INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

DEPARTMENT OF AEROSPACE ENGINEERING

Framework for Multidisciplinary Optimization

DUAL DEGREE PROJECT

SUBMITTED IN FULFILMENT OF PHASE I OF DUAL DEGREE PROJECT IN THE DEPARTMENT OF AEROSPACE ENGINEERING

Submitted by:
M Vishnu Sankar
18B030013

Supervisor:
Professor Abhijit Gogulapati

27 October 2022

Contents

1 Introduction and Objectives **2**
1.1 Problem Overview . . . 2
1.1.1 Aerodynamics Module . . . 2
1.1.2 RCS Evaluation Module . . . 3
1.1.3 Flight Mechanics and Controls Parameter Estimation Module . . . 3
1.1.4 Optimization Framework . . . 3
1.2 Report Overview . . . 4

2 Literature Review of Modern Optimization **5**
2.1 Background . . . 5
2.1.1 Extended Design Structure Matrix . . . 5
2.2 Architecture types . . . 6
2.2.1 Monolithic Architectures . . . 6
2.2.2 Distributed Architecture . . . 7
2.3 Type of Solvers . . . 7
2.4 Gradient based solvers . . . 8
2.5 Gradient Free solvers . . . 8
2.5.1 Advantages . . . 8
2.5.2 Disadvantages . . . 8
2.5.3 Choice of Solver . . . 9
2.5.4 Solver selection . . . 10

3 Geometry Parameterization **11**
3.1 2D Parameterization . . . 11
3.1.1 Class-Shape Function Transformation . . . 11
3.1.2 Hicks-Henne Bump Function . . . 12
3.1.3 Selection of Airfoil Parameterization . . . 13
3.1.4 3D Free form Deformation . . . 13

4 OpenMDAO **15**
4.1 OpenMDAO . . . 15
4.2 Justification for Selecting OpenMDAO . . . 16

5 Problem Description and Optimization Framework **18**
5.1 Problem Description . . . 18
5.1.1 Challenge 01: Number of Unknowns . . . 18
5.1.2 Challenge 02: Interfacing with Different Analysis Tools . . . 19
5.2 Philosophy of the Optimization Framework . . . 20

6 Implementing and Testing the Framework **25**

6.1 Framework for 2-D Aerodynamics 25

6.1.1 Training the Surrogate Model 27

6.2 Preliminary Results 27

6.2.1 Problem 1: Unconstrained Optimization 27

6.2.2 Problem 2: Constrained Optimization 28

7 Future Work **30**

List of Figures

1 XDSM of a Optimization problem | *Source: Lambe and Martins [5]* 6
2 MDAO Architecture 6
3 XDSM of IDF and MDF Architecture | *Source: Lambe and Martins [5]* 7
4 Functional evaluation comparison | *Source: Martins and Ning [7]* 9
5 Computational cost comparison | *Source: Martins and Ning [7]* 9
6 Bernstein Polynomials 11
7 CST functions with n=10 and coefficients of v=1 | *Source: Sóbester & Forrester [9]* 12
8 Bump function at $x = 0.2$ 13
9 FFD of Sphere 14
10 Herirarchy of code structure | *Source Gray and Martins [1]* 16
11 Bird's view of the Framework 20
12 Code Structure and Execution Order with Xfoil 26
13 Code Structure and Execution Order with Surrogate Model 26
14 Optimized Airfoil using different analysis tools 27
15 Constrained Optimization 28

Abstract

An attempt to build a framework for multidisciplinary optimization on top of an existing tool, OpenMDAO, with a goal to optimize an aircraft's performance is presented in this report. Such a framework is expected to be solver/code agnostic, provide a simple and efficient interface for the user, and accommodate different optimization architectures that give users the flexibility to solve various problems. This report discusses the key features, the code architecture, and the relevant tools in detail to develop the robust framework. A survey of the choice of solvers and geometric parameterization is presented. The philosophy for building the framework are discussed in depth. The key challenges encountered are addressed and crucial decisions that were taken are also presented. Finally, the framework is developed for 2D aerodynamics problem, its ability to be solver agnostic and seamlessly interface with two different solvers are highlighted.

Keywords: Multidisciplinary Optimization, OpenMDAO, Code Agnostic, MDAO Architecture, Parameterization

1 Introduction and Objectives

Optimization has long roots in the history of aviation, and it is no wonder that optimization, once from being a mere human intuition, has evolved into a significant branch of modern mathematics and computational science. Thanks to today's aviation sector, which continues to push the frontier of optimization. Not only is optimization relevant to engineers but also to physicists, as quoted by Euler "Nothing at all takes place in the universe in which some rule of maximum or minimum does not appear" [7].

Optimization precisely means "finding the best possible solution by changing variables that can be controlled, often subject to constraints [7]." There is no field where optimization is not relevant. In today's world with complex engineering systems, optimization is more needed than ever. It is a crucial phase of all modern-day engineering systems, and countless algorithms have been formulated to optimize such systems, thanks to the advent of electronic technology and computational prowess. Such optimization algorithms require crucial information about the system, generally provided by analysis codes, solvers, or programs. The analysis programs solve partial differential equations that govern engineering problems to provide the relevant information required by the optimization algorithms. Such analysis programs are available both as open-source and commercial tools, generally built as a standalone executable to provide information of only a few disciplines taken at a time. However, an engineering system is a combination of more than just a few disciplines, due to which the optimization algorithm requires information from multiple disciplines. Hence, there is a need for a robust framework that can independently and seamlessly communicate with any standalone analysis tool to acquire the necessary information and provide them to the optimization algorithms. This project attempts to create a robust framework on top of an open-source optimization tool, OpenMDAO, specifically built to optimize engineering designs. Such a framework is expected to be solver/code agnostic, provide a simple and efficient interface for the user, and accommodate different optimization architectures that give the user the required flexibility to solve a variety of problems.

1.1 Problem Overview

The scope of this work is to build a robust framework to carry out a multidisciplinary optimization of a flying wing configuration. The goal is to optimize the aircraft configuration with respect to three major disciplines: Aerodynamics, Stealth, and Flight mechanics and control with constraints on layout, propulsion, and structural parameters. So the MDO framework needs to contain the following:

- Aerodynamics Module
- Radar Cross Section Evaluation Module
- Flight mechanics and Control parameters estimation module
- Optimization framework that integrates every other module

1.1.1 Aerodynamics Module

The aerodynamics module evaluates the aerodynamic coefficients and performance parameters of the configuration. The possible objective functions of the aerodynamics module are $C_{D,min}$,

$C_{L,max}$, $\frac{C_L}{C_D}$, $(\frac{C_L}{C_D})^{\frac{1}{2}}$, Range, endurance and maximum speeds at various altitudes. Stability and control parameters have to be estimated using the aerodynamics module. The weightage functions for various parameters are yet to be decided.

Preferred Tool: Flightstream. A commercial tool which is a subsonic, inviscid, surface-vorticity based panel solver coupled with boundary layer evaluation. OpenVSP is an open-source tool with most of the features in Flightstream and can be used instead of Flightstream for developing and validating the proof of concept.

1.1.2 RCS Evaluation Module

The intake in the RCS calculation is considered to be a fixed entity. The RCS is to be estimated from 2-12 GHz from S-band to X-band, for a monostatic radar in both H-H and V-V polarization. The medial RCS, along with the standard deviation, must also be evaluated. The frontal, side, and rear sector azimuth, elevation angle ranges, and weightage functions for various parameters are yet to be decided.

Preferred Tool: ANSYS HFSS is expected to be used for all the computational calculations.

1.1.3 Flight Mechanics and Controls Parameter Estimation Module

Major flight mechanics parameters such as time to double, phase and gain margins for closed loop systems are expected to have a direct impact on the configuration. Any open-source software capable of evaluating the parameters mentioned can be chosen accordingly.

1.1.4 Optimization Framework

The framework should be able to call and run the individual modules as and when required seamlessly. The core optimizer needs to predict optima or a family of optima that are good enough. The inputs and outputs need to be developed for each module. This framework should have the following key features:

1. **Solver Agnostic**
    - The code should be able to communicate seamlessly with any solver, irrespective of its nature. This is necessary as the solvers can be replaced at a later time. Hence the framework should provide an interface that is generic and compatible with any solver
2. **Simplicity and Robustness**
    - Atmost not more than two files are to be given as input by the user to solve any multidisciplinary problem without needing to fiddle with the source code
    - Code structure should remain independent of the dimension of the problem (2D or 3D)
3. **Modular programming**

   - Modularity is a must. It facilitates faster debugging and makes the source code accessible. Codes can be built on top of each other without having to affect other code structure

4. **Scalability and Flexibility**
   - The framework should accommodate any number of objective functions, constrain equations, and design variables to incorporate any segment(s) of the mission profile

5. **Efficient Gradient Evaluation**
   - The framework have different methods to compute gradients by leveraging the structure of the analysis tools. The choice of gradient based algorithms is justified later.

Preferred Tool: OpenMDAO

1.2 Report Overview

The report is divided into five chapters, the second chapter beginning with the literature review of modern optimization. It introduces a general setup of a multidisciplinary problem, the notations used, and the various kinds of architecture used for engineering applications. A summary of solver types, advantages and pitfalls of each type is discussed, and the choice of solver for an MDO problem is justified. A survey of the available tools is presented before justification of the final selection

The third chapter presents a study of standard techniques for parametrizing any geometry in 2D and 3D, along with their mathematical formulation. The chapter discusses the importance of choosing relevant parametrization techniques based on the desired objective of the user (CST reduces space, but more sensible airfoils. Bump function for more generalization but ad-hoc constraints to prevent solver divergence). Relevant simulations from the author's experience are also presented to justify the choice of final parametrization.

The fourth chapter is more focused on the agnostic structure of the framework relevant to this problem, the features of the tool (openMDAO) used to build it and discusses the key decisions that need to be taken to build a robust framework.

The final chapter showcases a few validation problems that establish the seamless nature of the framework and its ability to work independently of the problem's dimensionality. The results obtained are discussed and verified against published journals.

2 Literature Review of Modern Optimization

2.1 Background

Before delving into the mathematical background we introduce the notation used based on work of Martins and Lambe[8]

$$\begin{aligned}
&\textit{minimize} && f_0(\mathbf{x},\mathbf{y}) + \textstyle\sum_{i=1}^{N} f_i(\mathbf{x}_0,\mathbf{x}_i,\mathbf{y}_i) \\
&\textit{with respect to} && \mathbf{x},\hat{\mathbf{y}},\mathbf{y},\bar{\mathbf{y}} \\
&\textit{subject to} && c_0(\mathbf{x},\mathbf{y}) \\
& && c_i(\mathbf{x_0},\mathbf{x_i},\mathbf{y_i}) \\
& && c_i^c = \hat{\mathbf{y}}_\mathbf{i} - \mathbf{y_i} = 0 \\
& && R_i(\mathbf{x_0},\mathbf{x_i},\mathbf{y_{j\neq i}},\bar{\mathbf{y}}_\mathbf{i},\mathbf{y_i}) = 0
\end{aligned} \tag{1}$$

$\mathbf{x}$	=	vector of design variables
$\mathbf{x_0}$	=	vector of design variables shared by more than one discipline
$\mathbf{x_i}$	=	vector of design variables of i^{th} discipline
$\bar{\mathbf{y}}$	=	vector of state variables
$\hat{\mathbf{y}}$	=	independent copies of vector of coupling variables
$\mathbf{y}$	=	vector of coupling variables
R	=	governing equation in residual form
c	=	vector of design constraints

Eqn:1 is the most generic formulation of any optimization problem

2.1.1 Extended Design Structure Matrix

Next we present a formulation of Extended Design Structure Matrix proposed by Lambaste and Martins [5], a specific type of visual architecture diagram or a flow chart that gives a bird's eye view of any MDO problem.

1. Optimizer, any type of solver or in general any component that performs any computation task by processing the data is present along the diagonal
2. The flow of data is shown as thick gray coloured lines
3. Vertical directions are reserved for inflow of data into a component
4. Horizontal directions are reserved for outflow of data from a component
5. All the connections above the diagonal represents data flowing inside the components
6. All the connections below the diagonal represents data flowing outside the components
7. Parallelograms are used to label the data or copies of data
8. Thin black line indicates the direction of the optimization process
9. The numbering indicates the order in which the components are executed

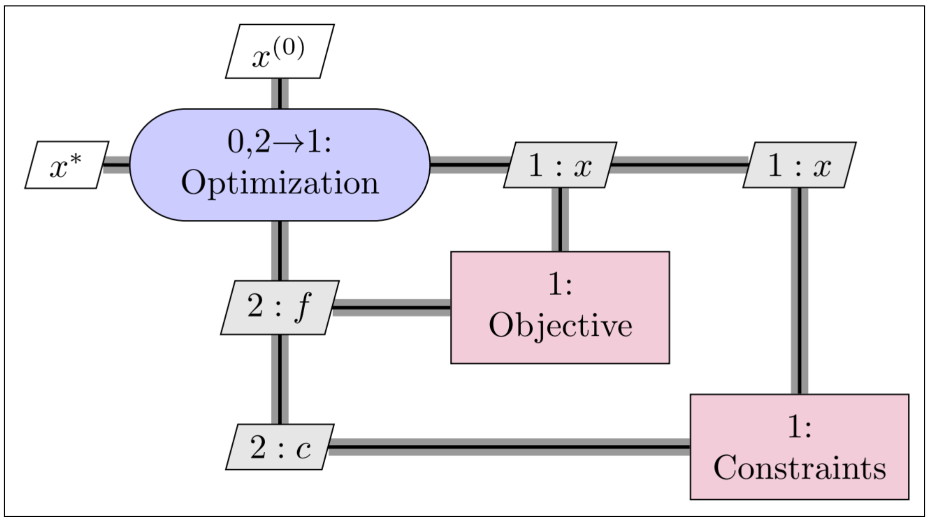


Figure 1: XDSM of a Optimization problem | *Source: Lambe and Martins [5]*

2.2 Architecture types

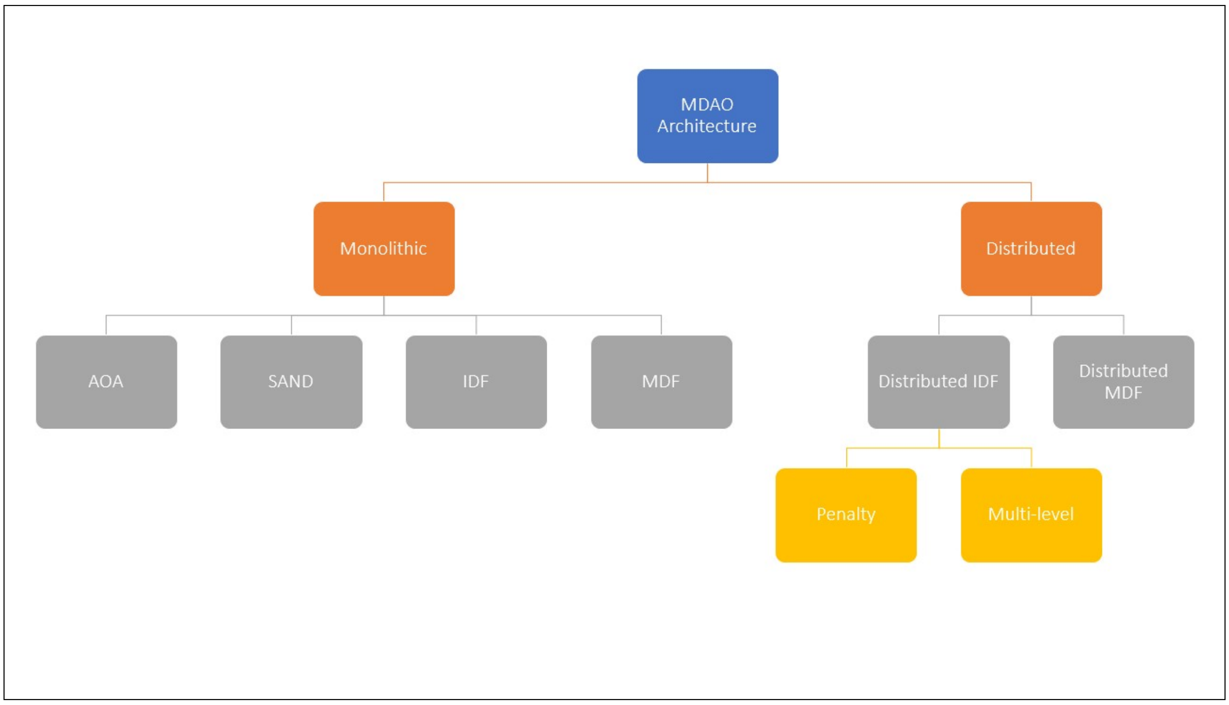


Figure 2: MDAO Architecture

Architecture is an approach or method to solve any MDO. The same problem can be solved with different architectures, but the optimal solution set remains unchanged. Such architectures are classified under two categories: Monolithic and Distributed [8]

2.2.1 Monolithic Architectures

Monolithic Architectures: These architectures formulate a single unified Obj function from all disciplines. The AAO and SAND architectures are very similar and share the following features:

- The design variables are the same as state variables (for e.g. the nodes of a CFD mesh are the state variables, and so are the design variables)
- It solves the governing equation of the analysis tools as a constraint equation in terms of a residual equation. This requires access to the source code of all the analysis tools, which is a huge disadvantage as it does not allow black box-type solvers to be used.

By removing the state variables as design variables and residual form of governing equations as a part of the constraint equation, we obtain Individual Discipline Feasible (IDF) and Multidisciplinary Feasible (MDF) architecture. The main difference between the two is MDF solves the analysis in sequential order while IDF solves all the analysis independent of the disciplines involved. For e.g., optimizing a fluid static interaction problem falls under MDF, while solving a decoupled aero-structural problem is IDF.

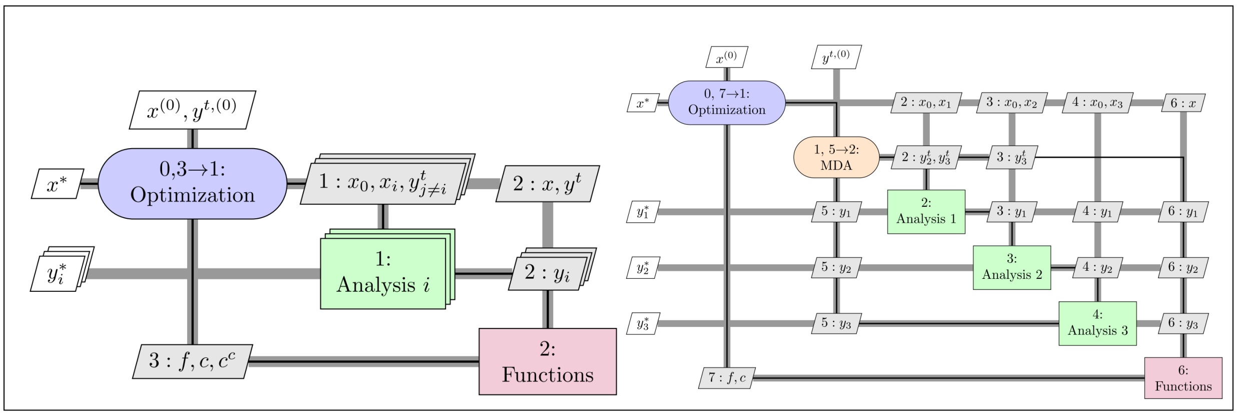


Figure 3: XDSM of IDF and MDF Architecture | *Source: Lambe and Martins [5]*

2.2.2 Distributed Architecture

Distributed Architecture solves multiple subproblems of a larger optimization problem which, when assembled together, has the same solution as the larger problem. This is only feasible in problems where there are no shared design variables among the disciplines, the objective functions are separable, and the subproblems can be optimized to a solution that co-exists and are independent of each other. Since the problem of this project needs a single solution satisfying different disciplines, it cannot be tackled by Distributed Architecture, so the detailed literature on this type of architecture is avoided in this report. Interested readers can refer to [8] for a comprehensive study.

2.3 Type of Solvers

Once an optimization problem is setup and the details of the problem are well understood, such as the expected number of design variables from all the disciplines, type and number of objective functions, and constraints equations, one can go ahead and choose a solver that is suitable for the kind of problem. There is no *"holy grail"* or one single optimization scheme that works the best for all problems. A lot of thought is put into selecting a suitable solver, and some key factors that drive this decision should also account for the fact that there are real-life constraints from the user's end, such as low computation power or less sophisticated resources.

Broadly any solver can be categorized into either gradient-based or gradient-free solvers, the advantages and pitfalls of the two kinds of solvers are discussed in the next section of this chapter.

2.4 Gradient based solvers

Gradient-based algorithms start from a guess point and use the gradient (or sometimes hessian also) to find a direction to move along to find an optima. No prior assumption regarding the modality of the design space is assumed. However, the objective functions(s) must be at least C^2 continuous (at least close to the optima) and deterministic. So these methods perform a local search, and it is not guaranteed to find global optima. These methods have a solid mathematical formulation, unlike the heuristic methods.

2.5 Gradient Free solvers

As the name suggests, Gradient free solvers do not use the gradient information or the hessian of a function. Here only a brief comparison between gradient free and gradient based solver is presented.

2.5.1 Advantages

- Since only function evaluation is required and not the smoothness of the function, these algorithms are robust towards discontinuities or functions with high numerical noises

2.5.2 Disadvantages

- Computational cost is directly proportional to the number of function evaluations which increases drastically as the number of design variables increases
- Need a significantly high number of iterations for convergence compared to gradient-based algorithms
- Many methods do not directly tackle constrained problems and require filtering methods or penalty methods to enforce constraints

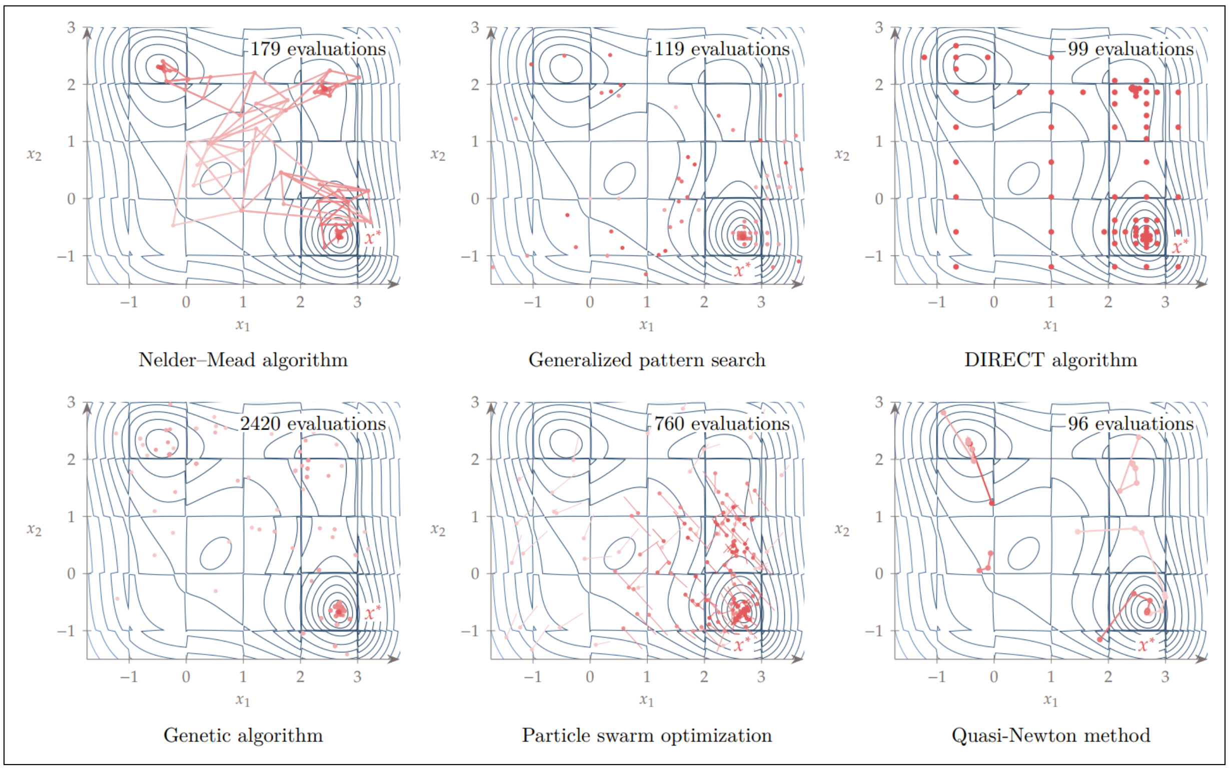


Figure 4: Functional evaluation comparison | *Source: Martins and Ning [7]*

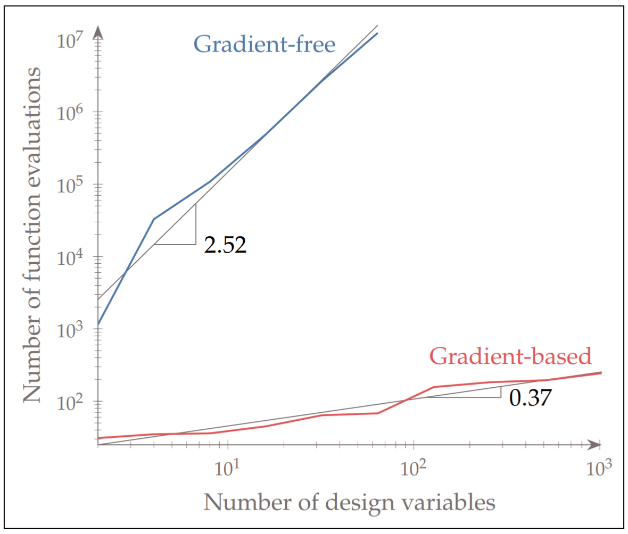


Figure 5: Computational cost comparison | *Source: Martins and Ning [7]*

2.5.3 Choice of Solver

- Gradient free methods can be preferred if objective functions and constraints functions have high numerical noise or if they are highly non-smooth functions whose gradients are almost impossible to compute
- If the dimension of the problem is too large, gradient-based methods are not only computationally inexpensive, but a multi-start approach can be used to explore the entire design space to increase the likely hood of a global optima

2.5.4 Solver selection

It is clear now that gradient based methods have a solid mathematical formulation and are computationally more efficient than gradient free methods by orders of magnitude. This clearly establishes the superiority of gradient based methods for well behaved continuously differentiable functions. Hence, gradient based methods are selected to tackle this problem. Before moving on to the formulation of the problem statement, an introduction to the geometric parameterization and the code framework of OpenMDAO are presented.

3 Geometry Parameterization

Since aerodynamic characteristics and radar cross-section area are functions of geometry, parameterizing it is crucial to explore the entire design space. Here geometry refers to both shape and size. This chapter of the report is derived mostly from the book Aerodynamic Design Geometry and Optimization by András Sóbester and Alexander I J Forrester [11]. Bernstein polynomials are often used to build other parameterization techniques that exhibit the above characteristics. The basis of the Bernstein polynomial of degree *n* is defined by

$$b_{v,n} \; = \; {}^{n}C_{k}\, x^{v}(1-x)^{n-v}, \;\; v = 0, 1 \ldots n \tag{2}$$

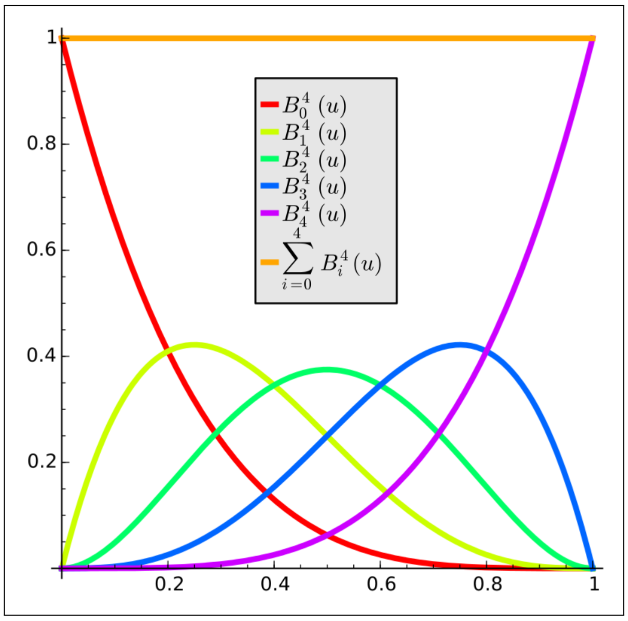


Figure 6: Bernstein Polynomials

It is also the basis of the polynomial vector space of functions of the order N. With the above features in mind, a few 2D parameterization techniques are introduced for the problem, and a widely used 3D parameterization technique is introduced which will be used for a 3D problem later in the next phase.

3.1 2D Parameterization

Almost all the parameterization techniques can be classified under two categories: Constructive and Destructive methods. While constructive methods design the geometries based on polynomial functions, destructive methods perturb an existing baseline design to create a modified shape. There are tonnes of techniques in each category, in this report, one suitable and efficient technique from each of the methods is introduced and used for the airfoil optimization problem.

3.1.1 Class-Shape Function Transformation

Introduced by Brenda Kulfan, Class-Shape Function Transformation [4] falls under the constructive parameterization category. The equation describing the upper surface of the airfoil is:

$$z^u(x, v_0^u, v_1^u, ..., v_n^u, z_{TE}^u, v_{LE}^u) = \sqrt{x}(1-x)\sum_{r=0}^{n} v_r^u \, {}^rC_n x^r(1-x)^{n-r} + z_{TE}^u x + x(\sqrt{1-x})v_{LE}^u(1-x)^n \tag{3}$$

where the superscript *u* stands for upper. The equation for the lower surface of the airfoil is the same with *u* replaced by *l* (lower). The class functions are $\sqrt{x}(1-x)$ and $x\sqrt{1-x}$. The shape is expressed as a product of the class function and a linear combination of the Bernstein polynomials. The square root term ensures that the leading edge of the airfoil is rounded. Each term affects the entire shape, but the effect is dominant only at a local spatial position, and it decays rapidly everywhere else. The lower-order Bernstein polynomials describe the nose region, while higher orders describe the aft part of an airfoil. The individual term of the summation is the basis function. An example of visualizing an airfoil is shown below

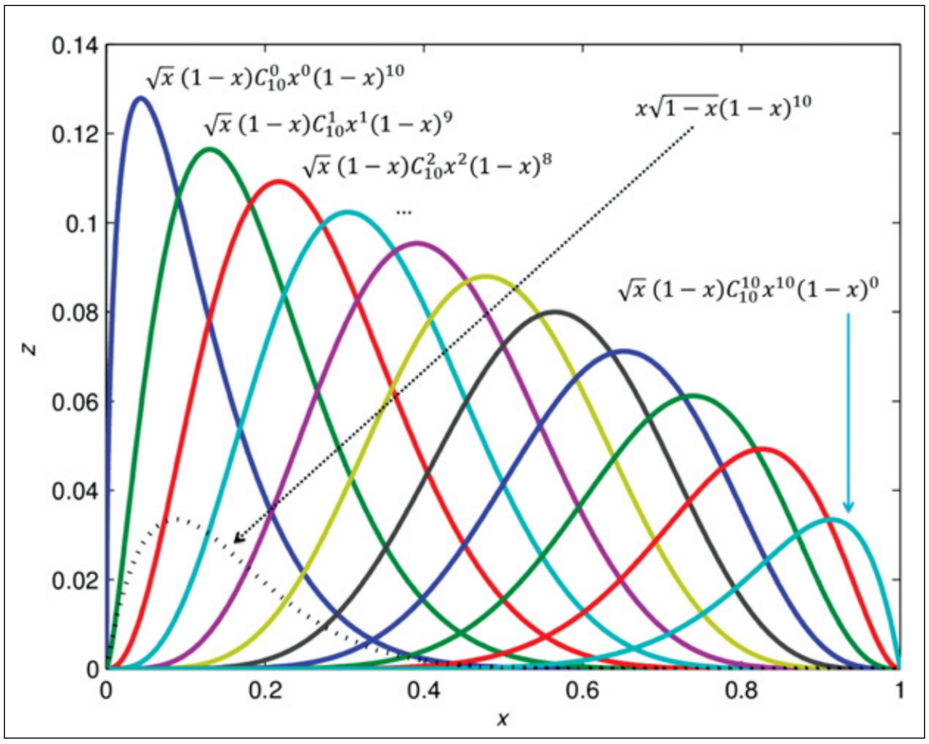


Figure 7: CST functions with *n*=10 and coefficients of *v*=1 | *Source: Sóbester & Forrester [9]*

The critical consequence of the CST parameterization is that any combination of the polynomials and their coefficients always results in an airfoil-like shape.

3.1.2 Hicks-Henne Bump Function

Hicks and Henne proposed a destructive parameterization technique that modifies the baseline airfoil geometry by introducing a bump in it at defined locations [2]. The amplitude of the bump decays quickly except at the specified location.

$$z^{n+1} = z^n + \textstyle\sum_{i=0}^{n} a_i\phi_i(x)$$

$$\phi_i(x) = sin^{t_i}(\pi x^{\frac{ln(0.5)}{ln(h_i)}}) \tag{4}$$

where hi is the location of the maxima and t_i is the width of the function usually taken as 3 for airfoils. Once the suitable locations on the airfoils are chosen, a_i remains the only design variable. The value a_i controls the amplitude of the deformation. The disadvantage is a few trial experiments need to be implemented to decide the range of a_i, naïve values can result in a sharp leading edge that can be catastrophic to the solver, and sometimes certain ad-hoc constraints are also required to ensure only airfoil-like shapes are generated.

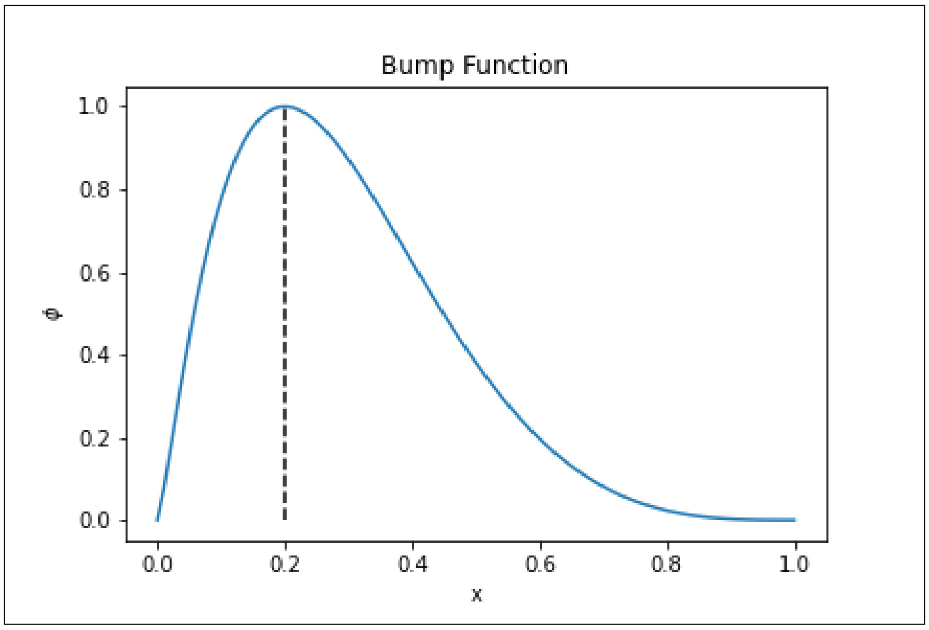


Figure 8: Bump function at $x = 0.2$

3.1.3 Selection of Airfoil Parameterization

Research done by Masters et al. [9] compares different parameterization techniques based on the number of design variables required by each of them. The method indicates that almost all the techniques, requires about the same number of design variables to accurately reconstruct any typical airfoil available in the UIUC airfoil database. For the aerodyanmics problem discussed in chapter 6, the bump function is used.

3.1.4 3D Free form Deformation

Changing the airfoil geometry at each spanwise location is computationally prohibitive at more of a brute-force approach. An ingenious way proposed by Sederberg and Parry [10] deforms the geometry locally by constructing bezier curves at the mesh points. Appropriate locations at the geometry are identified first, and the relevant number of mesh nodes are selected to fit the Bezier curves such that the surface generated by them exactly fits on the surface of the wing/geometry chosen. E.g., the FFD on a grid of $m \times n \times \times p$ is of the form:

$$X(u,t,s) = \sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{p} a^{(i,j,k)} f_i(u) g_j(t) h_k(s) \tag{5}$$

$A_{i,j}$ are the control points or weights, and f,g, and h are the Bernstein polynomials. Once the function defines the baseline shape, the weights $a_{i,j,j}$ can be perturbed to deform the geometry locally and obtain a different geometry. An example from [11] is reproduced below based on the algorithm provided:

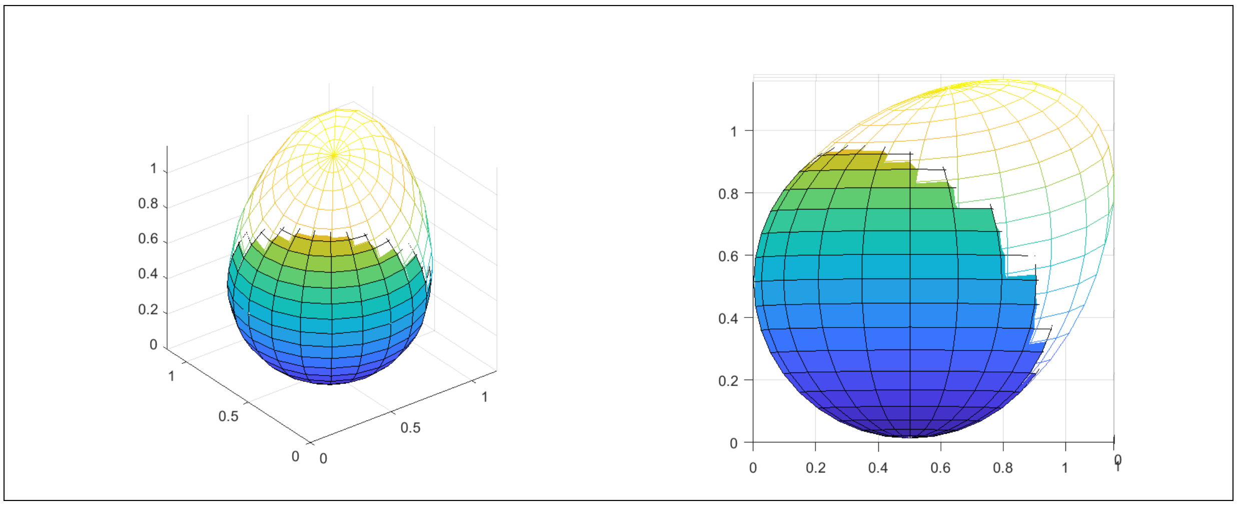


Figure 9: FFD of Sphere

The left half is the undeformed sphere and the right half of the sphere is the deformated one. This technique is widely used for 3D optimization, especially for wings and fuselages. All the techniques discussed above will be used whenever required for the problem statement.

4 OpenMDAO

OpenMDAO is an open-source MDO tool developed by Justin Gray et al. from NASA Glenn Research center [1] that provides a framework to solve an optimization problem on any complex engineering system. Engineers and researchers working in diverse fields, such as aircraft, nano-satellites, wind turbines, and turbofan engines, to name a few, have already used OpenMDAO for optimizing the complex system with 5000+ design variables up to 25,000+ design variables, as mentioned by Hang and Martins [3]. One of the key reasons behind the successful implementation of OpenMDAO is its architecture supporting the modular analysis and unified derivatives (MAUD) proposed by John T. Hwang and Martins [3]. In this chapter only some of the key features of OpenMDAO are discussed

4.1 OpenMDAO

Almost all the commercially available so-called 'MDO' software use "Flow-based Computational frameworks" that provides a GUI, built-in solvers, and ways to couple heterogeneous models. The pitfalls of the tools that follow a flow-based computational framework are listed below:

1. Inefficiency in converging the coupled models and computing the derivatives for gradient-based methods
2. Relatively low number of inputs and outputs
3. Optimizer solves the coupling between disciplines, thus limiting the scalability of the optimizer
4. Do not scale well with the total number of state variables
5. Limited to fixed point iteration (nonlinear Gauss-Siedel) for coupled models
6. Limited to finite differencing for derivatives

The potential drawbacks of the commercial tools are points 4, 5, and 6. This is not surprising since most commercial solvers are not open-source, so access to source code is not always available. By using MAUD architecture, OpenMDAO overcomes the issues mentioned above by adopting the following features, which are directly quoted from Gray and Martins [1]:

1. The architecture allows efficient parallel processing
2. Hierarchically solves the nonlinear and linear systems
3. Adopts a matrix-free approach
4. Incorporates preconditioning
5. Various methods to calculate derivatives
6. Also implements Newton-type methods for strongly coupled models
7. Implements analytical derivatives if the residuals can be made available from the solver, even if it is a black box solver
8. Exploits sparsity in the structure to compute derivatives

OpenMDAO has the following hierarchy (from lowest to highest) in its code architecture.

1. **Component:** Any discipline analysis (usually one discipline) or a set of calculations are encapsulated inside a component. It could be as simple as y=tan(x) or a complex discipline like a CFD or FEA analysis.
2. **Group:** Group is a collection of components or even other groups, that links the coupling variables across multiple disciplines
3. **Driver:** The core optimizer consisting of various algorithms for number crunching is the Driver class.
4. **Problem:** The top-level enclosure that consists of the driver class, groups, and components is the problem class.

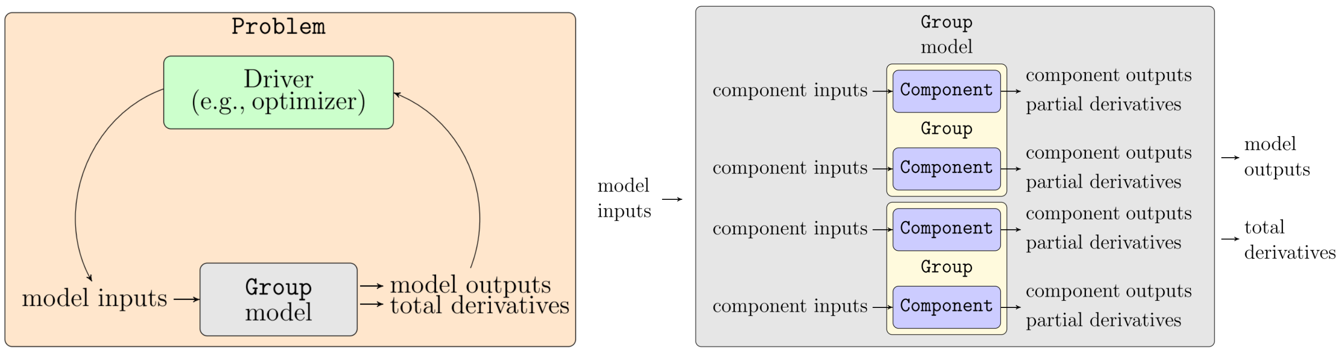


Figure 10: Herirarchy of code structure | *Source Gray and Martins [1]*

Derivative computation is one of the critical challenges in gradient-based methods. OpenMDAO provides three different methods to compute the gradients, which are:

1. Finite Differences and Complex-step
2. Analytical methods (Direct and Adjoint based)
3. Semi-analytical (Combination of Finite Differencing and Analytical methods)

One of the defining features of OpenMDAO is its analytical methods to calculate gradients of strongly coupled systems. If the analysis tools have open source codes, such as OpenFOAM or ADflow, their code structure is leveraged to compute the gradients using Adjoint formulation. If the analysis tools are completely black box except that they can reveal the residuals and state variables, which most black box solvers allow, a semi-analytical formulation is used to calculate the gradients. This method developed by Marriage and Martins [6] has proved to outperform the conventional finite differencing approach of the entire black-box solver.

4.2 Justification for Selecting OpenMDAO

One of the required features of the Optimization framework, as discussed in chapter 1, is **Scalability and Flexibility**. As mentioned above, OpenMDAO can handle several thousand design variables and constraint functions, at a time, in a vectorized format efficiently.

The hierarchy of OpenMDAO code architecture, discussed in the above section, naturally promotes modular programming as its most desirable trait. Modular programming is a crucial feature for developing the framework for the problem statement and hence makes OpenMDAO a desirable option.

The unique approach of semi-analytical approach, is efficient in terms of both accuracy and computational cost, is available in OpenMDAO. Commercial optimization tools only perform finite differences for computing gradients. This is expensive since the analysis tools need to be run for every finite differencing, and the inaccuracies/loss of precision incurred due to finite differencing often leads to sub-optimal results.

The above unique features of OpenMDAO agree with three out of five defining characteristics of the optimization framework outlined at the end of chapter 1. These features justify the selection of OpenMDAO as a relevant tool to build the rest of the required features. In the next section, the agnostic framework is explained in depth.

5 Problem Description and Optimization Framework

In the first phase of this project the primary goal is to build and test a framework that gives a plug and run feature to any analysis code/solver with as minimal change as possible to the framework. In the subsequent sections all the key challenges to build such a framework are identified and a methodological process is described to build the framework. Some crucial decisions that were taken are also mentioned.

5.1 Problem Description

Optimization problems which need to be solved using the Framework are general in nature. Some information regarding the problems are not known prior to creating the framework. Two key challenges that need to tackled while building the framework are categorised below. Both the challenges are described in detail and plausible ways to tackle them are also discussed.

5.1.1 Challenge 01: Number of Unknowns

The following quantities are either partially known or completely unknown and the number of such unknowns are also unknown.

- Number of disciplines involved and how many of them are coupled
- Nature of and number of design variables
- Nature of and number of coupling variables
- Number of objective functions, equality and inequality constraints

So the framework needs to be generic enough to accommodate any number of disciplines, design variables, objective functions and constraint equations.

Issue 01: OpenMDAO already solves the problem of number of design variables. It can accommodate any number of unknowns at any scale. However, the information regarding the number of variables are usually hard-coded while dealing with a specific optimization problem in OpenMDAO.

Issue 02: The code structure already provided by Openmdao is generic enough to handle any number of disciplines. But the nature of the disciplines are also hard-coded generally for a given problem. That is, the disciplines are predetermined and structured in a way that is efficient in solving a particular problem even before setting up the unknowns of that problem.

Solution Methodology:

1. The number of unknowns from the user is not taken as input, rather the input is directly taken from them and based on it the unknowns are calculated dynamically through a subroutine.
2. Create a general code structure that allows an easy way to integrate additional disciplines if required

A **key decision** is already made here. The number of disciplines, even if they are unknowns, need to be embedded in the framework giving the users an option to turn on or off the disciplines of their choice. So if a user needs an entirely new discipline that is not already present in the framework, it needs to be built and embedded in the framework manually. This is an extra task that cannot be automated. It can be automated if and only if all the source-code of the analysis tools are available open-source, which is not the case. Users may prefer to use commercial tools for some analysis and open-source tools for the rest. To promote this flexibility the process of creating a discipline in itself is not an automatic task. It is the limitation of unavailability of open-source codes for all the analysis tools and the user's preference to use any tool of their choice that made us take the decision of creating and embedding each discipline manually. It is to be noted that there is no limitation on the number of disciplines that can be added. Additionally, the code structure of all the disciplines are similar. So this similarity can be leveraged to add new discipline(s). This decision also allows us to create modular code structure for each discipline which is quite handy while debugging. In short, adding a new discipline does not need work from scratch or ground zero but needs to be done manually through a modular approach.

5.1.2 Challenge 02: Interfacing with Different Analysis Tools

The framework needs to seamlessly communicate with any analysis tool. For instance, replacing analysis tool A with B to solve the governing equations of i^{th} discipline should not even modify a line of code at the optimization framework. The user may prefer to use specific tools under certain circumstances. For example, if a designer is solving an aerodynamic shape optimization problem which, let's say, is to optimize a wing design at subsonic speeds, using a low fidelity solver such as panel methods or vortex lattice methods. These solvers can give optimized design at a fraction of cost that would take for a RANS solver due to their simplified physics. But if the same wing is intended to be optimized at transonic conditions, the user may prefer to optimize it using a high fidelity solver in order to capture non-linear shock regions and highly turbulent flows. Or the user may altogether switch to employ a surrogate model instead of any solvers. So based on the physics of the problem, constraint of resources on the user's end any analysis tool may be used. The framework needs to work independent of such analysis tools.

Solution Methodology: In short, the solver used to evaluate the state variables of a discipline is irrelevant to the framework. The accuracy, fidelity and robustness of a solver does not matter to the framework. This is defined as a **code agnostic** framework. Each analysis tool is created in a specific language, some of the widely used languages are C, C++ and FORTRAN, each of them having their own API's with Python. The way to go about it is described in a few steps below:

1. Each solver is expected to have an application programming interface (API). The framework will only communicate with the API in-order to execute the Solver whenever required. In a way we have decoupled the direct interaction between the framework and the solver through an API.
2. In case the problem demands to replace a solver with another solver, then only the API needs to be changed along with a set of predefined commands to integrate the API with the Framework. In this way the code agnostic features are preserved within the framework.
3. Once the solver outputs the simulation results to output files, another subroutine is in-

voked to read the data and provide the necessary information to the optimization framework.

A thorough and detailed discussion is presented in the next section to elaborate on various intricacies that are mentioned implicitly in the solution methodology.

5.2 Philosophy of the Optimization Framework

The desired deliverable from the core optimizer is to predict optima or a family of optima that are good enough. Further, it is also necessary for the framework to function completely independent of the analysis tools. **Code agnostic** and **Modularity** are at the heart of this framework. This section delves deeper to explain how modularity is used to achieve code agonstic framework. A flow chart is presented to show bird's eye view of the entire framework. All the gray arrow heads indicate the direction of optimization process, by default it also represents the direction of flow of data unless stated otherwise.

For four key blocks are identified:

1. OpenMDAO (core optimizer)
2. Discipline Manager
3. Outflow Processor
4. Inflow Processor

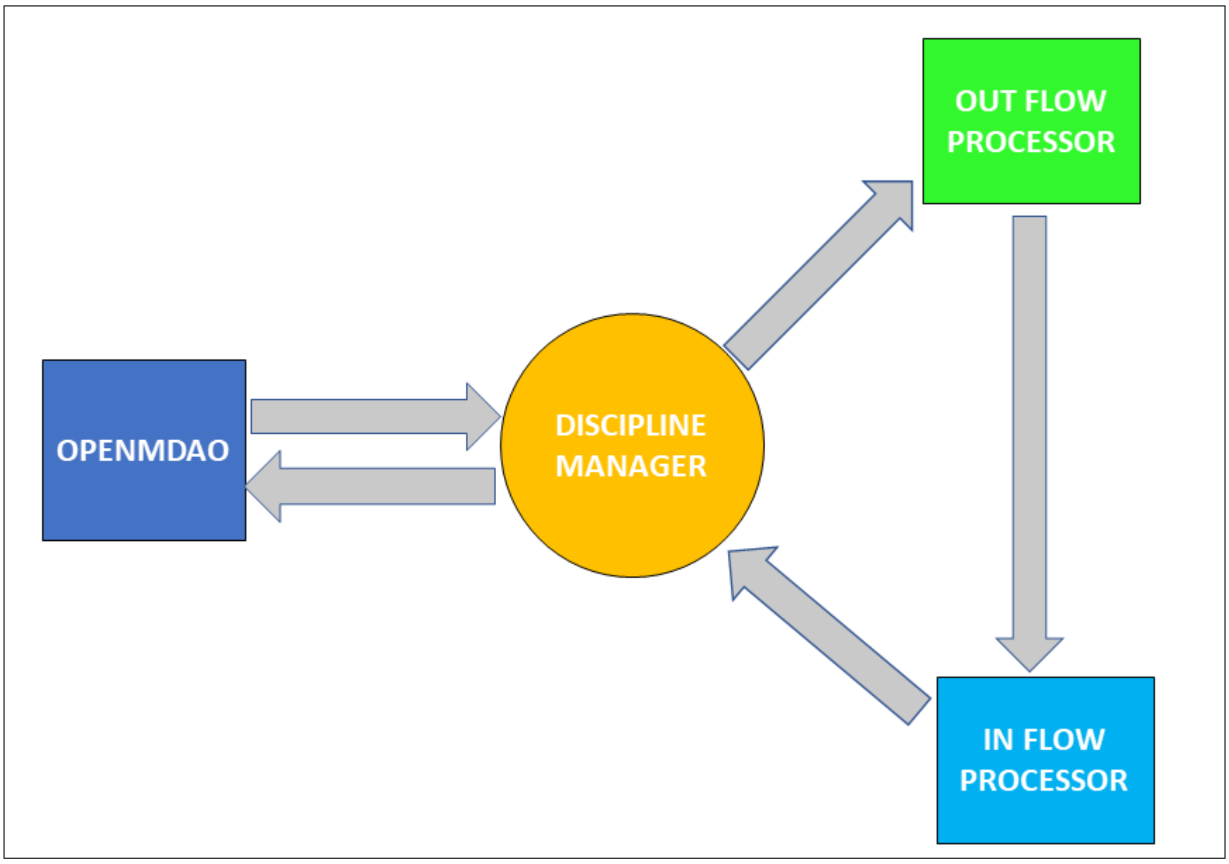
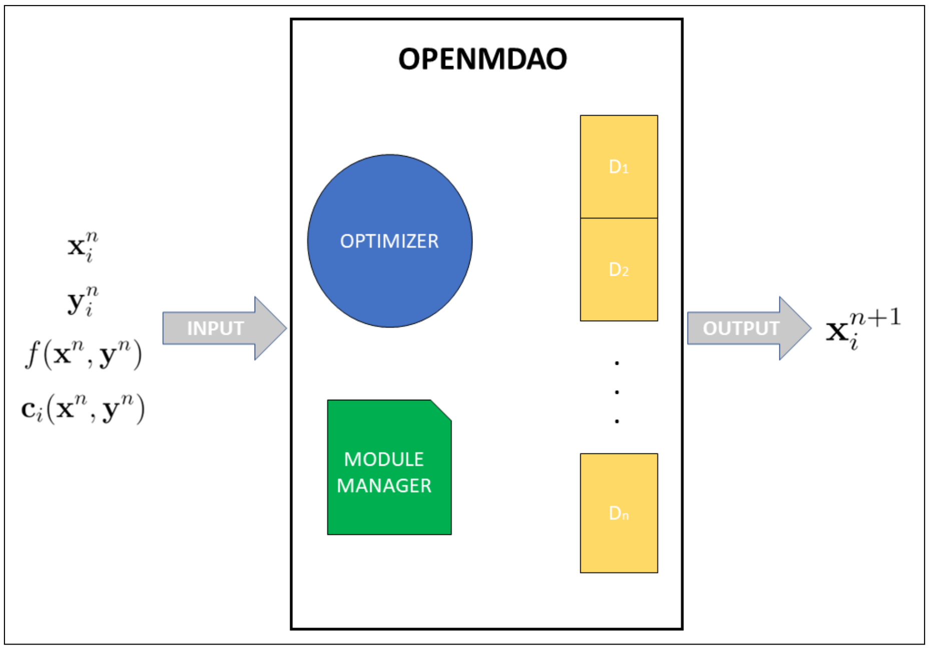


Figure 11: Bird's view of the Framework

OpenMDAO



Input: Design variables, coupling variables, objective functions and constraint equations

Output: The updated value of design variables based on the optimization technique given

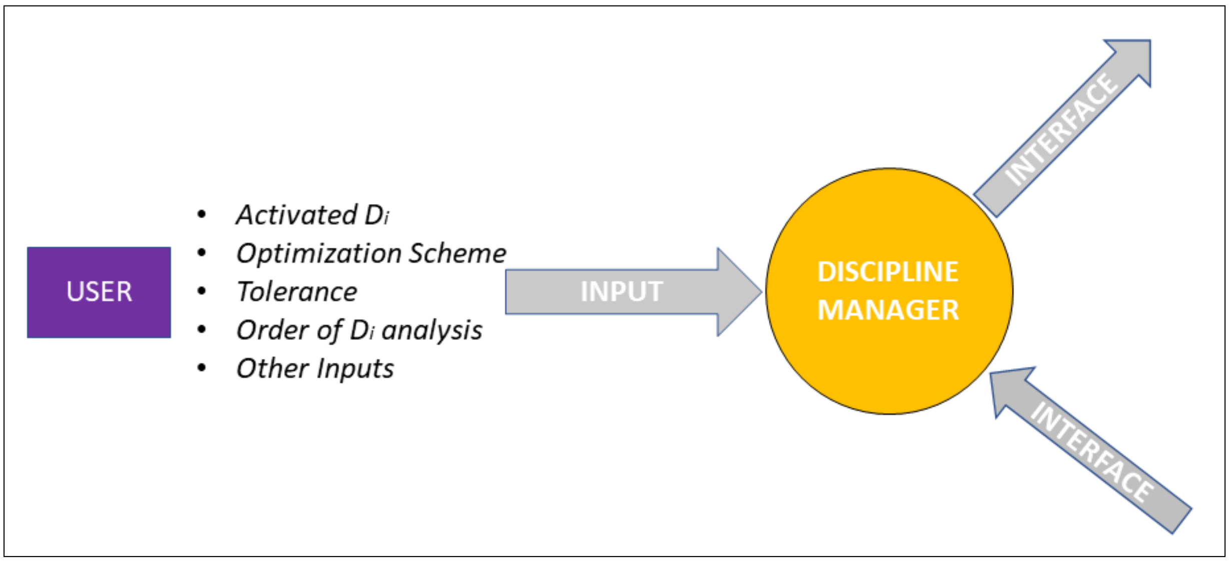
Description: The internal blocks of OpenMDAO are setup in a generic sense, it neither assumes any numerical value nor limits the dimensionality of the problem. It is purely setup in a way similar to symbolic representation.

Module Manager: Receives the input as vectors, the design variables are classified on the basis of their disciplines and sent to the discipline module. Module manager is the primary tool that inputs as well as outputs the necessary information from OpenMDAO to the discipline manager.

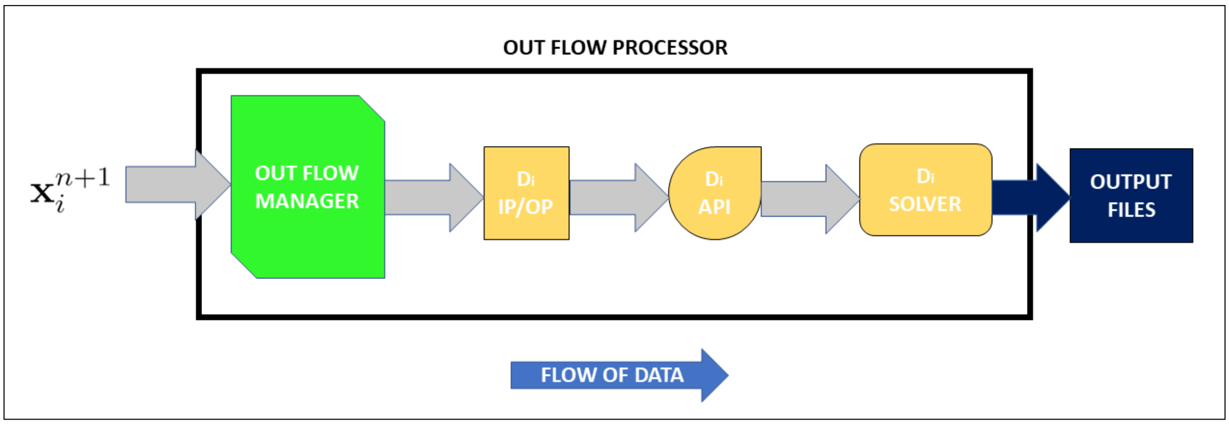
Discipline Module: Each discipline module (within OpenMDAO framework) sets up the objective function and constraint equations. It also contains the information regarding the type of gradients to be used for that particular discipline. Notice that each discipline is setup in a **modular fashion**. If a new discipline is to be added, it can be done so without modifying any of the other disciplines' code structure.

Optimizer: Runs the optimization problem by using the necessary information from the disciplines. Finally outputs the updated value of design variables to the Module Manager.

DISCIPLINE MANAGER
Discipline manager is the main interface that communicates with the user, OpenMDAO, outflow and inflow processors. It takes the essential inputs from the user and sets up the entire optimization loop to be executed. If there are any input error from the user, it raises an error message.



OUTFLOW PROCESSOR



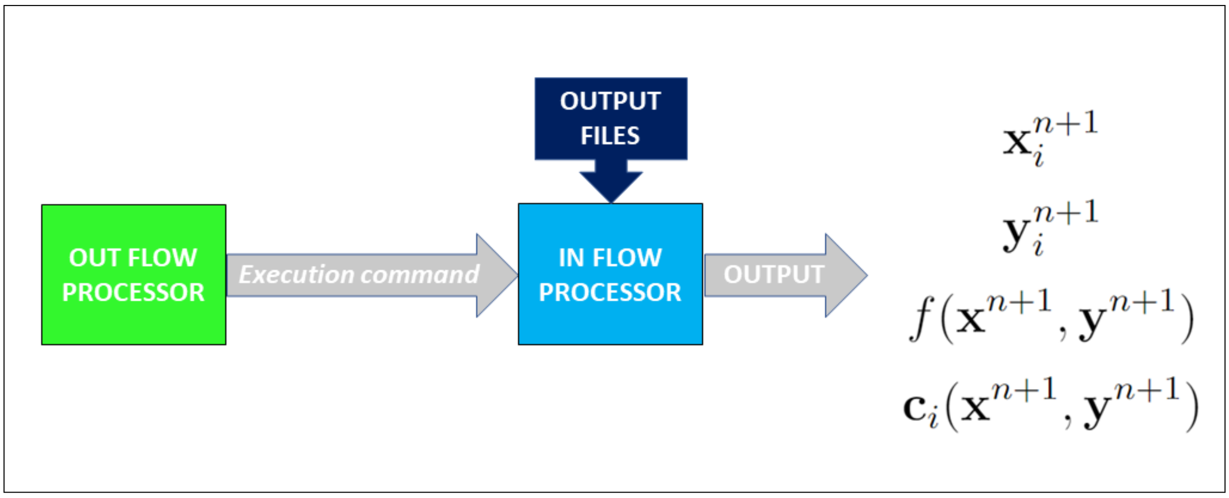
The outflow processor is further categorized into four distinct modules.

Description: The outflow processor takes the updated design variables as inputs, sends them to discipline specific modules to run a new analysis. The final simulation results are in separate output files for each discipline.

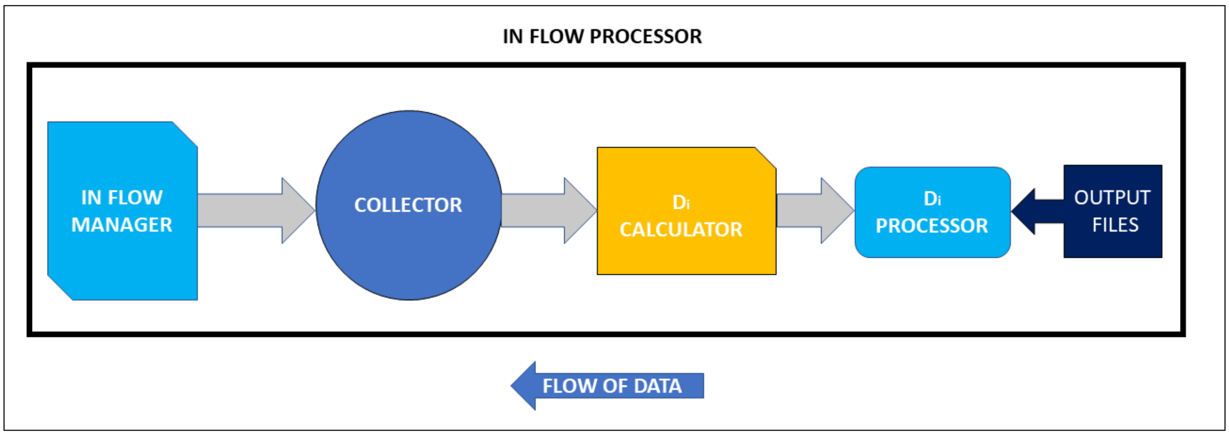
Outflow Manager: Outflow manager is completely code agnostic. It takes the updated design variables and sends it to the respective IP/OP module of each discipline. It also calls the respective analysis functions in a sequential or parallel way as per the instruction from the user.

Input/Output Module: The IP/OP module is available for each discipline. While the input segment is totally solver agnostic, the same cannot be said for the output component. Based on the API of the solver available, the Output segment may be subjected to modification. Everything on the right of the IP/OP module is treated as a black box. The APIs and solver analysis are NOT part of the framework.

INFLOW PROCESSOR



Description: Unlike the outflow processor, all the components in the inflow processor are solver agnostic. Once the output files are generated, an execution command is sent to the inflow processor from the outflow processor. Based on the execution command from outflow manager, the subsequent commands are passed to invoke the other modules by the inflow manager. The inflow processor also has four main modules.



D$_i$ Processor: The discipline processor goes through the output files and searches for the relevant objective function and constraint equations. If already available, the information is directly sent to the collector module. If unavailable, the discipline calculator module is invoked to calculate the required information.

D$_i$ Calculator: In case the required objective functions are not available, this module calculates the desired objective functions and constraints equations directly from the output files and sends them to the Collector module.

Collector Module: This module collects the numerical value of objective functions, constraint equations, design variables and gradients from each discipline and sends it to the inflow manager.

Inflow Manager: The inflow manager directly interacts with the outflow manager, discipline manager and collector. It manages the function calls and sends the final set of data needed for OpenMDAO through the Discipline Manager.

Here a crucial decision is implicitly assumed. Every analysis solver of the same discipline is expected to give the output files in the same format. However, this may not be true. For instance, the format of output data from OpenFOAM may be very different from that of SU2. In such cases another module is desirable to read the specific format of the output from the solver and generate a common format of output that can be interpreted by any simple script (such as .txt format). The reason for not including this module is that many such tools with the capability to automate already exist. Such a module may be a tool like Paraview or Techplot which have wide array of options to interpret any format of solver data. Such tools are also code agnostic in general.

In the next chapter, a basic version of the framework which was developed is tested on an aerodynamics problem.

6 Implementing and Testing the Framework

In this chapter two different analysis tools are used to show that the framework is indeed solver agnostic. The framework is tested for a single discipline: Aerodynamics. Other disciplines shall be included in the framework during the future work.

6.1 Framework for 2-D Aerodynamics

The code structure of the framework based on the philosophy discussed in the previous chapter is detailed below. Note that all the modules are code agnostic except the input/output module, analysis tool's API and the analysis tools themselves.

1. OpenMDAO
    - The aerodynamics module accommodates objective function, constraints and design variables.
    - The module manager tracks the design variables, objective function and constrain equation within OpenMDAO. It also acts as the interface between OpenMDAO and the discipline manager
2. Discipline Manager
    - Interface between the user and the OpenMDAO
    - Design variables, objective functions and constraint equations setup by the user (in a setup file) is read by this module
3. Outflow Manager
    - Communicates the updated design variables to the IP/OP aerodynamics module.
4. Input/Output Module
    - Communicates the received design variables in a format that can be processed by the Xfoil API.
    - This module has function calls that are specific to the analysis tool's API and hence is not solver agnostic
5. Xfoil API
    - Invokes Xfoil whenever required to calculate flow quantities
    - Provides necessary information to setup the analysis tool. For example, information such as angle of attack, reynolds number, number of panels etc.
    - Returns 1 in-case of successful analysis of Xfoil
    - Command returned to Outflow manager which in turn invokes the inflow manager
6. Xfoil
    - Runs the analysis and writes an output file with relevant flow information
7. Inflow Manager

- Collects the necessary information from the output file and communicates it to the discipline manager

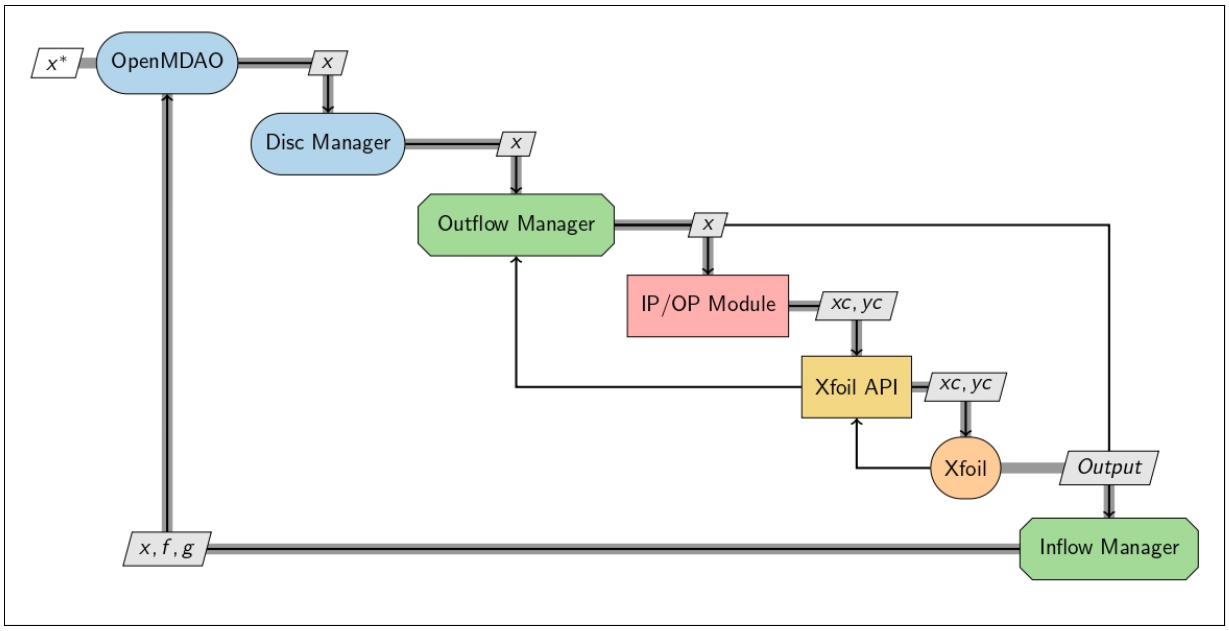


Figure 12: Code Structure and Execution Order with Xfoil

The gray thick lines represents flow of data while the thick blakc lines with arrow head represents the order of code execution.

Changing the analysis tool from Xfoil to any other solver only replaces the Xfoil API. The replaced API communicates with the replaced solver. The previous function calls in the input/output module are also replaced by calls that are suitable to the API. Rest of the program remains untouched. In our case, Xfoil is replaced with a **surrogate model**.

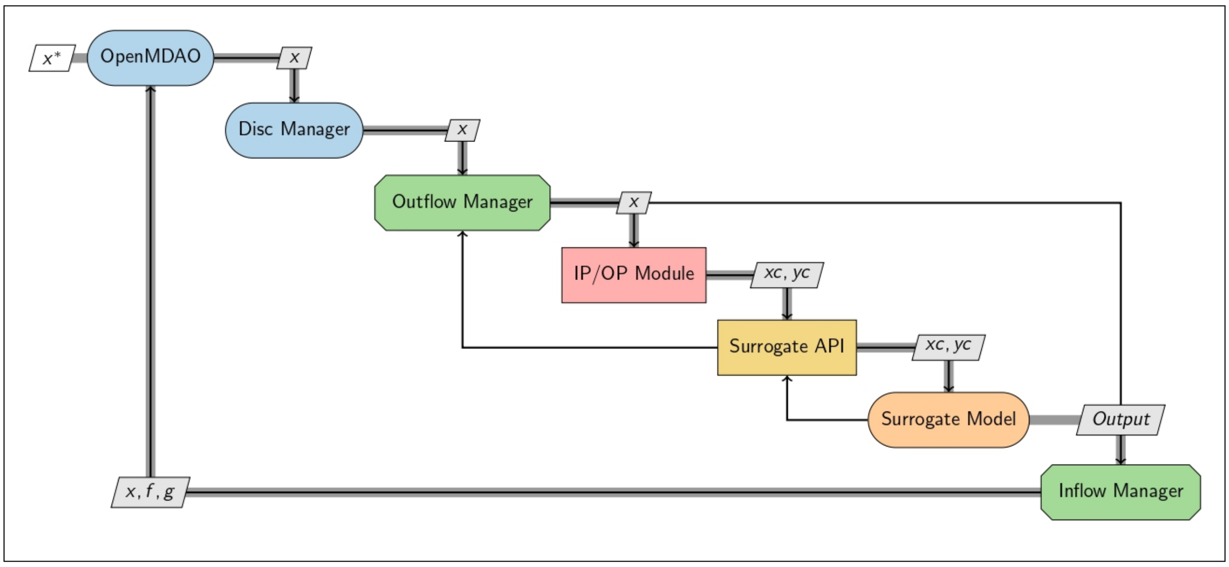


Figure 13: Code Structure and Execution Order with Surrogate Model

6.1.1 Training the Surrogate Model

The methodology of training the surrogate model is described in this section

1. A set of 1000 airfoils is generated first
2. A guess value of 50% is initially assumed to be the portion of training data set. Remaining 50% constitutes the test data set.
3. If the surrogate model can predict the test data set within 10% of accuracy the model is assumed to be completely trained.
4. Else, the guess is increased by 5% and the training is performed until prediction error goes below 10%

In this case, 75% of the training set is used to train the model and the rest 25% is used as the test data set to check the accuracy of the model. The prediction error is about 12% with the test data set. A kriging method is used here with the value of parameter θ as 10^{-2}.

6.2 Preliminary Results

6.2.1 Problem 1: Unconstrained Optimization

$$\begin{array}{rl} minimize & C_l \\ with\ respect\ to & -4\% \leq \frac{\mathbf{x}^u}{c} \leq 4\% \end{array}$$

where c is the chord length and x^u is a vector (length 13) of amplitude for the bump functions, each placed along the upper surface of the NACA 0012 airfoil. The plot of the optimized airfoils is shown below:

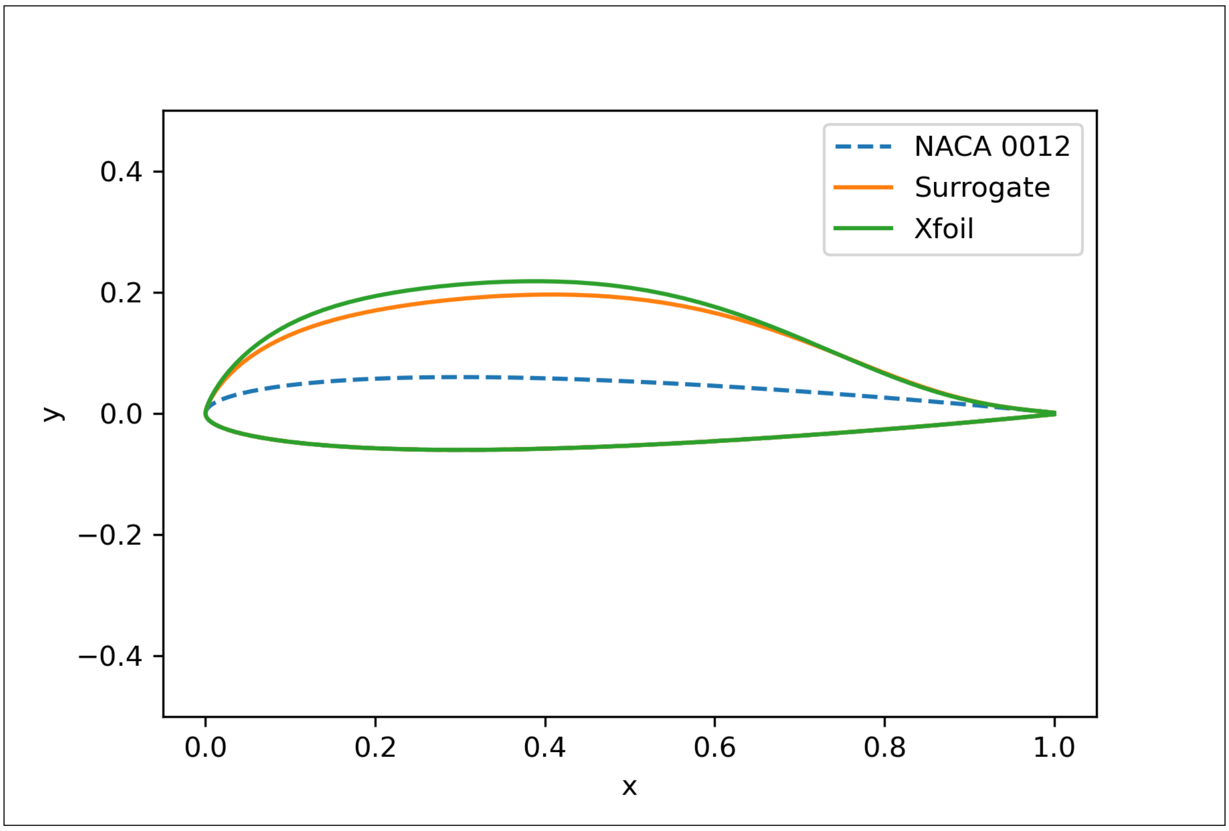


Figure 14: Optimized Airfoil using different analysis tools

Analysis tool	Objective Fn	Optimized Value
Xfoil	C_l	0.666
Surrogate	$C_{l,predicted}$	0.89
Surrogate	$C_{l,exact}$	0.606

Table 1: Lift coefficient magnitude from the analysis tools

6.2.2 Problem 2: Constrained Optimization

$$\begin{aligned} &minimize && C_d \\ &with\ respect\ to && -4\% \leq \frac{\mathbf{x}^u}{c} \leq 4\% \\ & && -1\% \leq \frac{\mathbf{x}^l}{c} \leq 1\% \\ &subject\ to && 0.45 \leq C_l \leq 0.55 \end{aligned}$$

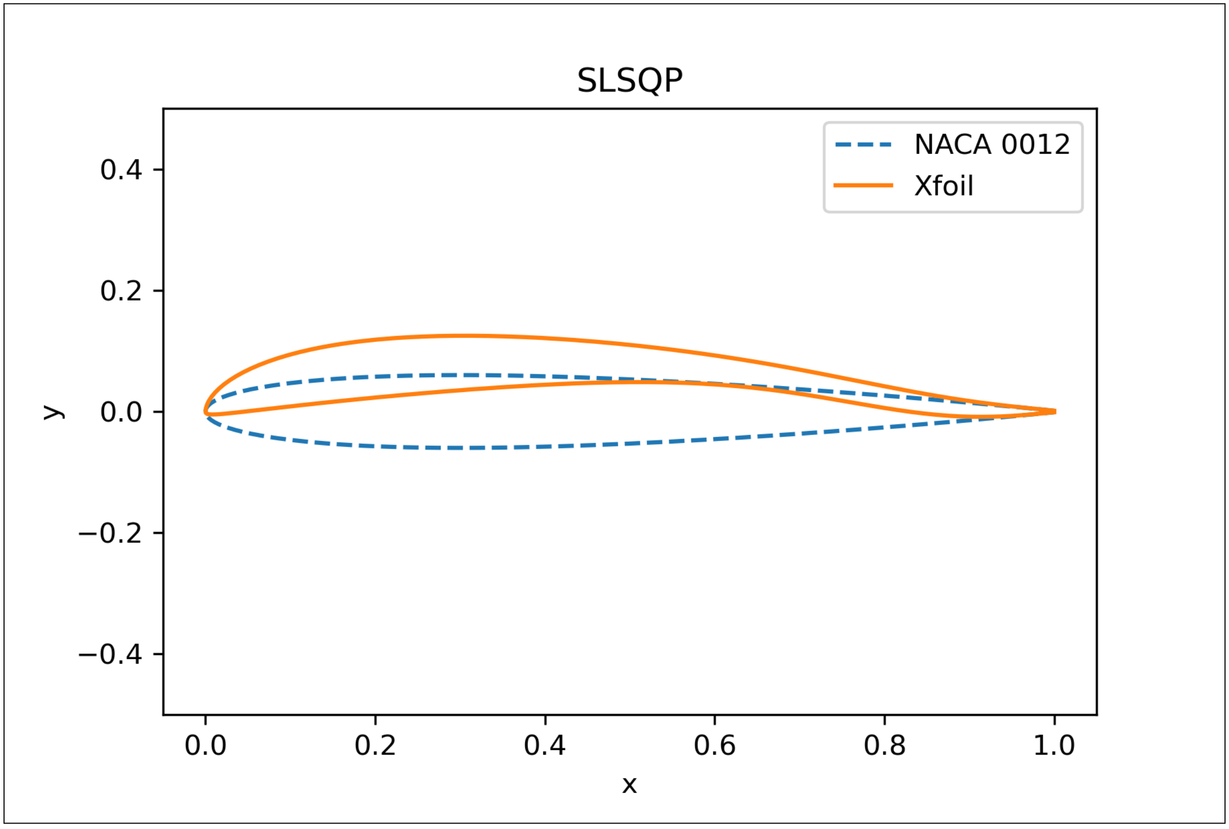


Figure 15: Constrained Optimization

Functions	Optimized Value
C_l^*	0.55814
C_d^*	0.005277

Table 2: Optimized Results for Constrained Optimization

The gradient based solver *SLSQP* Sequential Least SQuares Programming is used. The C_l value predicted by the surrogate model has an error of about 47% from the exact value predicted by Xfoil. The interface works as expected by seamlessly working with both Xfoil and the Surrogate model, treating both the tools as black boxes. Finite Differences was used to evaluate gradients for both the tools.

7 Future Work

The framework has been developed for a 2D aerodynamics problem, without having to use any complicated mesh generation based analysis tool. An immediate add-on is to setup the framework using an analysis tool that uses a simple mesh generation API. Doing so would give more confidence that the framework is indeed capable of communicating seamlessly with multiple components while they themselves are talking to each other. The process to integrate another independent discipline, such as RCS, is under progress and yet to be completed. Some of the key challenges faced while integrating any discipline are:

1. Identifying a suitable API and getting used to it in-order to integrate an analysis tool or solver into OpenMDAO framework in an efficient way
2. Mesh generation need to be automated to the maximum possible extant, in-built mesh deformation tools within an available solver are the easiest to work with. However most of the analysis tools do not have an in-built deformation capability, so alternative tools such as PyHYP or Gmsh which have robust Python APIs may need to be used
3. Identifying the mapping of different meshes and transforming the relevant physical quantities as per the mesh is also yet to be tackled

The following are the goals to be achieved before the next phase

1. Develop the framework to integrate another discipline, preferably RCS module
2. Optimize the 2D airfoil along with RCS and validate the results with existing literature
3. Optimize a 3D wing with only aerodynamics first, to figure out tools that would be needed additionally, such as meshing API and solver API
4. Optimize a 3D wing with aerodynamics and RCS combined and validate the results with existing literature
5. Optimize a 3D wing with atleast two disciplines coupled with each other, for instance aerodynamics and structures, along with another independent discipline such as RCS
6. Add other disciplines one by one on top of the existing framework to deliver the complete framework

The structure of the framework will also be refined further based on the challenges that will be encountered.

References

[1] Justin S Gray, John T Hwang, Joaquim RRA Martins, Kenneth T Moore, and Bret A Naylor. Openmdao: An open-source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 59(4):1075–1104, 2019.

[2] Raymond M Hicks and Preston A Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15(7):407–412, 1978.

[3] John T Hwang and Joaquim RRA Martins. A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives. *ACM Transactions on Mathematical Software (TOMS)*, 44(4):1–39, 2018.

[4] Brenda Kulfan and John Bussoletti. " fundamental" parameteric geometry representations for aircraft component shapes. In *11th AIAA/ISSMO multidisciplinary analysis and optimization conference*, page 6948, 2006.

[5] Andrew B Lambe and Joaquim RRA Martins. Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. *Structural and Multidisciplinary Optimization*, 46(2):273–284, 2012.

[6] Christopher Marriage and Joaquim RRA Martins. Reconfigurable semi-analytic sensitivity methods and mdo architectures within the pimdo framework. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 5956, 2008.

[7] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, Jan 2022.

[8] Joaquim RRA Martins and Andrew B Lambe. Multidisciplinary design optimization: a survey of architectures. *AIAA journal*, 51(9):2049–2075, 2013.

[9] Dominic A Masters, Nigel J Taylor, TCS Rendall, Christian B Allen, and Daniel J Poole. Geometric comparison of aerofoil shape parameterization methods. *AIAA journal*, 55(5):1575–1589, 2017.

[10] 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.

[11] András Sóbester and Alexander IJ Forrester. *Aircraft aerodynamic design: geometry and optimization*. John Wiley & Sons, 2014.