploration of the domain and exploitation of known regions containing optimal evaluations and often contain parameters to control the weighting between exploration and exploitation [22]. Many functions have been proposed throughout the literature for both single objective and multi-objective optimisation problems [33, 80]. We define and summarise some of the common functions here.

5.1.2.1 Expected Improvement

One of the more common acquisition functions is Expected Improvement (EI) [33]. This method defines an improvement function and calculates the expected value of this function over the domain [99]. The improvement function is defined as, $I(\boldsymbol{x}) = max(\tilde{f}(\boldsymbol{x}) - f_{max}, 0)$ for a maximisation problem or $max(f_{min} - \tilde{f}(\boldsymbol{x}), 0)$ for minimisation. The expected value is calculated using I as $EI(\boldsymbol{x}) = E[I(\boldsymbol{x})]$. The closed form of EI is given in Equation 5.18 where 5.19 transforms the distribution to a standard normal distribution. Φ and ϕ are the cumulative distribution function and probability density function respectively.

$$EI(\boldsymbol{x}) = z(\boldsymbol{x})\sigma(\boldsymbol{x})\Phi(z(\boldsymbol{x})) + \phi(z(\boldsymbol{x})) \tag{5.18}$$

$$z(\boldsymbol{x}) = \frac{\mu(\boldsymbol{x}) - f_{max}}{\sigma(\boldsymbol{x})} \tag{5.19}$$

5.1.2.2 Probability of Improvement

A simpler function that also makes use of the probability distribution from the regressor is Probability of Improvement (PoI) [91, 99]. This function instead is based solely on the probability density function. The intuitive explanation of this function is that it calculates the amount of the probability density function at x that lies above a target point, τ. At points below τ, the PoI will be low, increasing the closer to τ you get. The PoI is defined as the probability of the predicted value at x being greater than τ, $p(\tilde{f} > \tau)$ for maximisation problems, or $p(\tilde{f} < \tau)$ for minimisation problems. The closed form for this function is given in Equation 5.20 where the function $z(\boldsymbol{x})$ is in the same form as used for EI, given in Equation 5.19.

$$POI(\boldsymbol{x}) = \phi(z(\boldsymbol{x})) \tag{5.20}$$

It is common for τ to be selected as the optimal point from the set of currently evaluated points, f_{min} for minimisation problems or f_{max} for maximisation. However, PoI can be overaggressive with exploitation [91]. It has been shown that setting the target to a value slightly above the current optima has a positive impact on the convergence rate in some cases, however due to the value indeterminacy of τ, and it is generally suggested to simply set it to be the current optima [22]. If the target is chosen in this way then the probability becomes $p(\tilde{f} > (f_{max} + \epsilon))$, and Equation 5.19 changes to Equation 5.21

$$z(\boldsymbol{x}) = \frac{\mu(\boldsymbol{x}) - f_{max} - \epsilon}{\sigma(\boldsymbol{x})} \tag{5.21}$$

5.1.2.3 Confidence Bound

An even simpler equation is the Upper Confidence Bound (UCB) and is given as a linear combination of the mean and standard deviation from the regressor. The closed form for this equation is given in Equation 5.22. The parameter β can be changed to affect the balance between exploration and exploitation. When the value of β is small, the algorithm behaves in an exploitative manner. For the case where $\beta = 0$, exploration is eliminated and subsequent sample points are selected purely based on the predicted mean, which can tend the algorithm towards local minima. The opposite is true for large values of β, where the behaviour is more exploratory. High values of β ensure the algorithm minimises predictive uncertainty however can result in slow convergence rates [22] .

$$UCB(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \beta\sigma(\boldsymbol{x}) \tag{5.22}$$

Of course, the formulation in Equation 5.22 provides the upper limit and is useful for maximisation cases. It is equally as useful to subtract the weighted standard deviation from the mean to optimise for a minimisation case. This would then be referred to as a Lower Confidence Bound (LCB).

5.1.3 Methodology

We use the mean and variance from the Gaussian process model within the BO method to balance exploration with exploitation within the BO algorithm. Algorithm 13 provides an overview of the generic optimisation algorithm.

Algorithm 13 Single objective Bayesian Optimisation minimisation problem for d dimensional data of the expensive evaluation function $f : \mathbb{R}^d \to \mathbb{R}$. Here the posterior is trained using N data points before the algorithm sequentially tests new sample points until the termination conditions are met.

```
1: $S_{prior} \leftarrow$ Construct prior.                                   ▷ GP
2: $\boldsymbol{x} \leftarrow$ Sample $N$ training points.                     ▷ LHS.
3: $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$
4: $S_{posterior} \leftarrow$ Condition prior over $\boldsymbol{x}, \boldsymbol{y}$.
5: while termination condition not met do
6:     $AF \leftarrow AcquisitionFunction(S_{posterior})$                      ▷ EI
7:     $x_{new} \leftarrow argmin(AF)$        ▷ Covariance Matrix Adaption Evolution Strategy (CMA-ES)
8:     $y_{new} \leftarrow f(x_{new})$
9:     $\boldsymbol{x} \leftarrow \boldsymbol{x} \cup x_{new}$
10:    $\boldsymbol{y} \leftarrow \boldsymbol{y} \cup y_{new}$
11:    $S_{posterior} \leftarrow$ Update posterior with new $\boldsymbol{x}, \boldsymbol{y}$
12: end while
13: $i \leftarrow argmin(\boldsymbol{y})$
14: return $x[i], y[i]$
```

The posterior is initially conditioned using a sample set of N data points. The initial sample can be determined using any sampling method, LHS sampling is used here to ensure data sets have similar distributions of data compared to the study in Chapter 4. Once the initial posterior is conditioned, the sequentially loops through the process of determining a new sample point, and reconditioning of the joint distribution to include the evaluation of the new point. The next sample point is determined by calculating an acquisition function over the model and locating the functions maxima. This is a new optimisation problem in itself, however the computational cost of evaluating the acquisition function is significantly lower than that of the expensive evaluation function. As such, simple optimisation algorithms can be used to easily find the maxima of this function. Within this study, we use CMA-ES to maximise the function [40].

5.1.3.1 General Assumptions

In real applications of this method, it is likely that no knowledge of the shape of the dataset or it's mean is known. If it is, for example, if it is known that the data set is likely to fluctuate around $\mu(x) = 100$, then the mean function can be set to capture this feature [81]. If however no knowledge is known, the mean function is generally set to $\mu(x) = 0$. The model is then fit to the data using the variance and length scale parameters. Making this assumption simplifies the posterior mean and variance, the reduced form of which is given in Equation 5.23. This is the distribution that is sampled from by the GP implementations discussed within this chapter.

$$\begin{aligned} \boldsymbol{Y_{post}} &= \boldsymbol{Y_0}|\boldsymbol{Y_k} \sim \mathcal{GP}(\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c}) \\ \boldsymbol{\mu_c} &= \boldsymbol{\Sigma_{0k}}(\boldsymbol{\Sigma_k} + \boldsymbol{\eta I})^{-1}\boldsymbol{Y_k} \\ \boldsymbol{\Sigma_c} &= \boldsymbol{\Sigma_0} - \boldsymbol{\Sigma_{0k}}(\boldsymbol{\Sigma_k} + \boldsymbol{\eta I})^{-1}\boldsymbol{\Sigma_{k0}^T} \end{aligned} \tag{5.23}$$

5.1.3.2 Comparison with DOE

The dataset generated in Chapter 4 used 60 data points to train a surrogate model that was then optimised to determine a new optimal data point. This was a highly computationally expensive data set to train. To compare it with an alternative optimisation procedure, the datasets need to be the same size. Therefore, to reduce the computational overhead of generating additional large data sets of this size, a smaller set with 19 training data points and 1 optimised value from the surrogate model was generated, to ensure a meaningful and fair comparison of their performance. The basis function used for the RBF surrogate model was the Matérn 5/2 function to match the kernel used in the GP surrogates used throughout this chapter. The performance for the design generated from this is presented in Table 5.3 and 5.4. This test predicted a reduction in drag but this was not realised in the resulting drag calculation from calculating the drag coefficient using CFD.

Table 5.3: Design parameter values for suggested optimised design from minimisation of the RBF response surface trained using 19 sample points.

Parameter	Value
ϕ_1	0.362
ϕ_2	-0.045
ϕ_3	0.800
ϕ_4	0.972
ϕ_5	1.095

Table 5.4: Percentage change in drag coefficient from the baseline value of the optimised design from the minimised RBF response surface. The predicted value is the predicted response from the model and the actual value is the CFD calculated value. The model predicted a reduction in drag, however the CFD tests calculated an increase.

Name	Value
Predicted %	-1.50
Actual %	0.0157
α	4.284

5.2 Third-Party implementation

To generate an initial set of results to explore the effectiveness of BO, a third-party Python code was used. The GECCO-2017 Bayesian optimisation code was developed to explore the performance of different acquisition functions for multi-objective optimisation [80].

Although the implementation came with a good set of examples, there were a lot of dependency issues that required fixing before it could be used. To list just a few; the code required a suitable c compiler to build a single function written in cython, numerous dependency issues on the *numpy* module, dependency issues with the CMA-ES module, bugs in the code for some methods, issues with array shape for single-objective optimisation, and bugs in the visualisation code [39]. On top of this, documentation was minimal and the software was tricky to run, despite the examples provided. Significant refactoring was also required to enable the expensive evaluation function to communicate with a HPC to run the expensive parallel computations. The target test case for the optimiser is a constrained single-objective optimisation. While GECCO-2017 is capable of this, it's primary purpose is for multi-objective optimisation. As a result, it is an extremely complex code base for single-objective optimisation. It is also capable of running multi-surrogate tests, however, only mono-surrogate models are required for this thesis.

5.2.0.1 Configuration

After all the bugs were fixed and HPC communication was built in, the software could be configured and executed. A basic set of parameters was used to configure the optimisation process. The acquisition function used was EI, which was minimised by building a DEAP toolbox and minimising the evaluation function with CMA-ES [40]. The Python module GPy handled the construction and optimisation of the underlying GP.

5.2.0.2 Results

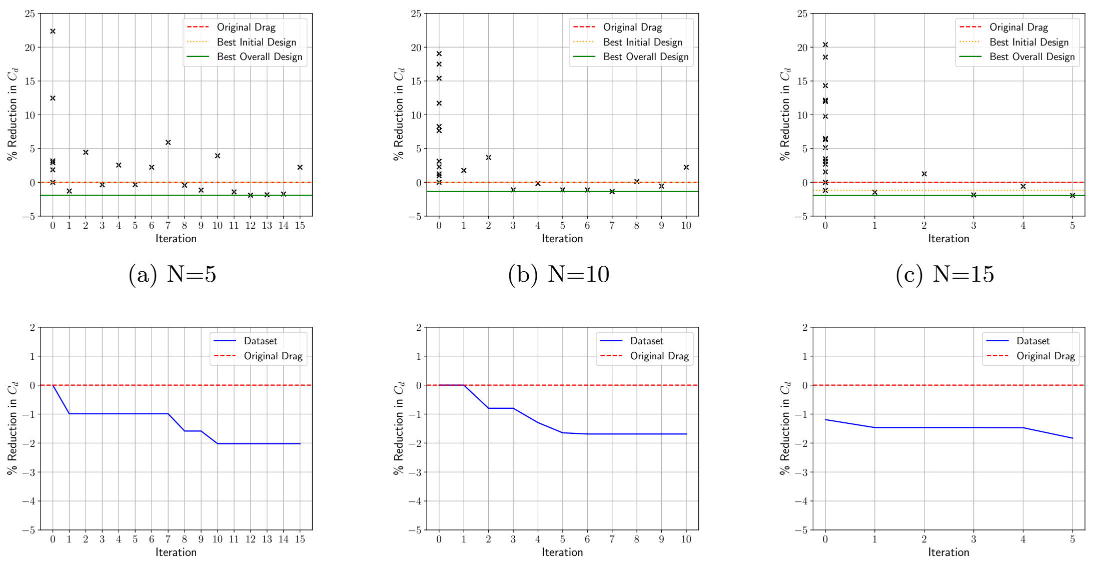
The GECCO-2017 optimiser was used to perform the same drag minimisation problem introduced in Section 1.4. The goal of using BO for optimisation is to produce a design with similar or better performance than using the DOE, however using reduced computational resources. The initial, static, training data can be calculated in parallel and the sequential, dynamic, calculations are calculated individually. Therefore, the size of the initial training set is varied to explore how the method reacts; however, the total number of cost function evaluations is performed in each case.

Table 5.5: Results for spaceplane optimisation using Bayesian optimisation with 3 different sizes of initial training data for 20 total function evaluations, compared with results from the DOE study from Chapter 4 and Section 5.1.3.2. The percentage reduction from the baseline is given for the best geometry from that test, as well as the iteration number that design occurred on.

Test	% Reduction
DOE (60)	1.6
DOE (20)	-0.0015
5	1.908
10	1.357
15	**1.942**

Table 5.5 shows the maximum percentage reduction in C_d between the initial training data. Three sizes of initial training data are explored, 5,10 and 15 and the total number of cost function evaluations for each test is 20. Each initial training data set was sampled using LHS with the same seed.

This is a similar magnitude of reduction as the DOE study from Chapter 4 using just 1/3 of the number of function evaluations. This is also a significant improvement when using a comparable number of training data points, as the DOE study was unable to predict an optimised design with the same number of sample points.



(a) N=5 (b) N=10 (c) N=15

(d) N=5 (e) N=10 (f) N=15

Figure 5.3: Drag search histories of Skylon optimisation using BO for varying initial training data set sizes, N, with the convergence history for each graph is shown beneath each search history. A maximum of 20 function evaluations were completed in each test. Iteration 0 contains all the initial training data points, and each subsequent iteration represents the next suggested sampling point from the Bayesian optimiser. The horizontal red bar marks C_d of the baseline design, the orange bar marks C_d of the best design from the initial training data (baseline design included), and the green bar marks C_d of the best overall design.

The search history of all three of these studies, using different sizes of initial training data, is shown in Fig. 5.3. The trade-off between exploration and exploitation can be seen in the search histories of all 3 plots, where those with limited information in the initial training set explore more than exploit.

Test Set N = 5 The first test set used 5 initial training data points and completed 15 sequential iterations. Observing the performance of the sample set of initial training data in Fig 5.3a, none of the sampled designs out performed the baseline design. Despite this, the first new sample point from the acquisition function of the Bayesian surrogate resulted in a reduction in C_d. With only six evaluation functions, a design was found that gave a reduction of -1.282%.

Test Set N = 10 The second test using 10 initial training data points did not perform as well as the other two test sets. While multiple designs with lower drag than the baseline were identified, none were better than the result from the DOE test.

Test Set N = 15 The final test set of 15 samples and 5 iterations yielded the best overall result from all tests completed.

5.3 Custom Implementation

The third party implementation of a Bayesian optimiser produced good results in the given case study; however, the implementation was hard to digest due to its complexity, was difficult to setup and install, and included many features that were not used or required. As a result of this, and issues of intellectual property with regard to the ownership of the output from this work, a new implementation of a Bayesian optimiser was produced.

The new implementation had a significantly reduced code base while allowing for easy extension of novel kernels and acquisition functions to be tested. It also uses fewer non-standard Python library dependencies. The support functions used to set up the algorithm are the only difference between the two methods. The kernels, acquisition function, log marginal likelihood, and optimisation procedure used are mathematically the same. All of these attributes make the new implementation significantly more favourable for use in industry applications, a key target of the deliverable outputs from this thesis. One drawback is that while multi-objective optimisation can theoretically be supported in the new implementation, it is currently untested.

In the remainder of this section, the configuration and testing of the new implementation are discussed. The model is compared with that of the third-party model previously used and is used to generate more results for the Skylon optimisation problem.

5.3.0.1 Configuration

Throughout the optimisation procedure, the optimisation of two inexpensive to evaluate functions must be made; the log likelihood and the acquisition function. No properties of these functions are guaranteed, such as whether they are differentiable or not. Therefore, it is suggested to use gradient-free optimisation methods. Based on this, the performance of the Nelder-Mead and CMA-ES algorithms is compared for both applications [73].

The Nelder-Mead algorithm is implemented in the Python module *scipy*. It has configurable parameters that control the initial value, maximum number of iterations, adaptive sampling, and convergence tolerance.

The CMA-ES algorithm is implemented as a stand-alone Python module. It has a wide range of configurable parameters, of which only the maximum function evaluation value and whether the BIPOP extension is turned on/off are explored [39]. The public Python module is forked and included within the software suite to allow for improved debugging and reduce the number of software dependencies.

5.3.0.2 Rigour

To verify the accuracy of this implementation, the values of the new implementation are compared with those of the GECCO-2017 optimiser. The kernel parameters after hyperparameter optimisation and the next suggested sample point from each data set are compared between each optimiser. While this process is stochastic in nature, for the same given training data and kernel function, minimisation of the log marginal likelihood should result in similar hyperparameter values in both implementations.

The third-party implementation uses the Python module *GPy* for the Gaussian process and kernel. One fundamental difference between the two methods is that when minimising the log marginal likelihood, the *GPy* module includes the noise variance of the GP as a hyperparameter, where the custom implementation does not. It was decided not to include the GP noise variance in the parameter optimisation of the custom implementation as it was recognised that in all test cases, the resulting GP noise variance from hyperparameter optimisation was set as the lower bound provided. This is because only one function evaluation is given at each known sample location; therefore, the hyperparameter optimisation has no built-in concept of how much error there may be in each function evaluation. Reducing this parameter to the smallest possible value results in the optimal fit for the model under these conditions.

During the optimisation of computationally expensive functions in this thesis, only one function evaluation is made at each point x, to ensure that resource is spent efficiently exploring the design space. Therefore, the GP noise variance is left as a user-configurable parameter with a small default value of $1e-8$ in the custom configuration. In cases where the user may have an idea of how much error the function evaluation may have, this allows engineers to specify the noise variance. For example, if the CFD solution does not fully converge, but fluctuates about a mean value, then the amplitude of the fluctuation can be used to specify the noise at each value of x.

The hyperparameters from the GECCO-2017 code and the custom implementation for the test case using 15 initial training data points are compared in Table 5.6. The configuration that produced the closest matching parameter values was the *scipy* module's Nelder-Mead method. The method was configured with an initial x-value of $1 * ndim$, adaptive sampling turned on, and a maximum of 10000 iterations (converged after < 5000 in all tests). This achieved the kernel parameters shown in Table 5.6. This table also shows the converged parameter values without adaptive sampling. Even after continuing the optimisation for over 100000 iterations, the algorithm did not identify a set of parameters with a lower negative log-likelihood. These hyperparameter values match very closely to those from the GECCO-2017 code.

In Section 4.2.3, it was recognised that changing parameter ϕ_2, the fuselage pinch, had the largest impact on the drag coefficient. Across all 3 BO implementations in Table 5.6, the length scale for this parameter is lower than the other 4, highlighting that it has the biggest impact on drag and confirming the original conclusion.

Table 5.6: Comparison between hyperparameter values for the GECCO-2017 and custom optimisation code after log marginal likelihood optimisation. The GP was configured using the Matérn 5/2 kernel, and the initial training data set with 15 data points was used to condition the posterior.

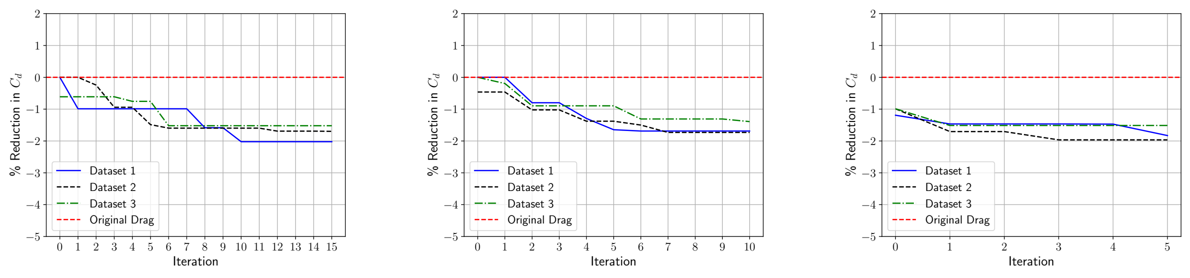
Parameter	Values		
	GECCO-2017	Custom	Custom (adaptive off)
Kernel Noise variance	3.337	3.649	1.304
Length scale 1	3.746	3.747	45912518.152
Length scale 2	2.082	2.073	1.083
Length scale 3	673815.024	28202146.588	62642046.428
Length scale 4	681354.425	42736273.302	11.700
Length scale 5	23.671	23.720	10.036

To find the next sampling point from the acquisition function, the function maximum is located. The CMA-ES implementation used is a minimisation function, so the objective function. Similarly to the GECCO-2017 implementation, this method was configured with an initial x value set to the mid range of the design space, $\sigma = 0.25$ and a maximum number of function evaluations of 10000.

5.3.0.3 Results

To further confirm the accuracy of the custom implementation, the results of Section 5.2.0.2 are reproduced. The same training data, kernel, and acquisition function are all used. Despite the best effort to keep the two methods comparable, Bayesian optimisation is an inherently stochastic method due to the probabilistic nature of the surrogate model used. On top of this, different optimisation methods are used in the hyperparameter optimisation and acquisition function maximisation, which due to the complex landscape of these functions, lead to slight differences in calculated values.

The convergence histories between both implementations differ, however both drive the results towards similar performance gains. This is confirmed in Fig. 2 which shows the repeated results for each size of the initial training data set. Furthermore, each test has been repeated 3 times. The search history graphs for these tests is included in Appendix C. Table 5.7 summarises the best result from each test and includes the mean reduction from each data set.



(a) N=5 (b) N=10 (c) N=15

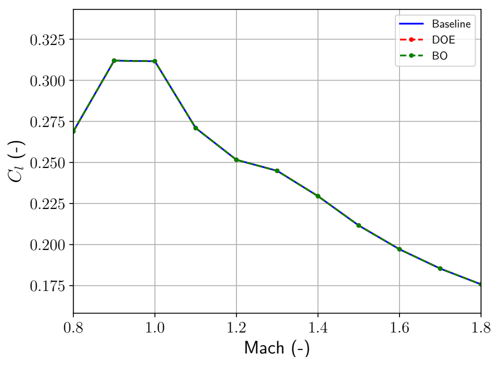
Figure 5.4: Convergence histories for the test cases using the custom BO implementation. Each test uses different sizes of initial training data set, N, and is repeated 3 times using different seeds for LHS sampling and surrogate modelling.

Table 5.7: Results for spaceplane optimisation using the custom implementation of Bayesian optimisation with 3 different sizes of initial training data for 20 total function evaluations, compared with results from the DOE study from Chapter 4 and the comparable DOE dataset from Section 5.1.3.2. The percentage reduction from the baseline is given for the best geometry from that test, as well as the mean reduction across repeats of the same test with identical parameters, but a different seed.

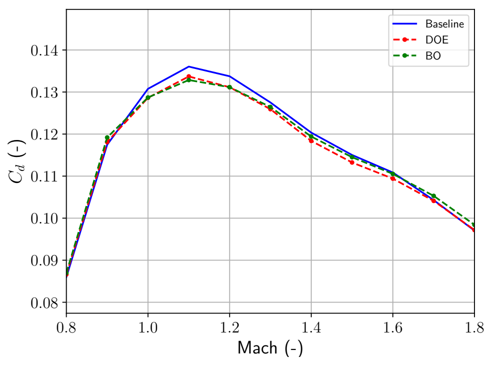
Test	% Change (Min)	% Change (Mean)
DOE (61)	-1.600	-1.600
DOE (20)	+0.016	+0.016
5	$\mathbf{-2.022}$	-1.747
10	-1.732	-1.603
15	-1.965	-1.769

5.3.0.4 Off-Design Performance

The optimal design from Chapter 4 was compared against the baseline across the transonic Mach range and slightly into the supersonic regime. The optimised design from this chapter, the geometry from iteration 9 of the custom implementation of the optimiser with 5 initial training data points, is compared in the same way, Fig 5.5. It is clear from this that this design results in similar improvements as before - reducing the drag coefficient in the transonic regime and equalling or slightly reducing the drag coefficient into the supersonic regime.



(a) C_l

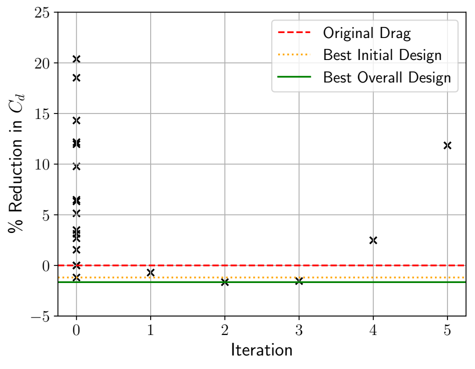


(b) C_d

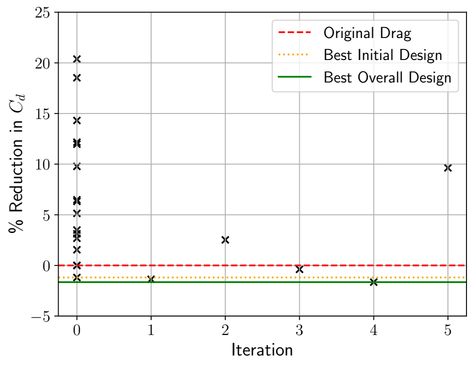
Figure 5.5: Mach sweep comparing the baseline design with DOE and BO optimised designs.

5.3.0.5 Parameter Sensitivity

The BO method has a number of parameters that can be changed depending on the problem. In particular, the kernel and the acquisition function. A brief sensitivity analysis of the effect of these parameters on the optimisers search history is performed. Due to the computational expense of running the tests, the data set with 15 initial training data points is used for the repeated tests, limiting the number of iterations required for each test to 5. The search history of each test is given in Fig. 5.6. For the first test, the PoI acquisition function was used instead of EI. The expected behaviour of this function is that the optimisation method should explore the domain less and exploit areas that are expected to perform well. For the second test, the maximum number of function evaluations allowed by the optimiser that minimises the log marginal likelihood and acquisition function was increased.



(a) POI



(b) EI - 50000

Figure 5.6: Search history graphs for BO tests investigating the effect of different parameters on the search histories. Fig 5.6a uses the PoI acquisition function and Fig 5.6b uses the EI function, the same used in previous tests, but the CMA-ES optimiser uses 50,000 function evaluations to determine the next sample point.

Table 5.8: Results for spaceplane optimisation using the custom implementation of Bayesian optimisation for three different BO parameter sensitivity studies.

Test	% Reduction	Iteration	Mean	σ
DOE	1.6	1	1.6	0.0
EI - 50000	-1.652	1	2.085	5.107
POI	-1.650	3	1.750	4.209

Table 5.8 shows the percentage decrease in drag of the best designs from each test which are all comparable to the reduction seen from using other acquisition functions in Table 5.7. However, the different functions don't find any larger reductions in drag, and also don't appear to explore the domain much differently. From comparing the results from different configurations of optimisation methods used within the optimisation method, it is suggested that different methods and configurations be used for new, unseen problems to compare the outcome.

These results demonstrate Bayesian optimisation is able to locate optimised designs without requiring careful tuning of the optimiser. When training a model to fit the data in the design of experiments study, the hyperparameters for the model had to be manually tuned to locate an optimised design, and the model was highly sensitive. The Bayesian optimiser does not appear to suffer similarly, locating an optimised design with minimal effort from the user to configure the method.

5.3.0.6 Computational Cost

The results in this section have so far been compared against the results of the DOE study from Chapter 4. Both studies found comparable reduction in the drag coefficient, with the Bayesian method slightly outperforming the former study. One metric that should be considered between these studies is computational cost to determine the optimised designs. In total, 60 test samples where generated to create the sample set of data for the DOE study. Comparatively, the Bayesian tests saw a reduction after using a total of 20 objective function evaluations. Further to this, many of the Bayesian tests identified the optimal design with fewer objective function evaluations than this, using the additional evaluations to explore untested areas of the design space. In total, 1/3 the total number of objective function evaluations were made by the Bayesian optimiser to return approximately 1/3 more reduction in C_d. Table 5.9 shows a more detailed break down in how much the C_d was reduced per objective function evaluation that was performed.

Study	Num Tests to Optimal	% Reduction	% Reduction Per Test
DOE	61	1.600	0.026
BO (N=5)	20	**2.022**	0.101
BO (N=10)	20	1.687	0.083
BO (N=15)	20	1.830	0.092

Table 5.9: Comparison of C_d reduction per objective function evaluation between different studies. The final column shows the percentage reduction found divided by number of objective function evaluations performed.

5.4 Summary

The results from the Skylon optimisation, in both implementations of a Bayesian optimiser, reinforce the notion that the BO-based optimisation performs well in settings where the total number of function evaluations is a strict constraint. In particular, the method holds up well as the size of the initial training data is reduced. All of this implies that where wall clock time is a constraint and computational resource is available, increasing the size of the initial training dataset can lead to a more reliable convergence rate. On the other hand, in cases where computational resource is limited and large numbers of parallel solutions are not possible, similar performance reduction can be found by letting the optimisation run for an increased number of iterations.

Another benefit of using this optimisation method over the DOE is that the exploration is much more targeted, locating better designs, in less time, with less input from the user. These features make it a highly attractive method for optimising expensive cases such as the Skylon transonic drag coefficient reduction study.

Chapter 6

Use of Area Ruling for Design Optimisation

One of the key objectives of this thesis, introduced in Section 1.5, was to "Investigate and evaluate different configurations of the proposed method". In this chapter, the methodology is applied to a similar optimization problem, but using a different computational approach to assess the performance of different designs. The optimisation methodologies described in Chapters 4 and 5 demonstrated how the aerodynamic performance of a spaceplane in the transonic regime could be optimised by using mesh morphing to explore a design space and CFD to calculate aerodynamic coefficients. These methods worked well, resulting in a reduction in C_d, however, each CFD solution was computationally expensive to produce. In this chapter, the hypothesis that the transonic area rule can be used as a cheaper alternative to CFD that still drives the optimisation scheme towards a design with a similar reduction in drag coefficient. This demonstrates how the mesh morphing methodology in Chapter 3 can be flexibly applied to different problems, and is not explicitly tied to CFD based problems.

This chapter provides a summary of the wave drag approximation in the transonic and supersonic regimes, and outlines how it is used to construct an evaluation function for each geometry. The implementation is then discussed, including notes on the considerations made to the numerical treatment of the discretisation of the analytical theory. Justification for why it was believed that this alternative method could provide similar results to the CFD study is given, and the chapter concludes with the results of the optimisation study which are compared with those of the optimisation studies that used CFD solutions in the evaluation function. The results from this study suggest that for the same optimisation problem introduced in Section 1.4, using the alternative evaluation function was not capable of driving a geometry towards a similar shape as when using CFD to determine the performance of the geometry.

6.1 Background

There has been extensive research both theoretically and empirically on the relationship between longitudinal area distribution and wave drag within the transonic and supersonic regimes [56, 112, 7, 37, 89].

Overall drag within these regimes becomes highly sensitive to the cross-sectional area distribution of the geometry along its length. This section begins by discussing some of the mathematical formulae that describe this relation, followed by summarising the minimum wave drag body that results from these formulae.

6.1.1 Transonic Area Rule

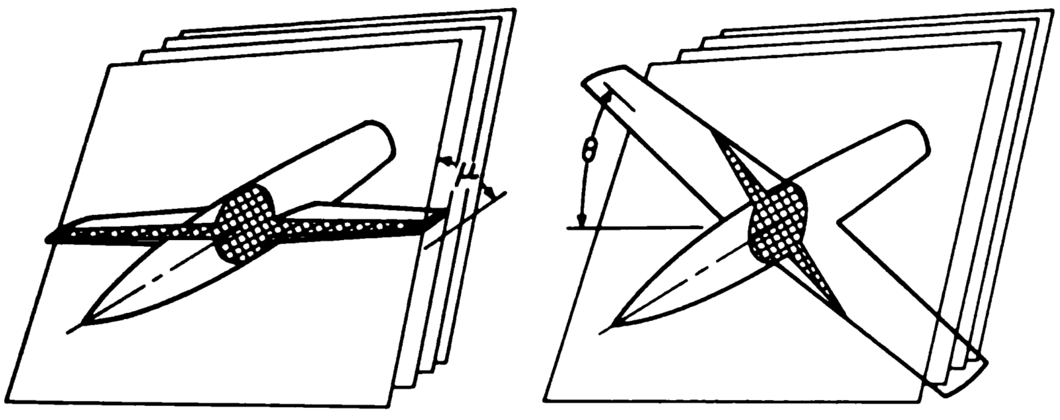
The area-rule was introduced by Whitcomb, often termed Whitcomb's transonic area rule. A relationship between the drag-rise profiles of complex geometries and bodies of revolution was observed, and Whitcomb presented a generalised theory that the drag rise observed was driven by the development of the cross-sectional area distribution along the length of the geometry. The theory was investigated using wind tunnel experiments for a range of bodies in which the smoothness of the area distribution for a wing-body geometry had been optimised to give a substantial drag reduction [112]. The empirical tests of bodies around transonic speeds from Whitcombs work demonstrated the significant drag reduction by cutting or wasting of the body of a wing-body geometry to drive the area distribution of it towards that of the original body definition.

The experimental results were given more theoretical consideration and extended to the supersonic regime by Jones [56], where an equation was sought to determine the wave drag as a function of the area distribution of a geometry. The wave drag for a given geometry as $M \to 1$ is given by Equation 6.1 where $S(x)$ represents the cross-sectional area of the geometry at a location along the length of the body, x. The integral is scaled by the free stream conditions, where density (ρ_∞) and velocity ($\boldsymbol{u_\infty}$) are given in Equation 6.1 . The cross-sectional area is calculated from the intersection between the geometry and a plane perpendicular to the geometry reference line (the same line used to measure the angle of attack).

$$D = -\frac{\rho_\infty \boldsymbol{u}_{\boldsymbol{\infty}}^2}{4\pi} \int_0^l \int_0^l S''(x_1) S''(x_2) ln|x_1 - x_2| dx_1 dx_2 \tag{6.1}$$

This theory is valid when the intersection plane is perpendicular to the free stream, however, is only valid within the transonic regime around the speed of sound. The theory can extended into the supersonic regime by angling the intercepting plane when calculating the area distribution. The plane is first angled relative to the x axis by an angle μ_m, a function of the Mach number (Equation 6.2), and then rotated around the x axis by an angle ψ. An example of the plane inclination at angle μ_m from Jones' original work is shown in Fig. 6.1a [56].

$$\mu_m = arcsin(1/M) \tag{6.2}$$



(a) Inclined Plane (b) Rotated Plane

Figure 6.1: Visualisation of the intersecting plane applied to a simple wing-body geometry. In Fig. 6.1a, the plane is inclined at the Mach angle μ_m relative to the freestream. Fig. 6.1b shows the subsequent rotations of the plane at an angle ψ around the roll axis. The parallel planes along the length of the geometry are shown behind the highlighted plane in each case. Figures from [44]. Note that Harris uses θ to denote the rotational angle, however ψ is used throughout this thesis to avoid confusion with θ as the coordinate transformation variable.

The plane rotated by angle ψ is used to calculate the wave drag following the same method as before, but now as a function of the angle of revolution around the x-axis of the inclined plane,

$$D(\psi) = -\frac{\rho V^2}{4\pi} \int_0^l \int_0^l S''(x_1, \psi) S''(x_2, \psi) ln|x_1 - x_2| dx_1 dx_2$$

. The wave drag at x is then calculated by averaging all values of $D(\psi)$ for $\psi \in [0, 2\pi]$ such that

$$D = \frac{1}{2\pi} \int_0^{2\pi} D(\psi)$$

. An example of this plane rotation is given in Fig. 6.1b from Lomax [64].

For cases where $M = 1$, the angle of inclination $\mu_m = 0$. Thus, area calculation is unaffected by rotation of the plane about the x-axis, and the equations reduce to the original form found in Whitcomb's transonic work.

6.1.2 Minimum Wave Drag Body

From this theory, an interesting consequence is that bodies of minimum drag for given conditions can be determined. The theoretical area distribution profile for a body of revolution with minimum wave drag for a given volume and length was calculated independently by Sears and Haack [89]. To determine this optimal distribution, the rate of change of the cross-sectional area is expanded in a Fourier sine series.

$$S'(\theta) = \sum_{n=1}^{\infty} A_n sin(n\theta) \tag{6.3}$$

The coordinate transformation variable θ is used to transform x using the transformation in Equation 6.4 where l is half the length of the body, $0 \leq \theta \leq \pi$ and $0 \leq x \leq l$ [7]. The nose is defined at $\theta = \pi$ and the base at $\theta = 0$.

$$x = \frac{l}{2}(1 + cos(\theta)) \tag{6.4}$$

The cross-sectional area at the nose reduces to 0 while at the base, $S(x) \geq 0$. For example, $S(x)$ may be non-zero in the case of a munition or projectile, or equal to 0 resulting in a body that is pointed at both ends. The cross-sectional area of a geometry with respect to θ can be calculated by integrating Equation 6.3 with respect to x, along the length of the geometry. This integral is calculated by transforming the differential from dx to $d\theta$, calculating the indefinite integral this and using the boundary condition on the nose, $S(\pi) = 0$, to determine the integration constant.

$$S(\theta) = \frac{l^2}{4} A_1(\pi - \theta + \frac{sin(2\theta)}{2}) + \sum_{2}^{\infty} A_n(\frac{sin(\theta(n+1))}{n+1} - \frac{sin(\theta(n-1))}{n-1}) \tag{6.5}$$

This can be taken a step further; integrating the cross-sectional area along the body gives the total volume of the body [7].

$$V = \frac{\pi l^3}{8}(A_1 - \frac{1}{2}A_2) \tag{6.6}$$

Integration of $S'(\theta)$ in Sears formula, Equation 6.1, is made possible by using the Fourier transform of $S'(\theta)$, which results in the formula for the wave drag being a function of the Fourier coefficients, Equation 6.7 [7]. The body with minimum wave drag is then the body that minimises the Fourier coefficients, A_n.

$$D = \frac{\pi q_\infty l^2}{8} \sum_{1}^{\infty} n A_n^2 \tag{6.7}$$

These coefficients can be determined based on the conditions at the nose and base, as well as using other previously derived formulae. For a body that is pointed at both ends, the cross-sectional area at the base is $S(0) = 0$. As $sin(0) = 0$, all the terms in Equation 6.5 for $n \geq 2$ reduce to 0. Therefore, to satisfy the zero cross-sectional area, the first coefficient must be zero, $A_1 = 0$. Taking this value in the equation for the body's volume, for the body to have a non-zero volume, A_2 must therefore be non-zero. For a given volume, this results in

$$A_2 = -\frac{16V}{\pi l^3}$$

. The area distribution of a body where all other coefficients are zero, $A_n = 0, n \geq 2$, describes the body with minimum wave drag for a given length and volume. This is the body that was derived independently by Sears and Haack, and represents the optimum theoretical shape for at least two of a given length, radius or volume [89, 37]. The area distribution for this body is determined by substituting A_2 into Equation 6.5.

$$S(\theta) = \frac{4V}{\pi l}\left(sin(\theta) - \frac{1}{3}sin(3\theta)\right) \tag{6.8}$$

The area distribution can be transformed back into the x coordinate space by rearranging Equation 6.4 into $cos(\theta) = \frac{2x}{l} - 1$ and using the identity $sin(3\theta) = 3sin(\theta) - 4sin^3(\theta)$ to get

$$S(x) = \frac{16V}{3\pi l}\left(4\frac{x}{l}\left(1 - \frac{x}{l}\right)\right)^{3/2} \tag{6.9}$$

For a given length and volume, the body can be determined from this equation. Alternatively, the equation can be represented in terms of the maximum radius of the body. For example, when using a Sears-Haack body as the basis for the geometry of a vehicle, there may be some internal requirements that drive a minimum radius to be defined. The cross-sectional area of a slice through the Sears-Haack body at any x forms a circle and can therefore also be defined as $S(x) = \pi r^2$. The maximum radius R_{max} for this longitudinal symmetric body is located at the midpoint along the length of the body, $l/2$. Using this, Equation 6.9 ca n be rearranged to determine V in terms of maximum radius to be

$$V = \frac{3\pi^2}{16} l R_{max}^2$$

. The reverse of this can define Equation 6.9 in terms of length and maximum radius. An equation for the radius, or thickness, of the body at any station x can also be determined. This equation and its derivatives along the length of the body are given in Equation 6.10.

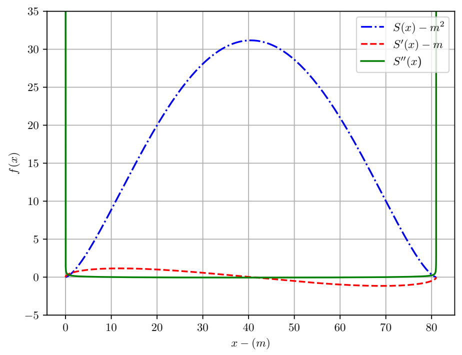
$$\begin{aligned} r(x) &= R_{max}(4x/l(1 - x/l))^{3/4}, \\ r'(x) = \frac{dr}{dx} &= \frac{3R_{max}(1 - 2x/l)}{(4x/l(1 - x/l))^{1/4}}, \\ r''(x) = \frac{d^2r}{dx^2} &= -3R_{max}\left(\frac{(1 - 2x/l)^2}{(4x/l(1 - x/l))^{5/4}} + \frac{2}{(4x/l(1 - x/l))^{1/4}}\right). \end{aligned} \tag{6.10}$$

The derivatives with respect to x for the cross-sectional area as a function of radius are given in Equation 6.11, calculated using the chain rule.

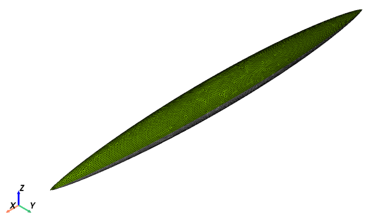
$$\begin{aligned} S(x) &= \pi r^2, \\ S'(x) = \frac{dS}{dx} = \frac{dS}{dr}\frac{dr}{dx} &= 2\pi r r', \\ S''(x) = \frac{d^2S}{dx^2} = \frac{d^2S}{dr^2}\left(\frac{dr}{dx}\right)^2 + \frac{dS}{dr}\frac{d^2r}{dx^2} &= (2\pi r'^2) + (2\pi r r''). \end{aligned} \tag{6.11}$$

6.1.3 Transonic Summary

The Sears-Haack body derived from the wave drag equations is the optimal body shape for a given length and volume in the transonic regime. This geometry often forms the initial fuselage design for many transonic design projects, such as Bloodhound LSR and Skylon. An example of the Sears-Haack body is given in Fig. 6.2b for $R_{max} = 3.15$ and $l = 81$, values that are similar in scale to the radius and length of the Skylon spaceplane's fuselage.



(a) Cross-sectional Area Distribution



(b) 3D Surface Mesh

Figure 6.2: Analytical solution of the area distribution, it's derivatives and example 3D mesh of the Sears-Haack body with parameters $R_{max} = 3.15$ and $l = 81.0$.

Theoretically, bringing the cross-sectional area profile of a geometry as close to that of the Sears-Haack body will reduce the wave drag of the geometry. Using the equations outlined in this section, this can be numerically approximated without the need for CFD. As the drag coefficient for a geometry in the transonic regime becomes dominated by wave drag, reducing the wave drag should result in a reduction in the overall drag.

6.1.4 Evolutionary Optimisation

Evolutionary algorithms are a class of metahueristic algorithms that have become widely adopted methods of optimisation. They can be categorised under the field of bio-inspired optimisation, methods that draw inspiration from nature to mimic biological behaviours [116]. Specifically, Evolutionary Algorithms (EA) replicates the biological mechanisms of evolution, natural selection, and genetics [9]. Evolutionary algorithms often involve making changes to a *population* of n individuals, often referred to as *agents*, and over several generations. Some examples of classifications of EA are evolution programming (EP), evolution strategies (ES), genetic programming (GP), and genetic algorithms (GA) [10, 110]. They follow similar methodologies, aiming to simulate the evolutionary process, however, they differ in their implementations [9]. An exhaustive review of genetic algorithms introduced between 1957 and 1993 demonstrates the rapid growth of the field [3].

One drawback of evolutionary algorithms is that they often require a substantial number of function evaluations to achieve convergence, which can lead to increased computational costs. In studies such as those in Chapters 4 and 5 the computationally expensive cost of each function evaluation made evolutionary algorithms an impractical choice. However, this chapter proposes that a function with significantly lower in evaluation cost is used. As a result, evolutionary algorithms become a more viable option. Particle Swarm Optimisation (PSO) is one such evolutionary algorithm that has been widely used since it's introduction in the 90's [58]. Given its widespread adoption and established efficacy, it was selected as the preferred algorithm for this study.

6.1.4.1 Particle Swarm Optimisation

PSO is a population-based stochastic metaheuristic consisting of some evolutionary properties [58]. The pseudocode for the algorithm is given in Algorithm 14. The algorithm redefines the terminology used compared to some evolutionary algorithms but is synonymous with the standard EA structure. A *population* is redefined as a *swarm* and an *agent* is redefined to a *particle*. The particles are also given a *velocity* property in addition to the *position* property. The best observed location for each particle is tracked as well as the best location of the overall swarm. Each generation, the velocity is updated for each particle, which is then used to move the particle to a new location, Equation 6.12. The velocity is updated through a contribution from the velocity of the particle in the previous generation, local knowledge of the best known position of the particles and global knowledge of the best location of the swarms, Equation 6.13.

Algorithm 14 Psuedocode for PSO algorithm. Each generation, the particle velocity is calculated and used to update the particle positions. After a set number of generations, the best agent from the population is returned.

```
1: $population \leftarrow$ Initialise $N$ agents
2: $evaluate(population)$
3: while generations $<$ max_generations do
4:     Update velocities (Equation 6.13)
5:     $newPopulation \leftarrow$ Update positions (Equation 6.13)
6:     $evaluate(newPopulation)$
7:     $population \leftarrow newPopulation$
8: end while
9: return $max(population)$
```

An in-depth review of recent research surrounding PSO was conducted by [14]. Their work concludes that despite extensive research on novel modifications to the core algorithm, most instances lack theoretical background to suggest specific characteristics of the evaluation function landscape that would warrant the adoption of a particular algorithm variant over others. For this reason, in this thesis, a minor extension of the original implementation is implemented, which was suggested by one of the original authors of the algorithm [92]. In this modification, the velocity of the previous generation is weighted with a coefficient, ω, defined as the inertial weight. Contributions of local knowledge and global knowledge remain unchanged. Local contribution is given as the product of cognitive weight ϕ_1, a uniformly distributed random number in the region $[0, 1)$ such that $r_1 \sim U(0, 1)$, and the difference between the best known location of the particles x_{pb} and the current location x_k. The global contribution is given as the product of the social weight ϕ_2, a random number similar to r_1 such that $r_2 \sim U(0, 1)$ and the difference between the best known location of the swarms x_{sb} and the current location of the particles x_k. Note here that there is nothing in the literature to prevent $r_1 = r_2$, however, the two numbers should be sampled separately. Equation 6.13 is the resulting equation for these three contributing terms.

$$x_{k+1} = x_k + v_{k+1} \tag{6.12}$$

$$v_{k+1} = \omega v_k + \phi_1 r_1(x_{pb} - x_k) + \phi_2 r_2(x_{sb} - x_k) \tag{6.13}$$

A default set of parameters suggested for this algorithm were used that have been shown to be effective for a range of different applications based on substantial mathematical convergence analysis and empirical data [62]. Specifically, the values used were: $\omega = 0.42$ and $\phi_1 = \phi_2 = 1.55$.

6.2 Methodology

Sears formula, Equation 6.1 forms the basis for an objective function that evaluates each geometry based on its area distribution rather than calculating the computationally expensive CFD solution. The wave drag is proportional to the integral of the second derivative of the area distribution squared and so theoretically, a geometry that minimises the magnitude of this integral across it's length minimises the wave drag of the geometry.

6.2.1 Objective Function

The objective function, J, is defined in Equation 6.14 and gives an indication of the *smoothness* of the geometry. The second derivative of the area distribution represents the smoothness of the area distribution. A low value at a given point for the smoothness suggests that there are no large jumps in the area distribution near this point. Reducing the second derivative at any point reduces the overall objective function and so in turn the wave drag of the geometry.

$$J(M) = \int_0^l (S''(x))^2 dx \tag{6.14}$$

For a geometry parameterised with values of $\phi(\phi_1, ..., \phi_n)$, the optimisation problem is $min(J, M(\phi))$. In cases where an optimisation method is only capable of performing maximisation is used, the cost function is negated, equating to a minimisation problem. The optimum geometry for a given baseline and design space is the mesh M with the lowest value for J. The remainder of this section discusses how an optimisation procedure using this objective function is implemented.

6.2.2 Optimisation Procedure

The design space is constructed using a parameterised geometry with the mesh morphing method introduced from Chapter 3 in the same manner as described in the DOE optimisation approach from Chapter 4 and the Bayesian optimisation approach from Chapter 5. Given the low computational cost of calculating the objective function in Equation 6.14 for a given geometry compared to a CFD solution, many more designs can be explored using this optimisation approach. Therefore PSO, an evolutionary optimisation algorithm introduced in Section 6.1.4, is used to explore the design space. When applied throughout this chapter, different population sizes and generation counts are explored depending on the test case.

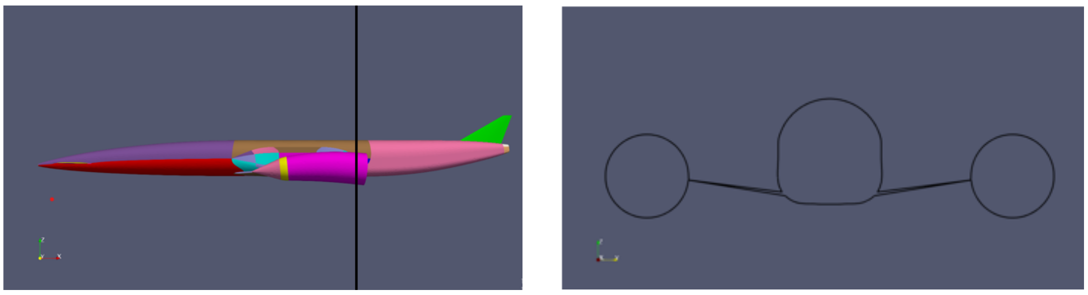
6.3 Implementation

The implementation of the objective function requires consideration with regard to the numerical methods used and how the area distribution at a given point x is determined. In this section, the Python implementation that slices the geometry to determine the area distribution is calculated and the numerical methods used for differentiation and integration are summarised.

6.3.1 Geometry Slicing

To calculate the area distribution of a given surface mesh, the Python module *trimesh* is used to calculate the intersection slices [21]. This module expects the surface to be a triangulation, so any hybrid meshes with quad elements are first passed through the Python module *pyvista*, which is used to triangulate a hybrid mesh when required [104]. The meshes should be watertight for the *trimesh* module to complete. This is a property that meshes must also satisfy to be valid for CFD computation using FLITE3D, so any surface mesh that is valid for CFD analysis is also valid for this method.

The function: *section* is used from the *trimesh* module, which returns the intersection curve between a given mesh and a given plane. The object returned for the intersection curve also provides the area of this intersection slice as one of its built-in properties. This cross-sectional area is calculated along the length of the geometry, constructing the area distribution. Fig. 6.3 shows an example of the slice resulting from a particular station of x along the length of the Skylon geometry.



(a) Side view with plane location. (b) Cross-sectional intersection.

Figure 6.3: Mach plane cutting through the Skylon geometry and the intersection curve between the geometry and the plane.

6.3.2 Numerical Treatment

When calculating the area distributions discretely, numerical methods are required to calculate derivatives and integrals. Some of the numerical methods used to treat each of these cases are described in this section.

Differentiation Second order central difference is used to calculate derivatives of discrete functions, Equation 6.15, where h represents the step size between uniformly distributed values of x. Boundary values are handled and compared using two different methods, extrapolating the domain using a constant and using the forward difference, Equation 6.16, and the backward difference, Equation 6.17, where appropriate.

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{6.15}$$

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} \tag{6.16}$$

$$f'(x) = \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h} \tag{6.17}$$

The same treatment is used for second order derivatives, for which the central, forward and backward difference stencils are given in Equations 6.18, 6.19 and 6.20 respectively.

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{6.18}$$

$$f''(x) = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} \tag{6.19}$$

$$f''(x) = \frac{f(x) - 2f(x-h) + f(x-2h)}{h^2} \tag{6.20}$$

Integration Integrals over one dimension are the only integrals considered in this chapter. Therefore, the trapezium rule is used to approximate the integral. The equation for the calculation of the integral (or antiderivative) F at a point x is given in Equation 6.21, which gives the area under f when calculated over the domain of x.

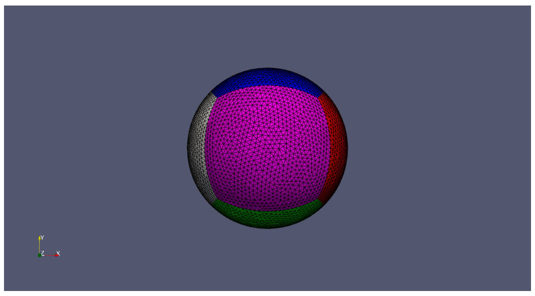
$$F(x) \approx \frac{h}{2}(f(x+h) + f(x-h)) \tag{6.21}$$

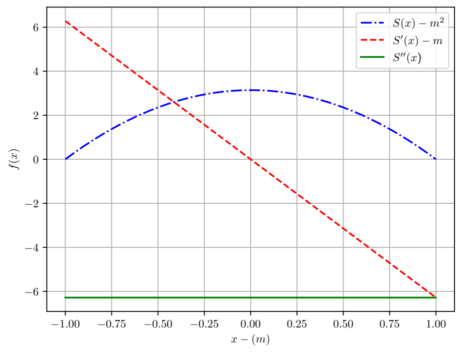
6.4 Test Cases

In the preceding section, the theoretical foundation was laid. The practical implementation of these methods needs to be explored and the numerical stability assessed. In this section, 3 test cases are introduced that are later used to analyse the stability of the numerical methods used in the implementation of an area-distribution calculation. Some of the test cases used in this chapter are the same as those introduced in Section 3.2. Additional details about the analytical solutions for these geometries are included in this section. Some additional cases are also introduced that are of significant relevance to the wave-drag theory introduced in this chapter.

6.4.1 Sphere

The unit sphere introduced in Chapter 3 is used because the analytical area distribution for this geometry is known and easy to calculate. The mesh for the geometry is included in Fig. 6.4a and the geometry is parameterised the same way as described in Section 3.2.2. In addition to this, the analytical area distribution (including derivatives) are shown in Fig. 6.4b.


(a) Mesh



(b) Area Distribution

Figure 6.4: Baseline unit sphere mesh and analytical solution of the area distribution (and derivatives) of a unit sphere.

6.4.2 Sears-Haack Body

A more complex geometry with a more significant relevance to the topic of wave-drag and area distribution analysis is the Sears-Haack body introduced in Section 6.1. This type of geometry is a more realistic shape to expect in aerospace vehicle design. The Sears-Haack body gives the theoretical minimum wave drag for a given length and volume, or given radius at a fixed length. The analytical definition of the area distribution is known for this geometry and well studied. The analytical solutions for the distribution and derivatives are given in Fig. 6.2a. An example of a discretised mesh for the Sears-Haack body is generated by first creating the 1D distribution of points along the x-axis using Equation 6.10 for a given length and the required maximum radius. R_{max} is selected here as 3.15, however, it can also be calculated by defining a required volume and using Equation 6.9 to determine the required radius. This 1D distribution is then imported into a CAD package, and a body of revolution is created. The CAD model is then imported into the FLITE3D CFD suite and a surface triangulation is created with an approximate node spacing of 0.3, Fig. 6.2b. This spacing is approximately the same value in the coarse Skylon mesh used throughout the optimisation studies in Chapters 4 and 5.

6.4.3 Skylon Spaceplane

The Skylon spaceplane, used throughout this thesis, is used as an example of an industry test case. It provides an example of a significantly more complex geometry than the other test cases and is used to explore the effectiveness of the implementation on a complex body, It is used as the test case for the optimisation procedure with the aim of identifying an optimised design for the geometry that is similar to those found in Chapters 4 and 5.

6.5 Sensitivity Study

Numerical methods can be highly sensitive to the choice of step size h. A sensitivity study therefore needs to be performed when applying the numerical methods to area distribution calculations. In this section, the area distribution for the Skylon spaceplane is calculated using the implementation in Section 6.3

6.5.1 Number of Slices

Before using this method for an optimisation cases, the level of discretisation that can be achieved before the numerical error introduced to the derivative calculations grows too much should be investigated. Section 6.4 introduced two test cases that have analytical solutions available for both the first and second derivatives, the sphere and the Sears-Haack body. Meshes of both cases are used to calculate the area distribution plot, using varying numbers of slices, and the stencils described in Section 6.3.2 are used to differentiate these curves. The RMSE from the analytical solution in each case can then be evaluated.

6.5.1.1 Sears Haack

The first geometry analysed is the Sears-Haack body. The first and second derivatives of the analytical area distribution contain a singularity at the boundaries that needs to be considered computationally to avoid zero division errors. When calculating the analytical solution, the singularity in the first derivative is treated by considering the limit. As the singularity is of the form

$$S'(x) = \frac{f(x)}{g(x)}$$

where both f and g tend to 0 as $x \to 0$, then it can be assumed that $S'(x) \to 0$. However, in the case of the singularity for the second derivative, $f(x) \nrightarrow 0$ as $x \to 0$. Therefore, this singularity is handled computationally by replacing the calculated datatype NAN with an arbitrarily large number.

The analytical curves for the Sears-Haack geometry are compared in Fig 6.5 to those calculated using the discrete method from Section 6.2. Fig 6.5a uses 200 slices and clearly shows significant numerical error along the length of the body. Fig 6.5b uses just 20 slices and is qualitatively much closer to the exact solution for the second derivative, except at the boundaries, however is less accurate for the first derivative. This highlights that using too many slices can have a detrimental impact on the second derivative.

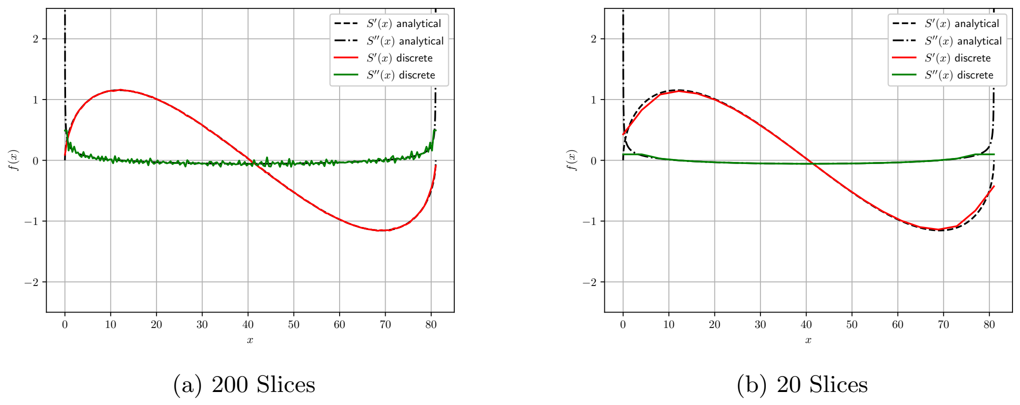
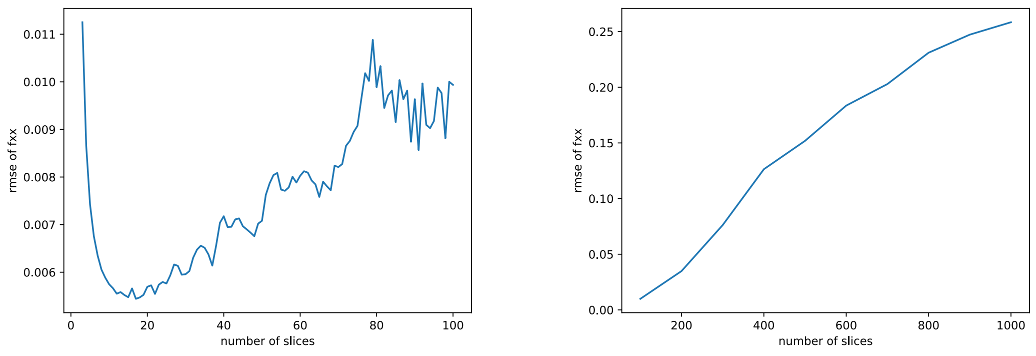


Figure 6.5: Two examples of the discretized calculation of first and second derivatives of the area distribution for a Sears-Haack body compared with the analytical solutions.

A convergence study is performed to identify the number of slices before numerical error appears in the second derivative. The RMSE of the discretisation is calculated from the analytical solution at each discrete point. Due to the singularity at the tip and tail of the geometry, the points at the boundaries are ignored when calculating the error. Slices are calculated for n within the range $[4, 100] \subset \mathbb{N}$. The results from this are shown in Fig 6.6a. Until approximately 17 slices are used, the RMSE in smoothness gradually reduces as the number of slices increases. After this point, the correlation becomes non-smooth and it is likely that a degree of numerical error is introduced to cause this. Slices are also calculated in increments of 100, starting from 100. Fig. 6.6b shows that the RMSE increases significantly across this range.



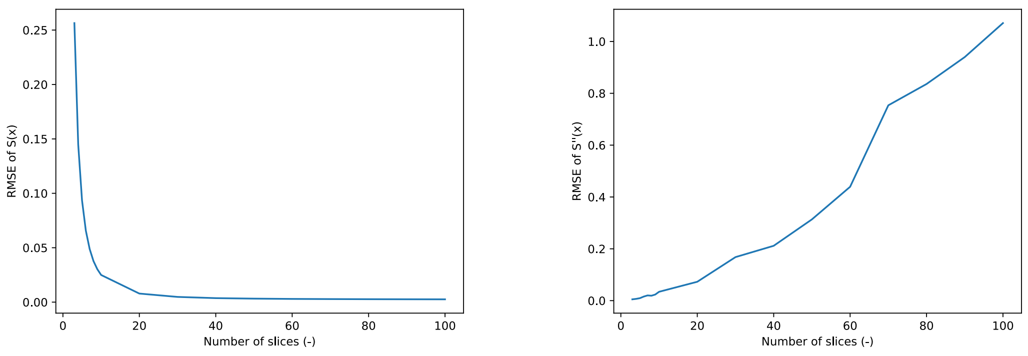
(a) 1-100 (Interval = 1) (b) 100-1000 (Interval = 100)

Figure 6.6: Convergence plots of the RMSE against number of discrete slices taken over the length of the body.

Introducing large quantities of error into the second derivative will have a detrimental impact when calculating the objective function J. This analysis highlights the importance of using an appropriate number of slices to ensure that this error is not introduced.

6.5.1.2 Sphere

The second geometry with an analytical solution discussed in Section 6.4 is the unit sphere. This geometry poses a different challenge to the Sears-Haack body.Fig 6.7 shows the RMSE of area distribution for the unit sphere and its second derivative as the number of slices increases. While the approximation of the area distribution itself increases monotonically, the second derivative approximation behaves in a completely opposite manner.

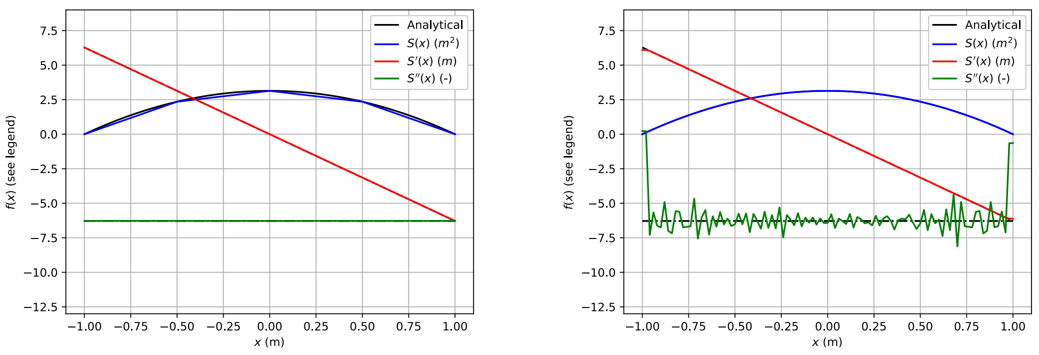


(a) $S(X)$ (b) $S''(X)$

Figure 6.7: The RMSE of the area distribution and second derivative compared to the analytical solution as the number of slices used to discretise the unit sphere for area-distribution calculation increases.

To identify why this may be the case, the analytical and discrete derivatives when using 4 and 10 slices are shown in Fig. 6.8. It is clear from these figures that, when using just four slices, the second derivative is calculated almost exactly.

The exact reason for this correlation is not clear, and further investigation here would be ideal.



(a) (b)

Figure 6.8: Comparsison between discrete area distributions (and derivatives) for a sphere using 4 and 10 slices.

6.5.1.3 Summary

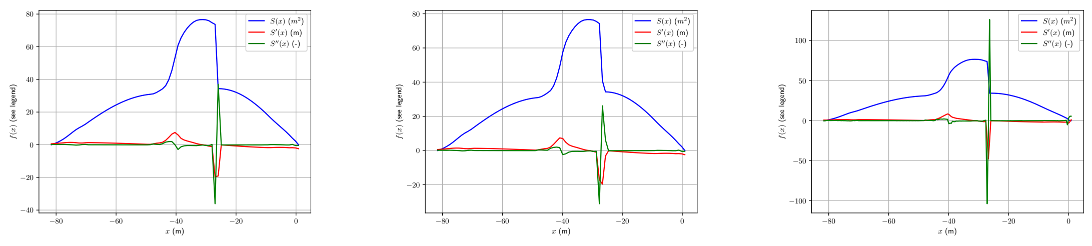
The aim of investigating the number of slices is to determine an optimal number to use within an optimisation procedure. While the study on a sphere would suggest that only 4 slices would be optimal, this is not practical for complex geometries. From both cases, it is clear that increasing the number of slices into the order of hundreds will not necessarily result in more accurate results. It is therefore suggested that approximately 100 slices is used for complex bodies.

6.5.2 Skylon Geometry

Extending the area distribution method into the supersonic regime (increasing beyond Mach 1) introduces additional corrections to the discretisation, rotating the planes relative to the oncoming freestream. In this section the sensitivity study is extended and tested on the Skylon spaceplane. The number of slices used is again considered, the effect of inclining the plane by the Mach angle is explored and a sensitivity study is performed on the number of rotations required to calculate the Mach corrected area distribution.

6.5.2.1 Number of Slices

Fig. 6.9a shows the area distribution of the baseline Skylon geometry at $M = 1.0$ with 80 slices along the length of the body. The significant difference in the cross-sectional areas between the slices before and after the end of the wings/nacelles at $x \approx -25$ results in a discontinuity in the cross-sectional area at this point. This has a huge impact on the geometry smoothness calculation, with significant spikes in $S''(x)$ around this point. As the number of slices used changes, the magnitude of the spikes in $S''(x)$ at $x \approx -25$ changes. Increasing the number of slices by just 1 results in a decrease in magnitude, while increasing the number of slices to 200 results in a steep increase in magnitude. Distributions for 81 and 200 slices are shown in Fig. 6.9. The objective function the case with 200 slices is dominated by the large spike around $x \approx -25$. These figures highlight how sensitive the calculation is. Using more slices will result



(a) 80 Slices: $J = 2780.81$ (b) 81 Slices: $J = 1746.89$ (c) 200 Slices: $J = 11245.22$

Figure 6.9: Area distribution plot for the Skylon spaceplane using varying numbers of slices along the length of the geometry, calculate at conditions $M = 1.0$, $\alpha = 0.0$.

The differences in the size of discontinuity between the different numbers of slices are due to the different locations along the length of the geometry at which the slices are taken. As the slices are uniformly distributed along the length, changing the number of slices changes the positions of the slices around the discontinuity. The slight changes in position in Fig. 6.9a and Fig. 6.9b show how significant this effect can be on the size of the discontinuity and, subsequently, on the second derivative of the area distribution. During the optimisation procedure of the Skylon spaceplane, the lengths of different geometries differ to enforce the volume constraint described in Section 1.4.2. This results in the positions of slices relative to reference points on the geometry being different between different geometries. Consider two volume-constrained geometries of different lengths, g_1 and g_2 where the CFD solution for these geometries suggests that g_1 has a lower C_d than g_2. The difference in lengths of the two geometries will result in the slice spacing being different in each design. Therefore, the area distribution of g_1 could be negatively affected by a larger change in the second derivative around the discontinuity, leading to the cost function J incorrectly determining that g_2 is a better design. Incorrect cost evaluation of designs can negatively impact the performance of optimisation algorithms, resulting in the optimisation procedure incorrectly optimising towards non-optimal designs. CFD analysis of the resulting ”optimal” design from such a study would reveal a more accurate indication of the performance of the design. Given that the lengths of each geometry are likely to be unpredictable, this is something that needs to be taken into account during the optimisation procedure when selecting the number of slices to use for a geometry.

6.5.2.2 Mach Plane

When extending the transonic area rule at Mach 1 into the supersonic regime, the planes used to slice the geometry must be inclined at an angle calculated as a function of the Mach value, Equation 6.2. The optimisation objective for the Skylon geometry is to minimise drag within the transonic regime at $M = 1.2$. The area distribution calculation should take this into account to match the optimisation parameters used in CFD driven optimisations, resulting in an angle $\mu_m = 0.985^c$.

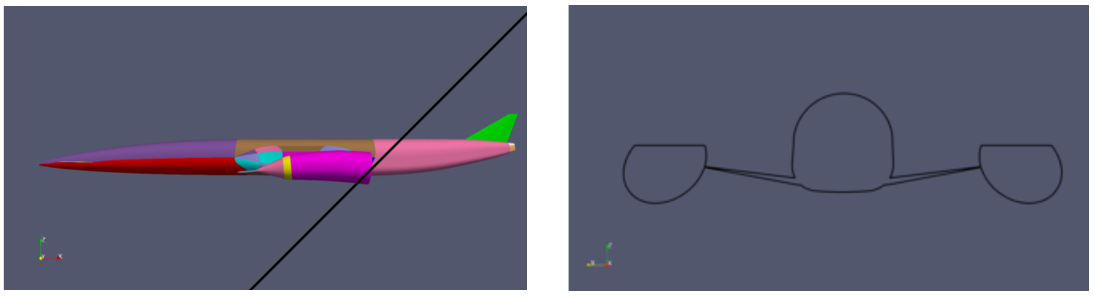
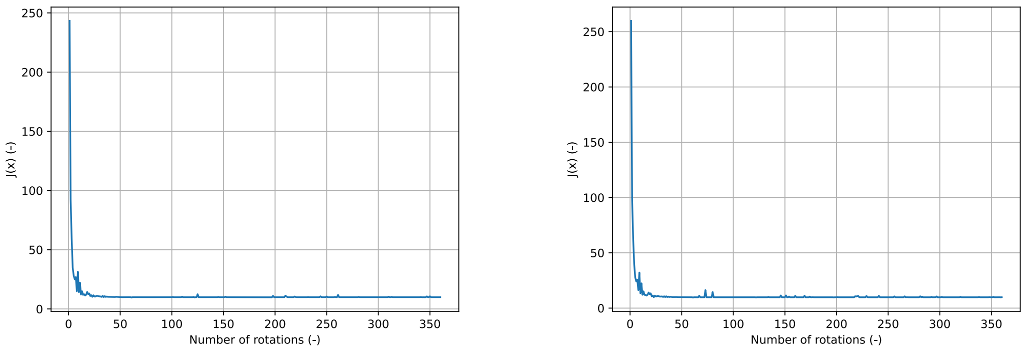

(a) Side view with plane location. (b) Cross-sectional intersection.

Figure 6.10: Mach plane cutting through the Skylon geometry. Angle of inclination is arbitrarily large to visualise the change in shape of the intersection curve.

6.5.2.3 Plane Rotation

The second step when calculating the supersonic area distribution is to rotate the inclined Mach plane about the axis normal to the oncoming flow and integrate the result. The continuous integration can be handled discretely by averaging over n rotations. A sensible choice would be 360 rotations, at a rate of 1 slice per degree rotation. To determine the suitability of this, and if this quantity is necessary, a convergence study is done to determine approximately how many rotations should be made.

Initially, 80 slices were taken across the length of the body, with the number of rotations varying from 1 to 360, for which the convergence history of the geometry smoothness is shown in Fig. 6.11a. The resulting graph exhibits numerous peaks that diverge significantly in value compared to their adjacent tests. This is an unexpected result, and the likely cause of this is similar to the reasoning exhibited in Section 6.3.1. Repeating this test with 81 slices instead results in the convergence history shown in Fig. 6.11b in which the spikes, while still present, are significantly reduced in magnitude.

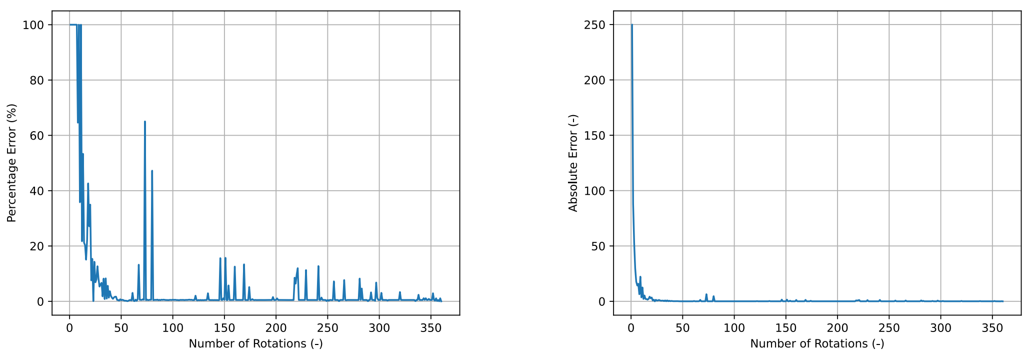


(a) 80 Slices (b) 81 Slices

Figure 6.11: Comparison between the convergence of the smoothness, $J(x)$, as the number of rotations used increases between 1 and 360 for 80 and 81 lengthwise slices respectively.

The main reason for running this convergence study was to determine whether increasing the number of rotations toward 360 resulted in the smoothness calculation converging. This is apparent from Fig. 6.11b, however, computationally, this takes 360 times longer to calculate than with just one plane. Where computational resource is limited, it is beneficial to identify if fewer rotations can be used and still achieve a similar value of smoothness. To determine this, the relative error from the previous number of rotations used and the absolute error compared to using 360 rotations is considered.

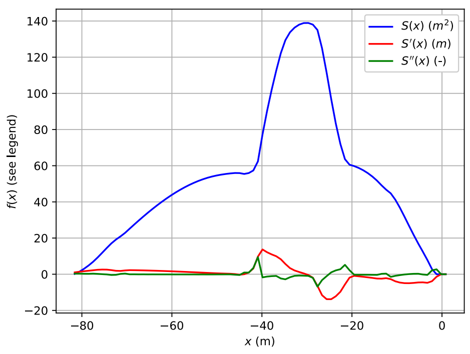
If the smoothness calculation with 360 rotational slices is taken as the "truth" value, then the percentage error of this value is shown in Fig. 6.12b. Using just 32 slices results in a difference of just 0.8%, and a relative error of using 31 slices of $2.3e-5$. This is a reasonable approximation, while taking significantly less time. Based on this, where computation time is of consequence, 32 slices can be used.



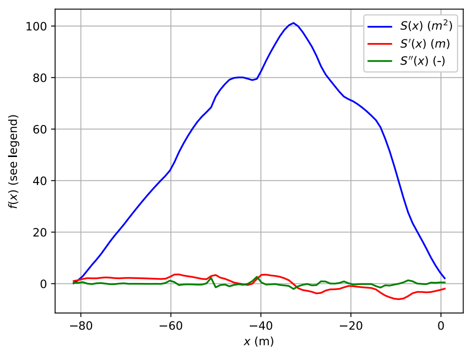
(a) Percentage Error (b) Absolute Error

Figure 6.12: Relative error from previous iteration (left) and percentage error compared with using 360 rotations (right) using 81 slices along the length of the Skylon geometry.

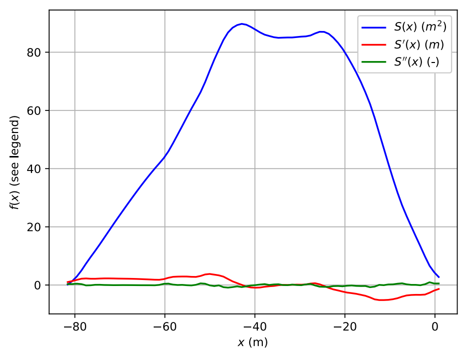
The reliability of using a coarser approximation with just 32 rotations can be qualitatively confirmed by comparing the area distribution plots. Fig 6.13 shows the area distribution plots when using 1, 4, 32 and 360 rotations. This figure highlights the differences between the plots when 4 and 360 slices are used and the similarity between the plots when using 32 and 360 rotations.



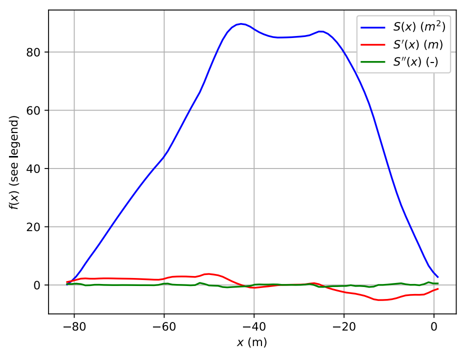
(a) 1 rotational plane



(b) 4 rotational planes



(c) 32 rotational planes



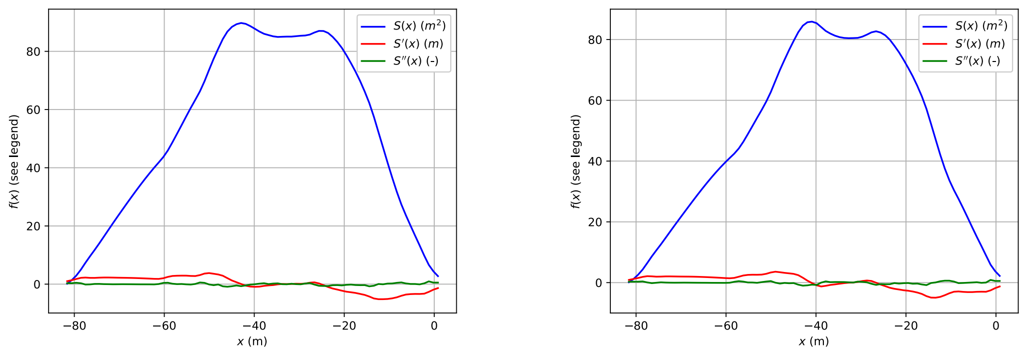
(d) 360 rotational planes

Figure 6.13: Area distribution calculations for the baseline Skylon design using varying numbers of rotational planes.

As the number of lengthwise slices used has a clear impact on the calculation of the geometries smoothness, the analysis of all geometries within the optimisation procedure should be compared using the same number of lengthwise slices. It should be noted that for some geometries, the selected value may not be the optimal value for that geometry, and a large spike in smoothness may be erroneously calculated. All of the spikes lead to an increase in smoothness value, which could lead to a well-performing design being erroneously calculated to be a poorly performing design. This false negative may lead the optimiser to avoid some good designs. On the other hand, consider the scenario where the optimiser drives towards false positives. A false positive in this instance would be a poor design is identified as a good design. This would be a significantly worse outcome.

6.5.2.4 Angle of Attack

So far all designs have been calculated at an assumed incidence of 0. All CFD driven optimisation studies were subjected to a lift constraint at an incidence of 4^o. Therefore, this should be taken into account. The handling of this is included in Jones' report, that the normal of the interception plane should be parallel to the incoming stream [56].



(a) $\alpha = 0^o$ (b) $\alpha = 4^o$

Figure 6.14: The area distribution curves and derivatives for the baseline Skylon geometry at varying angles of attack, α. While the distributions are visually very similar, a clear difference in the shape of the peak between $-40 \geq x \geq -20$ can be seen. There is also a significant difference in the smoothness; $J = 14.70$ at $\alpha = 0$ and $J = 19.71$ $\alpha = 4$, a 34% difference

The impact of including the additional plane inclination in the area distribution curve calculation can be seen by comparing the distributions for the baseline design at 0 and 4 degrees of incidence, Fig. 6.14. It is clear that even a small angle can have a large impact on the calculation.

6.6 Optimisation Study - Sphere

To demonstrate and test the implementation of the smoothness driven design optimisation, the unit sphere from Section 3.2.2 is used, along with the parameterisations described.

6.6.1 Objective

The objective of the optimisation is drag minimisation at Mach 1 and is volume constrained. Using the area distribution evaluation function described within this chapter, the expected outcome of the study is that the geometry should tend towards that of a Sears-Haack distribution as closely as the defined parameterisation allows.

6.6.2 Simple Parameterisation

The first optimisation case uses the simple parameterisation from Section 3.2.2. Scaling of the axial radius of the geometry around the x-axis is used as the independent parameter. The volume is then constrained by scaling the length of the geometry along the x-axis as the dependent parameter. An example of this is given in Fig. 3.12.

The bounds for the radial scaling are $\phi \in [0.3, 1.5]$ and the PSO algorithm was used with 5 particles for 4 generations. The convergence history for is given in Fig. 6.15. Within this short history, the algorithm converged to the optimal solution within this domain at $\phi = 0.3$. Comparing geometries, Fig. 6.16b and Fig. 6.16c show that the optimal design tends toward the slender Sears-Haack shape as expected as close as could be made by this parameterisation.

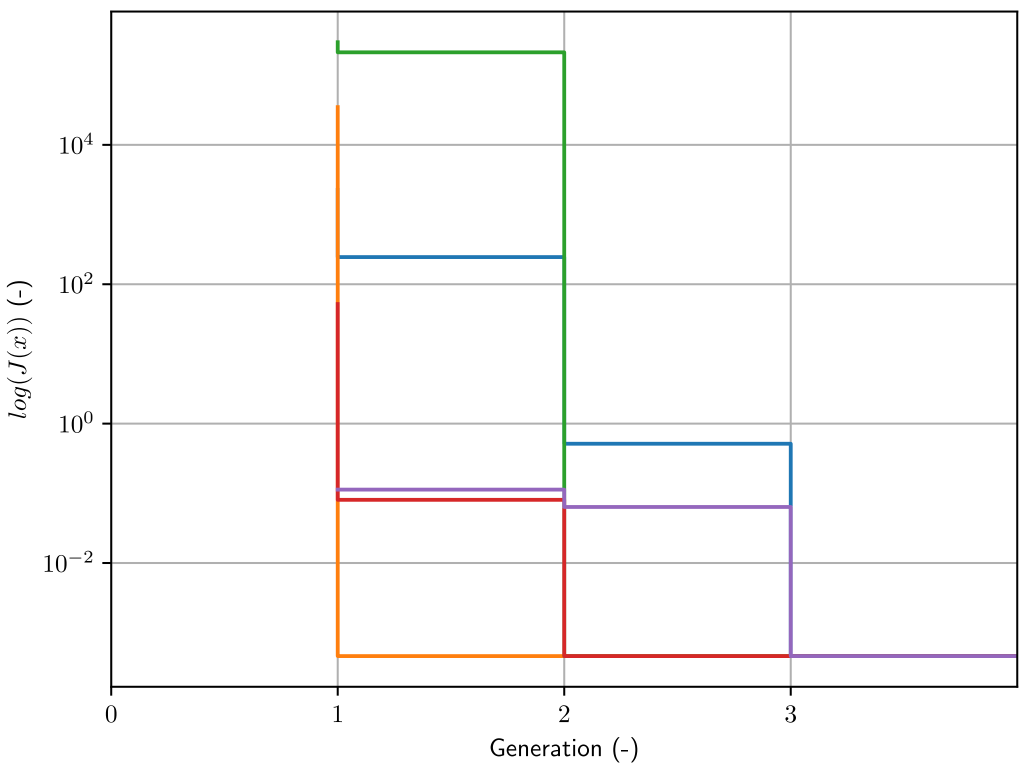
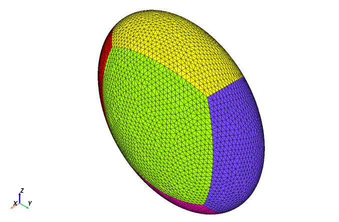
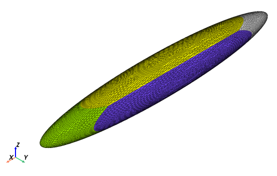


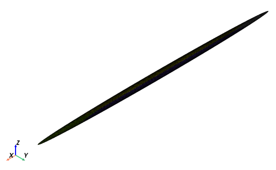
Figure 6.15: Convergence history of the volume constrained sphere optimisation case with one independent parameter (axial raidus) and one dependent parameter (body length). y axis is cropped due to large values of second derivative in short geometries.



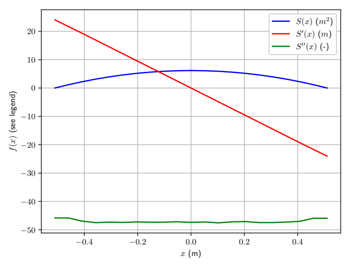
(a)



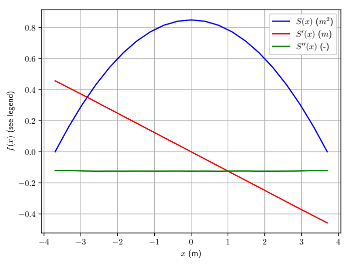
(b)



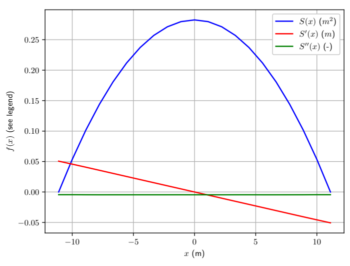
(c)



(d)



(e)



(f)

Figure 6.16: Geometries (top) and area distributions (bottom) of three different designs from the simple sphere optimisation case. Considering the limited domain, a poor design (left), good design (middle) and optimal design (right) are shown.

6.7 Optimisation Study: Skylon

One of the key aims of the parameterisation of Skylon described in Chapter 4 was to drive the wave drag of the design down by making the area distribution of the geometry smoother. Considering this goal, this makes the parameterised Skylon geometry a suitable optimisation problem to use the approach outlined in Section 6.2. The same parameterisation approach used in previous test cases was used, replacing the expensive evaluation function with this significantly cheaper alternative approach.

6.7.1 Justification

The aim of the proposed method is to find an approximate value for the optimal solution using a significantly less expensive cost function. Before exploring this in depth, a study of the existing datasets should be made to determine if there is, in fact, correlation between the calculated area distribution and the drag coefficient calculations.

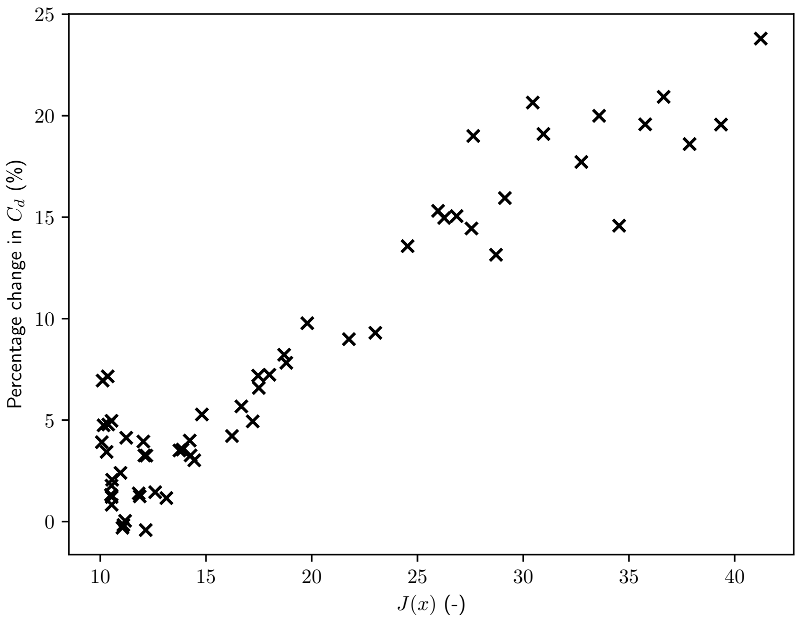


Figure 6.17: Correlation between calculating the area distribution cost function for a geometry and the percentage change in drag coefficient from the CFD data for that same geometry in a sample set of 60. The standout outlier is highlighted with a red circle.

The same training data set from Section 4.2.1 was taken, and the area distribution was calculated for each geometry at $M = 1.2$ and $\alpha = 4.0$. The average of 32 rotated slices was taken at each station for x for 80 slices along the length of the body. The resulting smoothness, Equation 6.14, for each geometry was compared against the percentage change from the baseline design in lift constrained drag coefficient from the CFD solutions. This correlation is shown in Fig. 6.17 where there is a clear correlation between the two analysis methods. The Spearman rank coefficient between the two data sets is 0.909, which represents is a strong correlation. This suggests that using the area distribution to approximate the cost function for a geometry will result in a pattern of results similar to that when using full fidelity CFD.

6.7.2 Results

An optimisation study is run at no angle of attack for the Skylon parameterisation from Section 1.4.2 using the PSO optimiser with the objective function from Equation 6.14. The area distribution is calculated by averaging the area from 100 planes rotated about the x-axis and 80 slices along the x-axis. Fig. 6.18 shows the convergence history from this study and the optimal design resulting is shown in Fig. 6.19

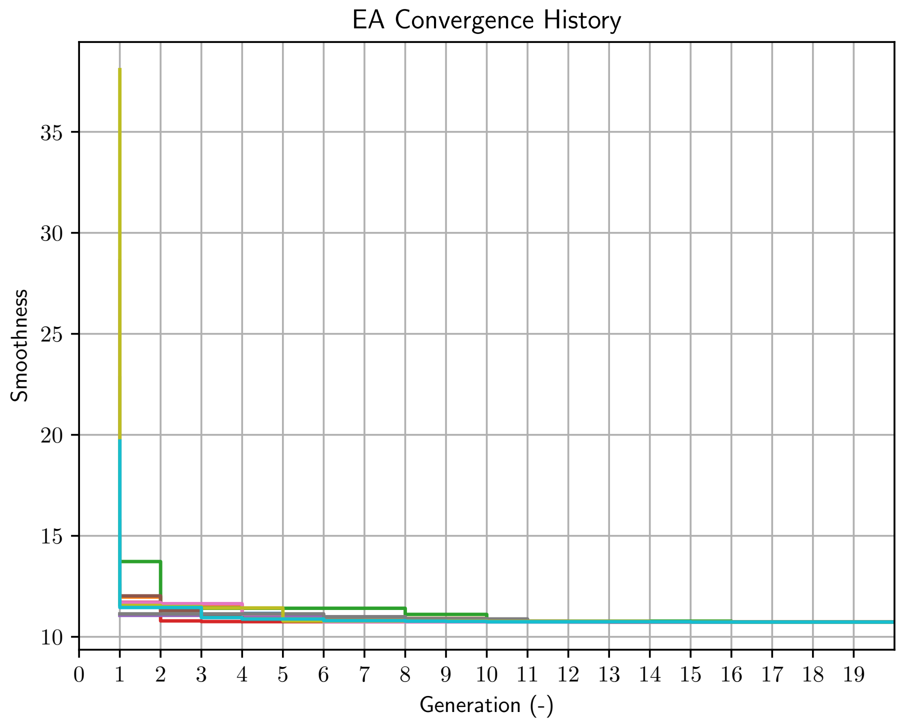
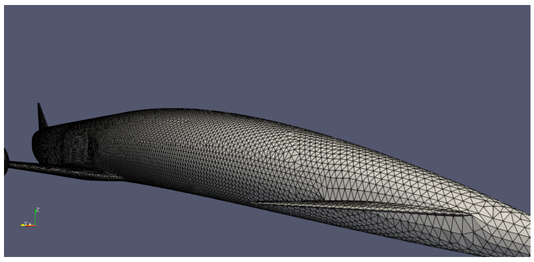


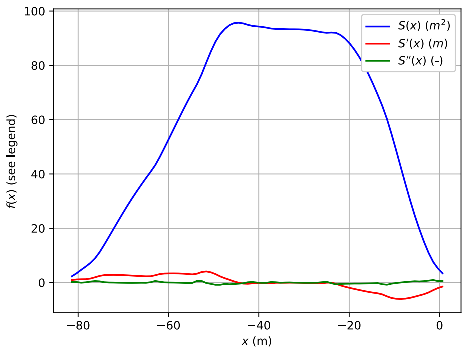
Figure 6.18: Convergence history for the Skylon optimisation using PSO using 10 particles over 20 generations. The evaluation function used the area distribution calculation including the averaged rotation of the inclined Mach plane and the incoming flow.

The optimal design is shown in Fig 6.19a with Table 6.1 containing the parameter values for this design. It is evident that the optimisation has driven the pinched section back out and drooped the nose down. Aerodynamically, compared with the results from the CFD driven optimisation, the central panel drawing back out is not expected to improve the geometry. From the model used to fit the DOE data set, it is expected that this design would not perform as well as the baseline design.

Parameter	Value
ϕ_1- Nose droop	-3.453
ϕ_2- Fuselage pinch	0.191
ϕ_3- Central panel stretch (x)	1.2
ϕ_4- Central panel stretch (z)	1.071
ϕ_5- Central section stretch	0.9

Table 6.1: Parameter values for the design with the lowest object function value following the smoothness driven EA optimisation of the Skylon spaceplane.


(a)



(b)

Figure 6.19: Front view of the optimal mesh from smoothness driven PSO and its area distribution plot.

The reason for this is clear from comparing the baseline area distribution, Fig 6.14b, with the optimal design from this study Fig 6.19b where it is clear that the pinched section results in a dip in the area distribution. Adding volume back into this region flattens the top of the area distribution curve, reducing the second derivative in this region, driving the optimisation to favour designs with this feature.

6.8 Conclusion

Despite the lack of performance gain from optimisation when using geometry smoothness as the evaluation function, this study has shown that the method is capable of identifying designs that change in ways we would expect from known theoretical knowledge.

It is also important to note that the optimisations were not direct comparisons. The CFD driven optimisation sought to reduce the overall drag while the area distribution driven optimisation would theoretically only optimise to reduce the wave drag. Aerodynamic system are highly complex and drag contribution can come from a number of additional sources.

Chapter 7

Conclusions

This thesis is concluded with a brief summary of the contributions presented in this work, along with some suggestions of future avenues of work and a closing statement. A list of conference presentations and journal submissions is also included.

7.1 Contributions

Two main contributions are presented in this thesis, a novel mesh morphing framework, and the design exploration of the Skylon spaceplane.

This section summarises these two contributions and references how they contribute to the key aims and objectives outlined in Section 1.5. For clarity, these 4 objectives are provided again here:

1. Provide indication of baseline performance for current Skylon spaceplane.
2. Develop a novel design optimisation methodology.
3. Investigate and evaluate different configurations of the proposed method.
4. Demonstrate methodology to a proof-of-concept test case provided by the industry sponsor.

7.1.1 Mesh Morphing Framework

The key output from this thesis was the mesh morphing framework discussed and implemented in Chapter 3. Radial Basis Functions are widely used for mesh morphing applications but do not scale well as the node count increases. This method sought to reduce the number of source nodes used within the $O(n^3)$ matrix inversion, therefore reducing the time spent computing matrix inversion when morphing. This was achieved by focusing on using the information within the boundaries between sections to train the RBF network and using this to interpolate and propagate the morph across sections of the mesh to ensure the mesh was still suitable for computational analysis following the morph. The framework also included the option to constrain certain properties of the geometry, such as volume, surface area etc. A user defined dependent parameter was used with a trial and error approach to constrain the property to the required value. Development of this methodology achieved the second objective introduced above.

The mesh morphing aspect of the framework can be combined with different optimisation methods and objective functions. This was demonstrated by performing different optimisations on the Skylon geometry using Design of Experiments, Bayesian Optimisation and Particle Swarm Optimisation, and different objective functions driven using both CFD and a custom objective function. These tests demonstrated the application of different configurations of the methodology, satisfying Objective 3.

7.1.2 Optimised Performance

The main target from an industry perspective was to demonstrate the capability of the work undertaken in this thesis. This was achieved by performing two optimisation studies using design of experiments and Bayesian optimisation. Both studies used the same mesh morphing parameterisation and resulted in locating improved designs with a reduction in drag coefficient across the transonic regime. The design of experiments study reduced the drag coefficient by approximately 1.6% while the Bayesian optimiser reduced by approximately 2%. Not only did using Bayesian optimisation find a better design, but it did so using 1/3 of the computational resource that the design of experiments study did. As part of these tests, the baseline performance of the Skylon geometry was presented. This satisfies the first objective for this thesis, and the collection of tests satisfies the fourth objective.

The framework was also demonstrated to work with a different evaluation method within the objective function and with a third optimisation method, Particle Swarm Optimisation. The aim of the area distribution-driven optimisation was to identify an optimised design that yielded similar design parameter values to those obtained from the more computationally expensive optimisation method. While this experiment ultimately did not lead to an optimised design being found, it's expected performance was justified by evaluating the method across a range of designs to assess its potential effectiveness. The resulting geometry from this optimisation study did not result in a design with a lower drag coefficient. The objective function only considered the wave drag into consideration, in the transonic regime, drag arises from several sources, including pressure drag, friction drag, and induced drag. This likely explains why the method did not yield the expected improvement in design, as it did not account for these other forms of drag in the optimization process. Despite the lack of success in this test, it helped to satisfy the third objective for this thesis.

7.2 Future Work

While the work presented in this thesis has successfully addressed the primary research objectives and provided valuable insights into the optimization process, there is always room for further refinement and exploration. This section outlines potential directions for continuing this research.

7.2.1 Applications

The flexibility of the framework introduced in this thesis is one of the most interesting features that stood out to the industry sponsors of this research. It is hoped that this will enable many future design optimisations, not just within the scope of the transonic performance of spaceplanes, but in many other fields. To support this, one of the major areas of improvement that this work would benefit from, is that a wider range of examples could be explored. While multiple examples of the principle use case are explored in this research, it would be more rigorous to demonstrate a wider range of applications. Expanding to include more examples would help validate the robustness and versatility of the approach, providing further confidence in its ability to address a diverse range of real-world scenarios. Additionally, exploring edge cases or more complex examples could reveal further limitations or opportunities for improvement in the method.

7.2.2 Refined Mesh

One of the main drawbacks with the results presented in this research is that the mesh resolution used in the CFD study was insufficient to capture the necessary detail, which limited the reliability and significance of the results. If the results from the Skylon geometry are required to contribute to a wider study, then a beneficial next step in this research would be to repeat the tests with a better resolved mesh.

One of the main reasons for the introduction of the novel mesh morphing method was to reduce the overall node count within the RBF matrix inversion. This could have been examined in more detail throughout this thesis and it could be beneficial to explore this in more detail. A study to explore how much longer the morphing takes when increasing the node count of the meshes would be useful information before applying the methodology to real world industry use cases.

7.2.3 Implementation

A major limitation of the current implementation of the mesh morphing framework is that it is implemented in the programming language Python. The decision to use this language was made because of its ease of use and speed at development. It allowed for rapid prototyping and testing of different implementations. It is believed that this decision was appropriate, however the language is not designed to be used for software where runtime speed is of importance. As a result, it is believed that re-writing the main software algorithms in a compiled language would offer significant speed-ups in operational runtime of many algorithms outlined within this thesis. Using the methodology in a production environment would benefit from decreased runtimes. In addition to this, many techniques exist to speed up the calculation time of matrix inversion. This is the most computationally expensive part of the process and as part of a re-writing/refactoring of the code, these techniques could be applied for additional speed up.

7.2.4 Extensions

The implementation could be expanded in a number of ways to further explore how different parameters affect optimisation problems. For example, when using the RBF during the morphing process, different shape parameter selection methods and basis functions could be explored during the different steps of the morphing process. In addition to this, the examples presented within this work, only the volume of the geometry is constrained. Different geometric properties could be implemented and explored such as surface area, or constraining properties for subsections of the mesh (e.g. just the fuselage) rather than the whole body.

In addition, each parameter is controlled by a single scalar value. As a result, only simple functions that were defined by one parameter could be implemented such as linear transformations. If the parameters could be passed as vectors, then more complex functions, such as quadratics, could be used to define the transformation function.

7.3 Publications

Throughout the course of the research performed during this thesis, several aspects of the work were shared with the academic and professional community. The following presentations and publications summarise the contributions shared during the study. These works serve as a testament to the relevance and impact of the research in the field.

7.3.1 Conferences

- Ben Smith, Ben James Evans, and Sean Peter Walton. Development of transonic and supersonic cfd modelling and design optimisation techniques for spaceplanes. In *UKACM*, Nottingham, UK, 2022
- Ben Smith, Ben James Evans, and Sean Peter Walton. Optimisation methods for computationally expensive design processes. In *UKACM*, Warwick, UK, 2023
- Ben Smith, Ben James Evans, and Sean Peter Walton. Optimisation methods for computationally expensive design processes. In *15th International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control*, Chania, Crete, Greece, 2023
- Ben Smith, Ben James Evans, and Sean Peter Walton. Optimisation methods for computationally expensive design processes. In *22nd Computational Fluids Conference.*, Cannes, France, 2023

7.3.2 Papers

- Ben James Evans, Ben Smith, Sean Peter Walton, Neil Taylor, Martin Dodds, and Vladeta Zmijanovic. A mesh-based approach for computational fluid dynamics-free aerodynamic optimisation of complex geometries using area ruling. *Aerospace*, 11(4):298, 2024
- Ben Smith, Ben James Evans, Sean Peter Walton, Neil Taylor, Martin Dodds, and Vladeta Zmijanovic. A novel mesh morphing methodology for computational aerodynamic shape optimisation. *In Preparation*, 2024

7.4 Closing Statement

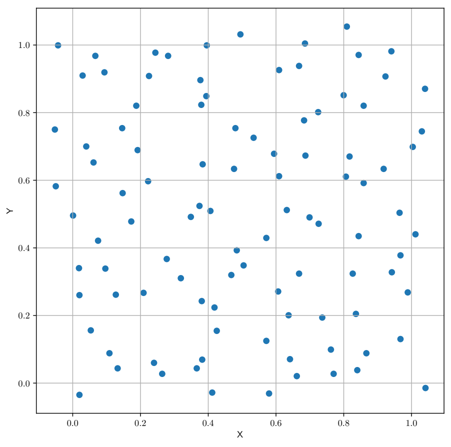
This thesis presented a novel mesh morphing framework that offers a flexible and efficient method for easy parameterisation of geometry. The framework was used to optimise an industry case study, demonstrating its compatibility with different optimisation algorithms and evaluation methods in the objective function. By streamlining the design optimization process for computationally expensive engineering problems, the methodology has the potential to significantly reduce computational costs while improving design outcomes. The field of design optimisation, particularly in complex, high-fidelity computationally expensive engineering problems, is an active and exciting field. The proposed framework serves as a valuable tool that can be extended to a range of applications. Future developments could further enhance its capabilities, allowing for more efficient optimizations and transforming how engineers optimise complex design problems. Ultimately, this work contributes to advancing the state of the art in design optimization and highlights the importance of innovative methodologies in tackling the increasingly complex problems faced by the industry.

Appendices

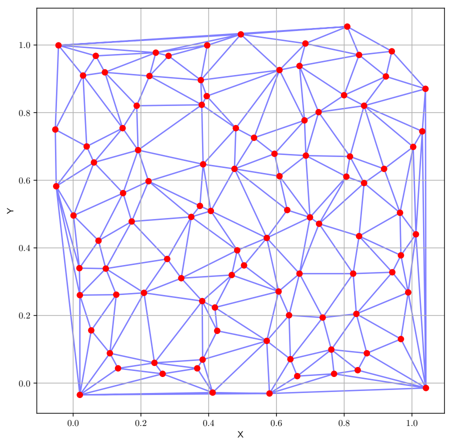
Appendix A Datasets

x	y
-0.04327152034139628	0.9990379689554798
-0.05266741846118675	0.7500528599908571
-0.04998290540056352	0.5822585233011337
0.019815560659341314	-0.03479152149514814
0.13256674772726407	0.04369290632950851
0.10840582295882728	0.08834773171274385
0.053372485955177684	0.1560068333561484
0.019815560659341314	0.26020170122219966
0.01847320172142075	0.3400392883670479
0.0010236620121513162	0.49565495888631006
0.07484889766299108	0.42123006064194746
0.09632530937080469	0.33868612904685635
0.12719772160601767	0.2615548605423913
0.20907659884633656	0.2669676010625962
0.23994890867394042	0.05993107627040428
0.2641099358499861	0.027454633149174457
0.41176040715166584	-0.028025621654751504
0.5795450336308476	-0.030731991914853957
0.3661229682678069	0.04369290632950851
0.38223035385437354	0.06940329475118417
0.42518317727000055	0.15465367403595665
0.5714913920413686	0.12488370441426766
0.6412897556936646	0.07075650569109493
0.7701483283481545	0.027454633149174457
0.839946794408059	0.0382801658093036
1.0412883461830769	-0.014493821973958404
0.9674631105322371	0.13029639331475357
0.8667923346447284	0.08834773171274385
0.7620946867586753	0.099173264372873
0.6372629861027296	0.20066176197882185
0.7365914030523181	0.1938958621384254
0.8359199224095151	0.20472129155911606
0.9889396246476592	0.26832076038278785
0.9419599292334884	0.3278606996261657
0.9674631105322371	0.37792834295918476
1.011758292885785	0.44017462665280527
0.9647785974716141	0.5037740954764771
0.8439736664066031	0.4347618603227408
0.8265240242897245	0.3238011700458715
0.6681352959303336	0.3238011700458715
0.6063905738675166	0.2710271306428903
0.4184717922108331	0.22366583176011406
0.38088799491645287	0.2426103719611124
0.31914327285363586	0.3102693703650783
0.2775327059683206	0.3671028619187747
0.17283505808226768	0.47806365543508234
0.14733187678351933	0.5619608237799439
0.06142612754465665	0.6526238919652022
0.0399497158368432	0.6999851908479783
0.06679525607351217	0.9679147883938799
0.0936407963101813	0.919200356000772
0.19162695672945806	0.6891596581878491
0.22249936896467104	0.5971435339218378
0.18760018713852278	0.8204182286549253
0.2251838820252942	0.9083749265800811
0.2439757806724842	0.9773871101140985
0.2815595779668646	0.9679147883938799
0.395653123972708	0.9990379689554798
0.4949815409222965	1.0315143088372711
0.376861225325518	0.8961962862194797
0.39431076503478735	0.8488349873367034
0.37954573838614136	0.823124598915028
0.3835726103846854	0.6472112030647165
0.37417660985728585	0.5240717949976669
0.3486734285585371	0.49159545511587566
0.4063912786228103	0.509186835996682
0.46813600068562733	0.3197416404655774
0.504377541449696	0.34815839914735547
0.4842433862721943	0.3928133277700291
0.5714913920413686	0.4293491972321144
0.6318938575738738	0.5118931804469248
0.6990077081655466	0.4902423216055435
0.7258532484022155	0.47129778140454526
0.47618974468271535	0.6336794033839231
0.4802165142736503	0.754112389571151
0.5339075947469883	0.7256956308893732
0.5943100602794935	0.6783343320065969
0.60907508692814	0.6120284929228226
0.6869271945775238	0.672921591486392
0.8063898691122227	0.6106753336026309
0.8587385906476404	0.5917307934016328
0.9177989020574429	0.6336794033839231
1.0037045488886969	0.6986320315277865
1.0305500891253658	0.7446401194706521
0.8171281261699341	0.6702152212262895
0.6829003225789798	0.7771164593524433
0.7245109918719038	0.8014736884539271
0.60907508692814	0.9259662558411683
0.6681352959303336	0.9381448962017702
0.8090743821728461	1.0545183528087039
0.7996784840530557	0.8515413575968058
0.8587385906476404	0.8204182286549253
0.9231679281786893	0.9070217156401701
0.8439736664066031	0.9706212102737017
0.9406175702955681	0.9814465880746732
1.0399459872451562	0.870485897797804
0.6855849380472121	1.0044507038297779
0.029211458779131783	0.9097280859002728
0.14598951784559877	0.754112389571151
0.6614239108711666	0.020688733308777796

Table 1: Data points for the randomly generated 100 point data set used for dmonstrating RBF interpolation in Chapter 3. Data set created using plot digitization from the figures presented in [31].



(a) Scatter



(b) Mesh

Figure 1: Scatter and mesh plot of the 100 point data set in Table 1 used for demonstrating RBF interpolation.

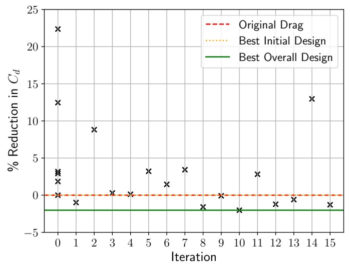
Appendix B DOE $N = 60$ Sample

Design	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5
1	-3.250	-1.938	0.803	1.022	0.988
2	1.917	-1.521	1.157	0.972	1.067
3	-3.083	-1.104	0.823	0.935	0.948
4	-1.417	-1.312	0.857	0.958	1.047
5	-4.917	-0.938	0.997	1.082	1.127
6	0.083	-0.479	0.863	0.968	1.172
7	-1.583	0.021	0.910	0.918	0.932
8	2.583	-1.604	1.017	1.045	1.182
9	-4.083	-0.229	1.170	0.988	0.983
10	-4.417	0.062	1.090	0.938	1.052
11	-4.250	-0.188	0.897	0.995	1.163
12	4.083	-0.063	1.177	0.942	0.943
13	-2.917	-1.062	1.057	1.065	1.157
14	2.917	-0.729	1.010	1.038	1.117
15	-0.917	-0.771	0.890	1.072	1.018
16	-3.917	-1.396	0.830	0.985	0.958
17	3.750	0.229	1.130	1.075	0.997
18	-2.750	0.312	0.950	1.055	1.083
19	0.750	-1.188	1.030	1.058	0.922
20	-0.083	-1.562	1.150	0.925	0.912
21	-1.917	-1.646	1.190	1.012	1.132
22	-2.250	-1.229	1.110	0.965	0.917
23	-3.583	0.479	1.123	1.042	1.038
24	-3.750	-0.396	1.077	0.928	1.198
25	1.250	-1.146	0.990	0.945	1.123
26	-0.417	-1.021	1.050	1.088	1.113
27	0.583	-1.354	1.083	1.078	1.008
28	2.417	-0.854	0.870	0.922	1.192
29	-3.417	-0.562	1.037	0.908	1.022
30	2.083	-0.146	0.883	0.982	1.143
31	0.250	-0.688	0.977	0.948	1.107
32	-1.250	-0.812	1.143	0.975	0.978
33	0.917	0.271	0.917	1.032	1.087
34	-4.750	-0.521	0.970	0.915	1.012
35	-2.583	0.354	0.957	1.098	0.938
36	-0.750	-0.896	0.810	1.062	1.002
37	3.250	-0.979	0.817	0.978	1.028
38	-4.583	-0.438	1.183	1.008	1.188
39	2.250	0.104	0.837	1.052	1.077
40	3.417	-1.271	0.930	1.035	1.137
41	3.917	-1.729	0.843	1.085	1.152
42	1.750	-0.312	0.903	1.025	0.953
43	3.583	-1.438	1.043	1.048	0.902
44	-1.083	-1.688	1.063	1.092	0.973
45	-2.417	-1.812	1.097	1.028	1.103
46	4.583	-0.104	0.877	0.998	1.073
47	0.417	-0.021	1.197	0.912	1.177
48	-2.083	0.438	1.103	0.905	0.963
49	2.750	-1.979	1.137	1.068	1.167
50	1.417	-1.896	1.003	0.952	1.042
51	1.083	0.188	0.937	0.955	1.032
52	3.083	-0.646	0.850	1.015	1.057
53	-0.250	-0.271	1.070	0.932	1.062
54	4.750	-1.479	0.983	1.018	0.993
55	4.250	0.396	1.163	0.992	0.907
56	4.417	-1.854	0.943	1.095	0.927
57	-0.583	-0.604	0.923	0.902	1.147
58	1.583	-1.771	1.117	0.962	0.968
59	4.917	-0.354	0.963	1.005	1.097
60	-1.750	0.146	1.023	1.002	1.093

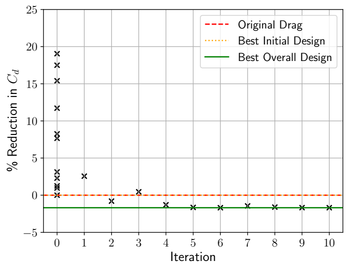
Table 2: Data set for $N = 60$ designs for the Skylon design of experiments study.

Appendix C BO Search History

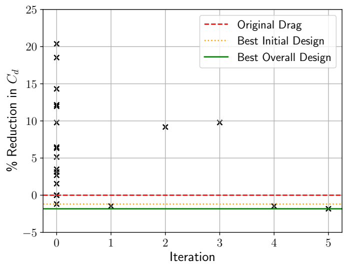
This appendix includes the search history figures for each repeat of the experiments discussed in Chapter 5. Drag search history of Skylon using the custom implementation of Bayesian optimisation. Iteration 0 contains all the initial training data, and each subsequent iteration represents the next suggested sampling point from the Bayesian optimiser. The horizontal orange dashed bar marks the C_d of the best design from the initial training data (baseline design included), and the green solid bar marks C_d of the best overall design.



(a) N=5

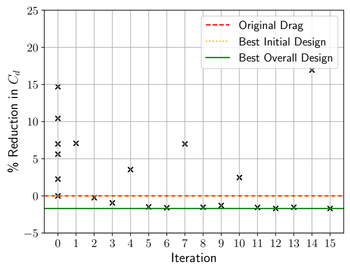


(b) N=10

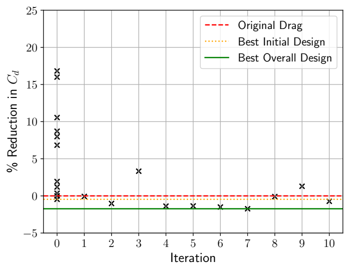


(c) N=15

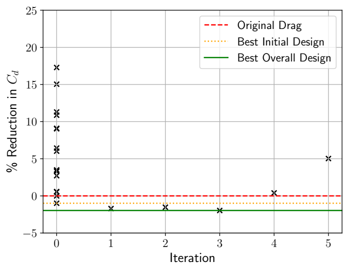
Figure 2: Repeat 1



(a) N=5

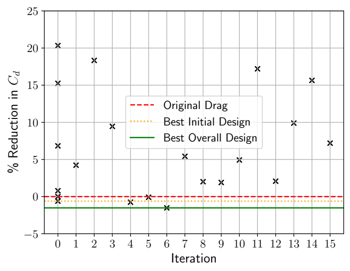


(b) N=10

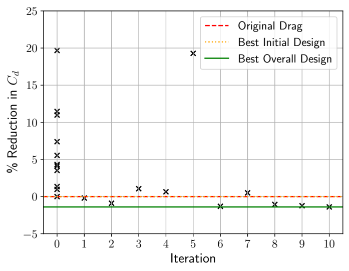


(c) N=15

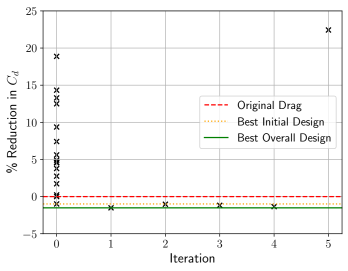
Figure 3: Repeat 2



(a) N=5



(b) N=10



(c) N=15

Figure 4: Repeat 3

Bibliography

[1] pydoe. `https://https://pypi.org/project/pyDOE/`. Accessed: 2023-08-30.

[2] Milton Abramowitz, Irene A Stegun, and Robert H Romer. Handbook of mathematical functions with formulas, graphs, and mathematical tables, 1988.

[3] Jarmo T Alander. *An indexed bibliography of genetic algorithms: Years 1957-1993.* Citeseer, 1994.

[4] Joe Alexandersen and Casper Schousboe Andreasen. A review of topology optimisation for fluid-based problems. *Fluids*, 5(1):29, 2020.

[5] Christian B Allen and Thomas CS Rendall. Cfd-based optimization of hovering rotors using radial basis functions for shape parameterization and mesh deformation. *Optimization and Engineering*, 14:97–118, 2013.

[6] J.D. Anderson. *Fundamentals of Aerodynamics.* McGraw-Hill series in aeronautical and aerospace engineering. McGraw Hill LLC, 6 edition, 2016.

[7] Holt Ashley, Mårten Landahl, and Marten T Landahl. *Aerodynamics of wings and bodies.* Courier Corporation, 1965.

[8] Utkarsh Ayachit. *The paraview guide: a parallel visualization application.* Kitware, Inc., 2015.

[9] Thomas Back, Ulrich Hammel, and H-P Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[10] Thomas Bartz-Beielstein, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. Evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(3):178–195, 2014.

[11] Michael Batdorf, Lori Freitag, Carl Ollivier-Gooch, Michael Batdorf, Lori Freitag, and Carl Ollivier-Gooch. Computational study of the effect of unstructured mesh quality on solution efficiency. In *13th Computational Fluid Dynamics Conference*, page 1888, 1997.

[12] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications.* Springer Science & Business Media, 2003.

[13] Marco Evangelos Biancolini. *Fast radial basis functions for engineering applications.* Springer, 2017.

[14] Mohammad Reza Bonyadi and Zbigniew Michalewicz. Particle swarm optimization for single objective continuous space problems: a review. *Evolutionary computation*, 25(1):1–54, 2017.

[15] Thomas Borrvall and Joakim Petersson. Topology optimization of fluids in stokes flow. *International journal for numerical methods in fluids*, 41(1):77–107, 2003.

[16] John P Boyd and Kenneth W Gildersleeve. Numerical experiments on the condition number of the interpolation matrices for radial basis functions. *Applied Numerical Mathematics*, 61(4):443–459, 2011.

[17] Martin D Buhmann. *Radial basis functions: theory and implementations*, volume 12. Cambridge university press, 2003.

[18] Jonathan C Carr, Richard K Beatson, Bruce C McCallum, W Richard Fright, Tim J McLennan, and Tim J Mitchell. Smooth surface reconstruction from noisy range data. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 119–ff, 2003.

[19] Tinkle Chugh, Alma Rahat, Vanessa Volz, and Martin Zaefferer. Towards better integration of surrogate models and optimizers. In *High-Performance Simulation-Based Optimization*, pages 137–163. Springer, 2020.

[20] Carlos A Coello Coello. *Evolutionary algorithms for solving multi-objective problems.* Springer, 2007.

[21] Dawson-Haggerty et al. *trimesh*.

[22] George De Ath, Richard M Everson, Alma AM Rahat, and Jonathan E Fieldsend. Greed is good: Exploration and exploitation trade-offs in bayesian optimisation. *ACM Transactions on Evolutionary Learning and Optimization*, 1(1):1–22, 2021.

[23] Aukje De Boer, Martijn S Van der Schoot, and Hester Bijl. Mesh deformation based on radial basis function interpolation. *Computers & structures*, 85(11-14):784–795, 2007.

[24] Jean Duchon. Splines minimizing rotation-invariant semi-norms in sobolev spaces. In *Constructive Theory of Functions of Several Variables: Proceedings of a Conference Held at Oberwolfach April 25–May 1, 1976*, pages 85–100. Springer, 1977.

[25] David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, pages 1166–1174. PMLR, 2013.

[26] Thino Eggers, Robert Dittrich, and Richard Varvill. Numerical analysis of the skylon spaceplane in hypersonic flow. In *17th AIAA International Space Planes and Hypersonic Systems and Technologies Conference*, page 2298, 2011.

[27] Ben James Evans, Ben Smith, Sean Peter Walton, Neil Taylor, Martin Dodds, and Vladeta Zmijanovic. A mesh-based approach for computational fluid dynamics-free aerodynamic optimisation of complex geometries using area ruling. *Aerospace*, 11(4):298, 2024.

[28] Gregory E Fasshauer and Jack G Zhang. On choosing "optimal" shape parameters for rbf approximation. *Numerical Algorithms*, 45:345–368, 2007.

[29] Joel H Ferziger and Milovan PeriC. Computational methods for fluid dynamics, 2002.

[30] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[31] Richard Franke. A critical comparison of some methods for interpolation of scattered data. Technical report, Monterey, California: Naval Postgraduate School., 1979.

[32] Richard Franke. Scattered data interpolation: tests of some methods. *Mathematics of computation*, 38(157):181–200, 1982.

[33] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[34] GE Gadd. A new turbulent friction formulation based on a re-appraisal of hughes results. *National Physical Laboratory, NPL, Report SH R113/68, The Royal Institution of Naval Architects, RINA Transactions 1967-25*, 1968.

[35] Tilmann Gneiting. Radial positive definite functions generated by euclid's hat. *Journal of Multivariate Analysis*, 69(1):88–119, 1999.

[36] Izrail Solomonovich Gradshteyn and Iosif Moiseevich Ryzhik. *Table of integrals, series, and products*. Academic press, 2014.

[37] W Haack. Geschossformen kleinsten wellenwiderstandes. *Bericht der Lilienthal-Gesellschaft*, 136(1):14–28, 1941.

[38] J Hall, TCS Rendall, CB Allen, and DJ Poole. A volumetric geometry and topology parameterisation for fluids-based optimisation. *Computers & Fluids*, 148:137–156, 2017.

[39] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, February 2019.

[40] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[41] Rolland L Hardy. Multiquadric equations of topography and other irregular surfaces. *Journal of geophysical research*, 76(8):1905–1915, 1971.

[42] Rolland L Hardy. Theory and applications of the multiquadric-biharmonic method 20 years of discovery 1968–1988. *Computers & Mathematics with Applications*, 19(8-9):163–208, 1990.

[43] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[44] Roy V Harris Jr. An analysis and correlation of aircraft wave drag. *National Aeronautics and Space Administration*, 1964.

[45] Oubay Hassan, K Morgan, EJ Probert, and J Peraire. Unstructured tetrahedral mesh generation for three-dimensional viscous flows. *International Journal for Numerical Methods in Engineering*, 39(4):549–567, 1996.

[46] Mark Hempsell. Progress on the skylon and sabre. In *Proceedings of the International Astronautical Congress*, volume 11, pages 8427–8440, 2013.

[47] Mark Hempsell, Julio Aprea, Ben Gallagher, and Greg Sadlier. A business analysis of a skylon-based european launch service operator. *Acta Astronautica*, 121:1–12, 2016.

[48] Mark Hempsell and A Bond. Skylon: An example of commercial launch system development. *JBIS-Journal of the British Interplanetary Society*, pages 11–12, 2014.

[49] Mark Hempsell and Roger Longstaff. The requirement generation process for the skylon launch system. *J Br Interplanet Soc*, 63:112–128, 2010.

[50] Raymond M Hicks and Preston A Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15(7):407–412, 1978.

[51] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[52] Andrew Hyslop, Luke J Doherty, Matthew McGilvray, Andrew Neely, Liam P McQuellin, James Barth, and Gerrie Mullen. Free-flight aerodynamic testing of the skylon space plane. *Journal of Spacecraft and Rockets*, 58(5):1487–1497, 2021.

[53] Plotly Technologies Inc. Collaborative data science, 2015.

[54] Stefan Jakobsson and Olivier Amoignon. Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization. *Computers & Fluids*, 36(6):1119–1136, 2007.

[55] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[56] Robert T Jones. Theory of wing-body drag at supersonic speeds. Technical report, 1956.

[57] Laurence Kedward, Alexandre D Payot, Thomas Rendall, and Christian B Allen. Efficient multi-resolution approaches for exploration of external aerodynamic shape and topology. In *2018 Applied Aerodynamics Conference*, page 3952, 2018.

[58] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[59] Patrick Knupp. Remarks on mesh quality. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2007.

[60] Patrick M Knupp. Algebraic mesh quality metrics for unstructured initial meshes. *Finite Elements in Analysis and Design*, 39(3):217–241, 2003.

[61] Lars Krog, Alastair Tucker, Martin Kemp, and Richard Boyd. Topology optimisation of aircraft wing box ribs. In *10th AIAA/ISSMO multidisciplinary analysis and optimization conference*, page 4481, 2004.

[62] Qunfeng Liu. Order-2 stability analysis of particle swarm optimization. *Evolutionary computation*, 23(2):187–216, 2015.

[63] Xueqiang Liu, Ning Qin, and Hao Xia. Fast dynamic grid deformation based on delaunay graph mapping. *Journal of Computational Physics*, 211(2):405–423, 2006.

[64] Harvard Lomax and Max A Heaslet. Recent developments in the theory of wing-body wave drag. *Journal of the Aeronautical Sciences*, 23(12):1061–1074, 1956.

[65] Roger Longstaff and Alan Bond. The skylon project. In *17th AIAA International Space Planes and Hypersonic Systems and Technologies Conference*, page 2244, 2011.

[66] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[67] Unmeel B Mehta, Michael J Aftosmis, Jeffrey V Bowles, and Shishir A Pandya. Skylon aerodynamics and sabre plumes. In *20th AIAA International Space Planes and Hypersonic Systems and Technologies Conference*, page 3605, 2015.

[68] Bijan Mohammadi and Olivier Pironneau. *Applied shape optimization for fluids*. OUP Oxford, 2009.

[69] Michael Mongillo et al. Choosing basis functions and shape parameters for radial basis function methods. *SIAM undergraduate research online*, 4(190-209):2–6, 2011.

[70] DS Naumann, B Evans, S Walton, and O Hassan. A novel implementation of computational aerodynamic shape optimisation using modified cuckoo search. *Applied Mathematical Modelling*, 40(7-8):4543–4559, 2016.

[71] DS Naumann, Ben Evans, Sean Walton, and Oubay Hassan. Discrete boundary smoothing using control node parameterisation for aerodynamic shape optimisation. *Applied Mathematical Modelling*, 48:113–133, 2017.

[72] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[73] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[74] International Civil Aviation Organization. *Manual of the ICAO Standard Atmosphere: Extended to 80 Kilometres (262 500 Feet)*. Doc (International Civil Aviation Organization). International Civil Aviation Organization, 1993.

[75] ADJ Payot, TCS Rendall, and CB Allen. Restricted snakes volume of solid (rsvs): A parameterisation method for topology optimisation of external aerodynamics. *Computers & Fluids*, 182:60–84, 2019.

[76] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[77] N Anders Petersson and Kyle K Chand. Detecting translation errors in cad surfaces and preparing geometries for mesh generation. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2001.

[78] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 1996.

[79] Michael JD Powell. Recent research at cambridge on radial basis functions. *New developments in approximation theory*, pages 215–232, 1999.

[80] Alma AM Rahat, Richard M Everson, and Jonathan E Fieldsend. Alternative infill strategies for expensive multi-objective optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 873–880, 2017.

[81] Carl Edward Rasmussen, Christopher KI Williams, et al. *Gaussian processes for machine learning*, volume 1. Springer, 2006.

[82] Shmuel Rippa. An algorithm for selecting a good value for the parameter c in radial basis function interpolation. *Advances in Computational Mathematics*, 11:193–210, 1999.

[83] Humberto Rocha. On the selection of the most adequate radial basis function. *Applied Mathematical Modelling*, 33(3):1573–1583, 2009.

[84] Jamshid A Samareh. A survey of shape parameterization techniques. In *CEAS/AIAA/ICASE/-NASA Langley International Forum on Aeroelasticity and Structural Dynamics 1999*, number Pt. 1, 1999.

[85] Robert Schaback. Multivariate interpolation by polynomials and radial basis functions. *Constructive Approximation*, 21(3):293–317, 2005.

[86] Robert Schaback. The missing wendland functions. *Advances in Computational Mathematics*, 34(1):67–81, 2011.

[87] Michael Scheuerer. An alternative procedure for selecting a good value for the parameter c in rbf-interpolation. *Advances in Computational Mathematics*, 34(1):105–126, 2011.

[88] Hermann Schlichting and Joseph Kestin. *Boundary layer theory*, volume 121. Springer, 1961.

[89] William R Sears. On projectiles of minimum wave drag. *Quarterly of Applied Mathematics*, 4(4):361–366, 1947.

[90] Thomas W Sederberg and Scott R Parry. Free-form deformation of solid geometric models. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 151–160, 1986.

[91] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[92] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE, 1998.

[93] Shaun N Skinner and Hossein Zare-Behtash. State-of-the-art in aerodynamic shape optimisation methods. *Applied Soft Computing*, 62:933–962, 2018.

[94] Ben Smith, Ben James Evans, and Sean Peter Walton. Development of transonic and supersonic cfd modelling and design optimisation techniques for spaceplanes. In *UKACM*, Nottingham, UK, 2022.

[95] Ben Smith, Ben James Evans, and Sean Peter Walton. Optimisation methods for computationally expensive design processes. In *UKACM*, Warwick, UK, 2023.

[96] Ben Smith, Ben James Evans, and Sean Peter Walton. Optimisation methods for computationally expensive design processes. In *15th International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control*, Chania, Crete, Greece, 2023.

[97] Ben Smith, Ben James Evans, and Sean Peter Walton. Optimisation methods for computationally expensive design processes. In *22nd Computational Fluids Conference.*, Cannes, France, 2023.

[98] Ben Smith, Ben James Evans, Sean Peter Walton, Neil Taylor, Martin Dodds, and Vladeta Zmijanovic. A novel mesh morphing methodology for computational aerodynamic shape optimisation. *In Preparation*, 2024.

[99] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[100] Jan Sokolowski, Jean-Paul Zolésio, Jan Sokolowski, and Jean-Paul Zolesio. *Introduction to shape optimization*. Springer, 1992.

[101] KA Sørensen, O Hassan, K Morgan, and NP Weatherill. A multigrid accelerated hybrid unstructured mesh method for 3d compressible turbulent flow. *Computational mechanics*, 31:101–114, 2003.

[102] Philippe Spalart and Steven Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th aerospace sciences meeting and exhibit*, page 439, 1992.

[103] C Stimpson, C Ernst, P Knupp, P Pébay, and D Thompson. The verdict library reference manual. *Sandia National Laboratories Technical Report*, 9(6), 2007.

[104] Bane Sullivan and Alexander Kaszynski. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, May 2019.

[105] Eleuterio F Toro, Michael Spruce, and William Speares. Restoration of the contact surface in the hll-riemann solver. *Shock waves*, 4:25–34, 1994.

[106] James Courtland Townsend, JA Samareh, RP Weston, and WE Zorumski. Integration of a cad system into an mdo framework. Technical report, 1998.

[107] Richard Varvill and Alan Bond. The skylon spaceplane. *Journal of the British Interplanetary Society*, 57:22–32, 2004.

[108] Richard Varvill and Alan Bond. The skylon spaceplane: progress to realisation. *Journal of the British Interplanetary Society*, 61(10):412–418, 2008.

[109] Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson education, 2007.

[110] Pradnya A Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC)*, pages 261–265. IEEE, 2016.

[111] Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in computational Mathematics*, 4(1):389–396, 1995.

[112] Richard T Whitcomb. A study of the zero-lift drag-rise characteristics of wing-body combinations near the speed of sound. Technical report, 1956.

[113] Frank M White. *Fluid mechanics*. New York, 1990.

[114] David C Wilcox et al. *Turbulence modeling for CFD*, volume 2. DCW industries La Canada, CA, 1998.

[115] Zongmin Wu. Compactly supported positive definite radial functions. *Advances in computational mathematics*, 4:283–292, 1995.

[116] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.