National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Multidisciplinary Analysis and Optimization: Theory, Implementation and Application

Diploma Thesis

Dimitrios Pallas

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

Acknowledgements

First and foremost I extend my deepest gratitude to my supervisor, Professor Kyriakos C. Giannakoglou, for guiding me throughout my Diploma Thesis, from start to finish. I truly appreciate all the helpful insight he provided and all the time he devoted to helping me at every step of the way.

I would like to also thank the members of the PCOpt/NTUA, especially Dr. Varvara Asouti, Dr. Xenofon Trompoukis and Dr. Marina Kontou. They were all very helpful and willing, assisting me a great many times, mainly on technical issues that I faced.

Lastly, I am grateful to my friends, family and parents for believing in me throughout my studies at NTUA. I thank them for their love and support.

Abstract

This diploma thesis is concerned with the theory and application of Multidisciplinary Analysis and Optimization (MDAO) methods, mainly in the area of PDE-constrained optimization. The vast growth of computational power in recent years has made numerical simulations an indispensable tool in engineering, enabling the analysis of ever more complex systems. In the context of engineering, these systems are often multidisciplinary, meaning that they require expertise from different scientific disciplines, which gives rise to the need for MDAO. The mathematical framework behind MDAO allows for efficient coupling of numerical models simulating different physical phenomena, with the aim of both analyzing and optimizing the system being modelled.

In order to manage the complexities involved with MDAO, a software package is developed in Python. The package, named mSense, is used for the applications throughout the thesis. mSense provides tools for both Multidisciplinary Analysis (MDA) and Multidisciplinary Optimization (MDO), allowing the user to easily switch between different methods, selecting the one best-suited to each individual problem. An MDO problem can be formulated and solved in multiple ways, each defining a MDO different architecture. In mSense three different architectures are implemented: Multidisciplinary Feasible (MDF), Individual Discipline Feasible (IDF) and Collaborative Optimization (CO). The package is first used to validate and benchmark the performance of the implemented MDO architectures on standard test problems.

The presented MDAO methodology is applied to two Fluid-Structure Interaction (FSI) problems. The first models an airfoil inside an inviscid flow field. The airfoil is attached to a torsional spring, and is therefore able to rotate. The system is analyzed with an MDA, and the equilibrium is found. Then the shape of the airfoil is optimized, using the MDF and IDF architectures, and comparing the results. The second problem is concerned with the flow of liquid through a vertical elastic tube. As the fluid flows through the tube, it deforms it. The aim of the optimization is to control the deformation of the tube, by manipulating the properties of the tube's elastic material. The MDF architecture is used.

Finally, the problem of aerostructurally optimizing an aircraft wing is solved.

The ONERA M6 wing, which is widely used as a benchmark for Computational Fluid Dynamics (CFD) codes, is considered. Since no structural model exists for the ONERA M6, a simple beam finite-element model for the bending of the wing is developed. The aerostructural model is analysed, computing the deformations and stresses that the wing undergoes during flight, and then optimized using MDF. Two different objective functions are used in the optimization, during which both the shape and structure of the wing are allowed to vary. The results obtained from the two objective functions are compared.

Περίληψη

Το θέμα αυτής της διπλωματικής εργασίας είναι η θεωρία και η εφαρμογή των μεθόδων Πολυτομεακής Ανάλυσης και Βελτιστοποίησης (Multidisciplinary Analysis and Optimization, MDAO). Η ραγδαία αύξηση της υπολογιστικής ισχύος τα τελευταία χρόνια εχει καταστήσει τις αριθμητικές προσομοιώσεις ένα πολύτιμο εργαλείο στον τομέα της μηχανολογίας, καθώς επιτρέπει την ανάλυση ολοένα και πολυπλοκότερων συστημάτων. Τα συστήματα αυτά είναι συχνά πολυεπιστημονικά-πολυτομεακά, απαιτώντας τεχνογνωσία από διαφορετικά πεδία, το οποίο δημιουργεί την ανάγκη για μεθόδους MDAO. Το μαθηματικό υπόβαθρο πίσω από το MDAO επιτρέπει την αποτελεσματική σύζευξη αριθμητικών μοντέλων, καθένα από τα οποία προσομοιώνει ένα διαφορετικό φυσικό φαινόμενο, με σκοπό την ανάλυση και βελτιστοποίηση του υπό εξέταση συστήματος.

Για τη διαχείριση της πολυπλοκότητας που σχετίζεται με το MDAO αναπτύσσεται ένα πακέτο λογισμικού στη γλώσσα προγραμματισμού Python. Το πακέτο, ονόματι mSense, χρησιμοποιείται για τις εφαρμογές της εργασίας. Περιλαμβάνει εργαλεία τόσο για πολυτομεακή ανάλυση (Multidisciplinary Analysis, MDA), όσο και βελτιστοποίηση (Multidiscplinary Optimization, MDO), επιτρέποντας στο χρήστη να επιλέξει τη μέθοδο που εφαρμόζεται καλύτερα στο εκάστοτε πρόβλημα. Ένα πρόβλημα MDO μπορεί να επιλυθεί με πολλαπλές προσεγγίσεις, και κάθε μία αποτελεί μία διαφορετική αρχιτεκτονική MDO. Στο mSense υλοποιούνται τρεις διαφορετικές αρχιτεκτονικές, που είναι: η Multidisciplinary Feasible (MDF), η Individual Discipline Feasible (IDF) και η Collaborative Optimization (CO). Το πακέτο χρησιμοποιείται αρχικά για να επικυρώσει την υλοποίηση της κάθε αρχιτεκτονικής και να συγκρίνει την επίδοσή τους, σε δύο πρότυπα προβλήματα MDO.

Η μεθοδολογία MDAO εφαρμόζεται έπειτα σε δύο προβλήματα αλληλεπίδρασης ρευστού-στερεού (Fluid-Structure Interaction, FSI). Το πρώτο πρόβλημα μοντελοποιεί την συμπεριφορά μίας αεροτομής εντός ατριβούς πεδίου ροής. Η αεροτομή είναι προσδεμένη σε ένα στρεπτικό ελατήριο, γύρω από το οποίο μπορεί να στραφεί. Το σημείο ισορροπίας του συστήματος αεροτομή-ελατήριο υπολογίζεται μέσω πολυτομεακής ανάλυσης (MDA). Στη συνέχεια, το σχήμα της αεροτομής βελτιστοποιείται με χρήση των αρχιτεκτονικών MDF και IDF, και τα αποτελέσματα συγκρίνονται. Αντικείμενο του δεύτερου προβλήματος είναι η ροή ρευστού σε κάθετο ελαστικό σω-

λήνα. Σκοπός της βελτιστοποίησης είναι ο έλεγχος της παραμόρφωσης του σωλήνα μέσω προσαρμογής των υλικών ιδιοτήτων του υλικού του σωλήνα. Χρησιμοποιείται η αρχιτεκτονική MDF.

Το τελευταίο πρόβλημα που επιλύεται είναι η αεροδομική βελτιστοποίηση πτέρυγας αεροσκάφους. Επιλέγεται η πτέρυγα ONERA M6, η οποία χρησιμοποιείται συχνά για την επικύρωση κωδίκων υπολογιστικής ρευστοδυναμικής. Ωστόσο, για την πτέρυγα αυτή δεν υπάρχει έτοιμο δομικό μοντέλο, οπότε αναπτύσσεται ένα απλό μοντέλο πεπερασμένων στοιχείων με δοκούς, το οποίο μοντελοποιεί την κάμψη της πτέρυγας. Μέσω πολυτομεακής ανάλυσης (MDA), επιλύεται το αεροδομικό μοντέλο και υπολογίζονται οι παραμορφώσεις και τάσεις που αναπτύσσονται στην πτέρυγα κατά την πτήση. Ακολουθεί δομική βελτιστοποίηση και βελτιστοποίηση μορφής, με σκοπό την βελτίωση ορισμένων χαρακτηριστικών της πτερύγας. Δοκιμάζονται δύο διαφορετικές συναρτήσεις στόχου και συγκρίνονται.

Nomenclature

CFD Computational Fluid Dynamics

CO Collaborative Optimization

DoF Degree of Freedom

FEM Finite Element Method

FSI Fluid-Structure Interaction

GPU Graphics Processing Unit

IDF Individual Discipline Feasible

MDA Multidisciplinary Analysis

$MDAO$ Multidisciplinary Analysis and Optimization

MDF Multidisciplinary Feasible

MDO Multidisciplinary Optimization

MPI Message Passing Interface

$NTUA$ National Technical University of Athens

$NURBS$ Non-Uniform Rational B-Splines

$PCOpt$ Parallel CFD and Optimization unit

PDE Partial Differential Equation

$PUMA$ Parallel Unstructured Multirow and Adjoint

$w.r.t$ with respect to

$XDSM$ Extended Design Structure Matrix

Contents

Chapter 1

Introduction

1.1 Brief introduction to MDAO

Modern engineering systems are most often multidisciplinary. They are comprised of multiple physical components which are analyzed and designed with regard to various physical phenomena. The design process employed currently in the majority of engineering applications is sequential. The engineers working on a project are often grouped by either discipline or physical subsystem, and information is usually passed between teams in a predetermined, one-way manner. For example, in the design of an aircraft, the aerodynamicists optimize the shape of the aircraft with the goal of minimizing its drag, subject to other aerodynamic constraints. The optimized geometry and the aerodynamic loads are passed to the structural engineers, who must then design the internal structure. Finally, the control engineers tune the aircraft's controls systems according to the aerodynamic and structural characteristics provided. This approach essentially disregards the interactions between the disciplines or components, and is therefore unable to exploit them, likely leading to sub-optimal designs. Multidisciplinary Analysis and Optimization (MDAO) aims to offer a standardized mathematical framework for the efficient design of such systems.

The history of MDAO is rooted in aeronautics. In his 1974 paper, Haftka [13] optimized the structural design of an aircraft wing using a finite element model and simplified aerodynamics. A decade later, a symposium was held at NASA's Langley Research Center where Sobieski [33], among others, discussed the use of MDA and MDO for the design of aeronautical, naval and other systems. Over the last twenty years, the evolution of both hardware and software has enabled the analysis and optimization of ever more complex systems. Instead of designing single components, it is now possible to consider entire systems at once [18, 25, 37].

1.2 MDAO terminology and mathematical description

An engineering system is modelled numerically through sets of equations which are solved to predict the internal state of the system. In the general case, these equations are non-linear and implicit in the internal state variables. The equations of a (single-discipline) model with n states can be represented in the following residual form:

$$r_i(y_1, y_2, \ldots, y_n) = 0,\ i \in (1, n) \tag{1.1}$$

In the above expression, r_i is the residual of the i-th equation and y_i the i-th state variable. Both the residuals and the states can be compactly written as vectors of size n, namely $R = \begin{bmatrix} r_1 & r_2 & \ldots & r_n \end{bmatrix}^\top$ and $Y = \begin{bmatrix} y_1 & y_2 & \ldots & y_n \end{bmatrix}^\top$. The equations are then succinctly written as:

$$R(Y) = 0 \tag{1.2}$$

A numerical model of a multidisciplinary system consists of multiple sub-models, one for each discipline. Each sub-model has its own set of equations and state variables. Consider a multidisciplinary model with m disciplines. For the i-th discipline, its residual and state vectors are R_i and Y_i. However, it is no longer correct to just write $R_i(Y_i) = 0$, as discipline i may depend on the state of some other discipline Y_j. The full set of equations describing the multidisciplinary model should be written as:

$$R_i(Y_1, Y_2, \ldots, Y_m) = 0,\ i \in (1, m) \tag{1.3}$$

The above representation of the multidisciplinary model will be referred to as the residual form. For eq. 1.3 to be solved, the residual and state vectors of all its disciplines are concatenated, resulting into a single large set of equations. In order to better understand the residual form, let us consider the aerostructural model of an aircraft wing. This system is comprised of two disciplines ($m = 2$), aerodynamics and structures. The aerodynamics discipline is, in essence, represented by the CFD solver which simulates the airflow around the wing. For compressible flow the solver might solve for the three velocity components ($V = v_x, v_y, v_z$), the density (ρ) and the energy (E) at each node of the CFD mesh. For the structures discipline, the FEM solver might solve for the displacement components ($U = u_x, u_y, u_z$) at each node of the structural mesh. The residual and state variables vectors for each discipline are R_{CFD} and Y_{CFD}, and R_{FEM} and Y_{FEM} respectively. Therefore, the set of equations is:

$$\begin{aligned} R_{CFD}(Y_{CFD}, Y_{FEM}) &= 0 \\ R_{FEM}(Y_{CFD}, Y_{FEM}) &= 0 \end{aligned} \tag{1.4}$$

Formulating a multidisciplinary model in the residual form is not always desirable, nor is it always feasible. There often exist efficient and specialized solvers for each discipline, which rarely give access to their internals. Even if all disciplinary solvers are able to compute and export their residuals, the solution of the resulting concatenated system is likely not as efficient, as if each solver solves its respective disciplinary equations.

A more modular form of multidisciplinary models exists, which considers not the states, but only the inputs and outputs of each discipline. Consider again a model of m disciplines. The outputs of discipline i are denoted by $\hat{Y}_i$ and are either a subset of its internal variables Y_i, or directly derived from them. The inputs of discipline i are the outputs of the other disciplines i.e. $\hat{Y}_{j\neq i} = \begin{bmatrix}\hat{Y}_1 & \dots & \hat{Y}_{i-1} & \hat{Y}_{i+1} & \dots & \hat{Y}_m\end{bmatrix}^\top$[1]. The variables $\hat{Y}_i$ are responsible for the coupling between the disciplines, and therefore named the coupling variables. The set of equations which describe the multidisciplinary model is now expressed as follows:

$$\hat{Y}_i = \hat{Y}_i(\hat{Y}_{j\neq i}),\ i \in (1, m) \tag{1.5}$$

This representation is referred to as the functional form of the multidisciplinary model. Returning to the aerostructural wing model example, the CFD solver now only exports variables of interest, like the pressure P and shear stress τ values, at all points on the wing surface. Similarly, the FEM solver exports the displacement values U at the wing surface. The output variables of each discipline are $\hat{Y}_{CFD}$ and $\hat{Y}_{FEM}$ respectively. The set of equations for the aerostructural wing model in the functional form are:

$$\begin{aligned} \hat{Y}_{CFD} &= \hat{Y}_{CFD}(\hat{Y}_{FEM}) \\ \hat{Y}_{FEM} &= \hat{Y}_{FEM}(\hat{Y}_{CFD}) \end{aligned} \tag{1.6}$$

Here, each disciplinary solver actually solves for its own state variables (unlike in the residual form, where it just computes and exports its residuals), and then exports the necessary outputs, without the need to expose its internals. This also means, that the functional form equations can be solved in a decoupled fashion, as each disciplinary analysis can be executed independently, in contrast to the residual form, where the large set of concatenated equations is solved at once. Essentially, the residual and functional forms differ in the set of variables they handle. The

[1]It is not necessary that discipline i has as inputs the outputs of all other disciplines. The input vector $\hat{Y}_{j\neq i}$ is written that way for the sake of generality.

functional representation handles a much smaller number of variables. This is because the size of the discipline output, denoted by $n_{\hat{Y}_i}$, is typically much smaller than the size of its state vector n_{Y_i}. For the aerostructural wing model example, the number of nodes on the wing surface, and therefore the number of variables, is typically much smaller than the number of nodes in the entire domain. Finally, it is noted again, that not all models can be written in residual form. This is true only if each disciplinary solver involved provides access to its internals, and is hence able to export its residuals. The two forms for multidisciplinary models, i.e. residual and functional, are presented in greater detail in [26, Chapter 13].

A Multidisciplinary Analysis (MDA) is the process of simultaneously satisfying all disciplinary equations, which requires the solution of either eq. 1.3 for all Y_i, or eq. 1.5 for all $\hat{Y}_i$. Multidisciplinary Optimization (MDO) is the process of optimizing a multidisciplinary system, while ensuring multidisciplinary feasibility, namely ensuring that all discipline states are compatible or that eqs. 1.3 and 1.5 are satisfied. Simply put, this means that MDO respects and takes into account the interactions between the disciplines.

An MDO problem can be formulated and solved in various different ways, which are termed formulations or architectures. MDO architectures are broadly placed into two categories, **monolithic** and **distributed**, based on whether they solve one or more optimization problems. Monolithic architectures formulate and solve a single optimization problem, whereas most distributed architectures solve an optimization problem for each discipline, and a system-level coordinating problem. For this reason, most distributed architectures are also called multi-level. Popular monolithic architectures are the **Multidisciplinary Feasible (MDF)** and **Individual Discipline Feasible (IDF)** [24] and **Simultaneous Analysis and Optimization (SAND)**[2] [14], while **Bi-Level Integrated System Synthesis (BLISS)** [35] and **Collaborative Optimization (CO)** [5] are representative examples of distributed architectures. Comparisons indicate that for most problems monolithic architectures tend to perform better, generally requiring fewer disciplinary evaluations and having more robust convergence [36, 12].

Distributed architectures often make a distinction between local and shared or global design variables. A design variable is **local** for a discipline if it directly enters this discipline only. The vector of design variables local to discipline i is denoted by X_i. If a design variable directly enters more than one disciplines (even if not all of them), then it is considered **shared**. The vector of shared design variables is denoted by Z. For a problem with m disciplines, the vector of all local and shared design variables is denoted by X and obviously $X = \begin{bmatrix} Z & X_1 & X_2 & \dots & X_m \end{bmatrix}^\top$. A similar distinction is made for constraints. A constraint evaluated by only a discipline's state Y_i or output variables $\hat{Y}_i$, local design variables X_i and shared design

[2] Also referred to as All-At-Once or AAO

variables Z, is considered local to discipline i. Else, namely if the constraint's evaluation requires the state/output or local design variables of more than one disciplines, it is then shared. The vector of all local and shared inequality constraints is denoted by G and obviously $G = \begin{bmatrix} G_0 & G_1 & G_2 & \dots & G_m \end{bmatrix}^\top$. The vector of all equality constraints is defined as $H = \begin{bmatrix} H_0 & H_1 & H_2 & \dots & H_m \end{bmatrix}^\top$. Finally, if f is the (scalar) objective function of the MDO problem, then $F = \begin{bmatrix} f & G & H \end{bmatrix}^\top$ is the concatenated vector of the objective and all constraints. This notation is used throughout this thesis. All symbols commonly appearing are found in table 1.1.

Most MDO architectures achieve multidisciplinary feasibility in one of two ways. The first is to use an MDA somewhere inside the MDO process (MDF, BLISS). The second method makes use of target variables (IDF, CO). A **target variable** is a copy of (usually) a coupling variable, which is entirely controlled by the optimizer. For example, for the coupling variable $\hat{Y}_i$, its corresponding target variable is $\hat{Y}_i^t$. Now, the inputs of a discipline i are no longer $\hat{Y}_{j\neq i} = \begin{bmatrix} \hat{Y}_1 & \dots & \hat{Y}_{i-1} & \hat{Y}_{i+1} & \dots & \hat{Y}_m \end{bmatrix}^\top$, but $\hat{Y}_{j\neq i} = \begin{bmatrix} \hat{Y}_1^t & \dots & \hat{Y}_{i-1}^t & \hat{Y}_{i+1}^t & \dots & \hat{Y}_m^t \end{bmatrix}^\top$. This enables all disciplinary analyses to be evaluated completely independently of each other, since the output of a disciplinary evaluation is no longer an input for any other discipline. In order to ensure multidisciplinary feasibility suitable constraints, called feasibility or consistency constraints, are applied. The **feasibility constraint** corresponding to target variable $\hat{Y}_i^t$, denoted by $H_{Y_i}^t$, is defined as follows:

$$H_{Y_i}^t = \hat{Y}_i^t - \hat{Y}_i = 0 \tag{1.7}$$

The optimizer uses the constraint $H_{Y_i}^t$ to drive the value of $\hat{Y}_i^t$ to be equal to the value of $\hat{Y}_i$.

An MDO problem, regardless of the architecture used, is most often solved with gradient-based optimizers. This is because of the large computational cost of an MDO solution, which arises from the disciplinary interactions and the need to resolve them. Gradient-based optimization algorithms are typically more efficient than their gradient-free counterparts, but require the computation of the derivatives of the objective and constraints F, with respect to the design variables X, namely $\frac{dF}{dX}$. For MDO architectures which use an MDA to achieve multidisciplinary feasibility at each cycle (for example MDF), the computation of $\frac{dF}{dX}$ must also take the interaction between the disciplines into account. For this reason, $\frac{dF}{dX}$ is commonly referred to as the **(total) coupled derivatives** in the context of MDAO. For efficient computation of the coupled derivatives, multidisciplinary analogs of the direct differentiation and adjoint method [10] exist, named the coupled direct and adjoint methods, first presented in [34].

Symbol	Description
m	Number of disciplines
Y_i	State vector of the i-th discipline
$\hat{Y}_i$	Output vector of the i-th discipline
R_i	Residual vector of the i-th discipline
$\hat{R}_i$	Functional form residual vector of the i-th discipline, $\hat{R}_i = \hat{Y}_i - \hat{Y}_i(\hat{Y}_{j \neq i})$
$n_{()}$	Size of a vector, for example n_{Y_i} is the size of Y_i
X_i	Design variables local to the i-th discipline
Z	Shared or global design variables
X	Vector of all design variables, $X = \begin{bmatrix} Z & X_1 & X_2 & \dots & X_m \end{bmatrix}^\top$
G_i	Vector of inequality constraints local to the i-th discipline
G_0	Vector of shared or global inequality constraints
G	Vector of all inequality constraints, $G = \begin{bmatrix} G_0 & G_1 & G_2 & \dots & G_m \end{bmatrix}^\top$
H_i	Vector of equality constraints local to the i-th discipline
H_0	Vector of shared or global equality constraints
H	Vector of all equality constraints, $H = \begin{bmatrix} H_0 & H_1 & H_2 & \dots & H_m \end{bmatrix}^\top$
f	Objective function (scalar)
F	Vector of objective function and constraints
$()^*$	Variable value corresponding to the optimal
$()^t$	Target variable, for example $\hat{Y}_i^t$ is a copy of $\hat{Y}_i$
$H^t_{()}$	Feasibility constraint, for example $H^t_{Y_i} = \hat{Y}_i^t - \hat{Y}_i$ corresponds to $\hat{Y}_i^t$

Table 1.1: Commonly used MDAO symbols and their description. All capitalized symbols used throughout this thesis are vectors, unless indicated otherwise.

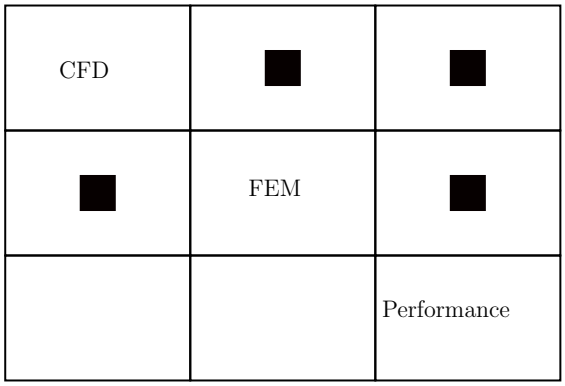


Figure 1.1: Design Structure Matrix (DSM) for the three-discipline, aerostructural wing model. Aerodynamics (CFD) and structures (FEM) are strongly coupled, since they communicate data to each other. Performance only receives data from the other two disciplines.

1.2.1 Graphical representation of multidisciplinary models with the (X)DSM

Although the mathematical definition of a multidisciplinary model should be an adequate description of its structure, the complexity of such models makes it often hard to interpret the underlying data-flow quickly and effectively. The complexity is further increased when considering an MDAO process, for example an MDO architecture applied to a specific model. For this purpose, the eXtended Design Structure Matrix, or XDSM, was developed [22]. As the name suggests, the XDSM is an extension to the simpler Design Structure Matrix (DSM), also called the N^2 chart or interaction matrix. For model with m disciplines the DSM is represented by a $m \times m$ grid. The diagonal entries are the disciplines and are marked by their name. The off-diagonal terms describe the interdisciplinary interactions. For a row i, the off-diagonal terms are the outputs of discipline i. If the j-th entry of row i is marked by a black square, then discipline's i output directly enters discipline j. In order to make the DSM's usage clear, consider again the previously presented aerostructural wing model. A third discipline is added, called performance. This discipline is not a PDE solver like aerodynamics (CFD) and structures (FEM). Instead it is a simple function which calculates some output with information from aerodynamics and structures. Further details about performance are not yet necessary. The DSM for the three-discipline, aerostructural wing example is shown in fig. 1.1. Since $m = 3$, the DSM is 3×3 matrix.

An apparent drawback of the DSM is that does not expose clearly what information is passed from one discipline to another. This is undesired, especially in large applications, where there exist many and complex interactions. The XDSM

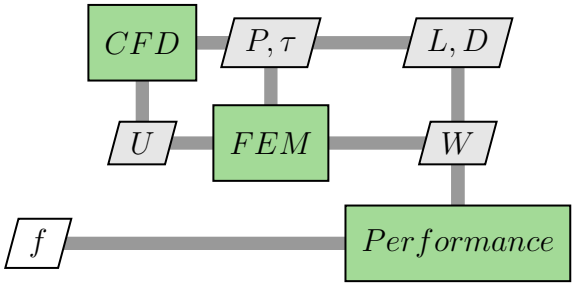


Figure 1.2: Extended Design Structure Matrix (XDSM) for the three-discipline, aerostructural wing example. Aerodynamics (CFD) and structures (FEM) are strongly coupled, since they communicate data to each other. Performance only receives data and computes a performance metric f.

tackles this problem by replacing the off-diagonal terms with line connections, which explicitly state the communicated variables. In the aerostructural wing model example, the outputs of aerodynamics might be the pressure P and shear stress τ at the wing surface, and the lift L and drag D of the wing. Similarly, the outputs of structures might be the displacement U of the wing surface and the weight of the wing W. Finally, performance might calculate some performance metric f with information from the other two disciplines. This is showcased in the corresponding XDSM in fig. 1.2.

Furthermore, the XDSM can be used to visualize MDAO processes such as an MDA, or an MDO architecture applied to a specific problem. This is done by placing not only the disciplines on the diagonal, but also MDA blocks and/or optimizers. Consider that the MDF architecture, although not thoroughly presented yet, is applied to the aerostructural wing example (fig. 1.3). The goal is to optimize the performance metric f w.r.t X_{CFD} and X_{FEM}, which are the design variables of aerodynamics and structures respectively. Inputs to the XDSM are usually placed on top of the diagram, such as the initial values of the design variables X^0_{CFD} and X^0_{FEM}, while outputs are placed typically on the left, such as the optimized values of the design variables X^*_{CFD} and X^*_{FEM}. At each optimization cycle, MDF uses an MDA to enforce multidisciplinary feasibility. This is shown in the XDSM by the diagonal block named "MDA". A second block, named "Optimizer", uses the value of the computed performance metric f, to update the values of the design variables. Inside this XDSM, there exist two closed-loop processes, one corresponding to the MDA and the other to the optimizer. This is indicated by a continuous black arrow for each process, which must begin and end at the same diagonal block. For the MDA process for example, the arrow begins at the "MDA" block, connects the "CFD" and "FEM" blocks, and returns to "MDA".

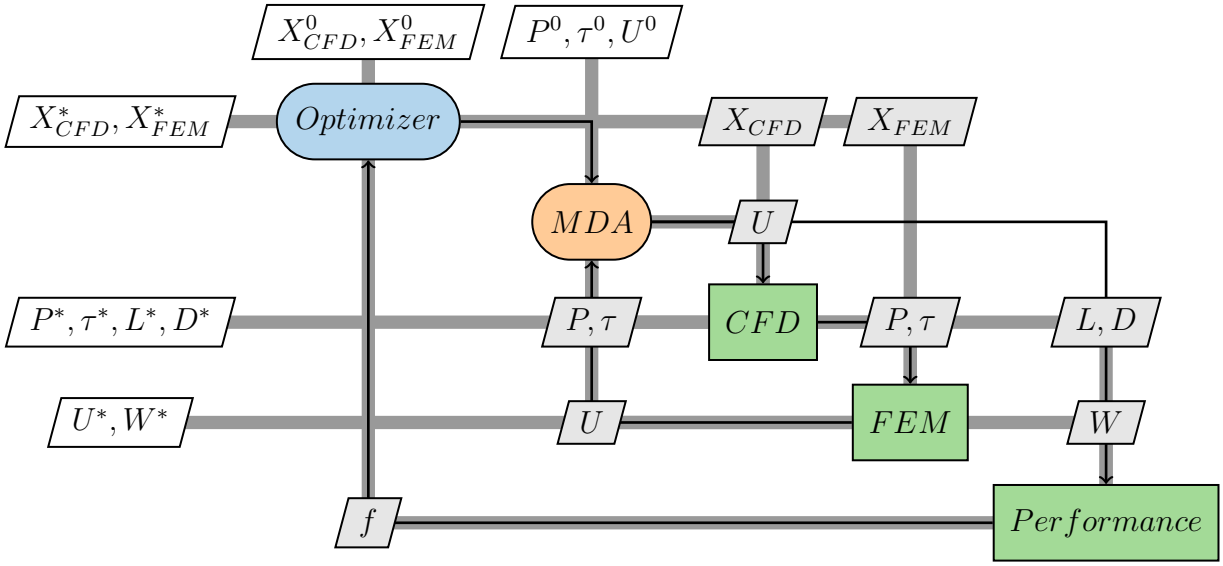


Figure 1.3: XDSM for the MDF architecture applied to the example aerostructural wing problem.

1.3 The mSense package for MDAO

A major part of this thesis is the implementation of a software platform for the rapid development and testing of MDAO methods. Several MDAO frameworks already exist, with OpenMDAO [11] and GEMSEO [9] being notable open-source examples. There also exist commercial packages, such as Isight from Dassault Systèmes [19], ModelCenter from Phoenix Integration, Optimus from Noesis Solutions and VisualDOC from Vanderplaats Research and Development [4], to mention just a few of them. Commercial packages provide easy component integration, having easy to use graphical interfaces and wrappers for engineering analysis programs. However, they lack the modularity of the aforementioned open-source codes and do not have the coupled derivative computation capabilities (coupled direct and adjoint methods), often relying on finite-difference approximations.

MSense (Multidisciplinary + Sensitivity) is an open-source Python package developed to facilitate the setup and solution of MDAO problems. It aims to provide a platform that is fairly easy to use, but can also be extended to suit the specific needs of each user. The MDA module of mSense can use either fixed-point or Newton-based methods to couple disciplines. MSense leverages the coupled direct

and adjoint methods, as well as finite-difference and complex-step [3] approximations for the computation of the coupled derivatives of multidisciplinary models. Derivative approximation can be used either at the discipline, or at the system level, which makes the process of computing sensitivities quite flexible. For example, in a two-discipline system, one discipline might provide its own partial derivatives, while the other might not. MSense is able to approximate the partial derivatives of the second discipline, and then use the coupled direct or adjoint methods to compute the total coupled derivatives. If that is not desirable, mSense is also able to compute the coupled derivatives directly, through approximation at the system/MDA level. The following three architectures are currently implemented in mSense:

- MDF
- IDF
- CO

They can be used for any MDO problem the user sets up. MSense is currently able to utilize the optimization solvers available in Scipy [38], such as SLSQP [20] and COBYLA [30], and the interior point optimizer Ipopt [39]. Finally, mSense has the capability to record the execution of each discipline, in order to avoid evaluating or differentiating a computationally expensive discipline multiple times with the same input values. It should be mentioned that the code only supports multidisciplinary models expressed in the functional form. All applications presented in this thesis are implemented in mSense. The source code, as well as examples, can be found at: https://github.com/dlmpal/mSense. A basic user guide is included in the appendix.

1.4 Thesis outline

The thesis is organized as follows:

- **Chapter 2** A thorough presentation of the theory behind MDAO. The difference between single-discipline and multidisciplinary models is explained. Methods for the solution of the coupled disciplinary equations are presented. The direct and adjoint methods for the computation of the coupled derivatives are derived. Several monolithic and distributed architectures are discussed.

[3] The underlying disciplinary code(s) must be able to use complex number arithmetic.

- **Chapter 3** The performance of three MDO architectures is tested. Two standard MDO benchmarks problems are used. The first is Sellar's problem, which a simple analytic problem of two disciplines. The second is Martins' scalable problem. As the name implies, this problem's dimensionality can be selected arbitrarily.

- **Chapter 4** A Fluid-Structure Interaction (FSI) problem is considered. The problem consists of an airfoil inside a two-dimensional flow field. The airfoil is able to rotate about an axis normal to the flow plane, and attached to the axis is a spring. The equilibrium point of the airfoil-spring system is found through an MDA. Then the shape of the airfoil is optimized using MDO, with the goal of achieving a desired lift value. The MDF and IDF architectures are compared.

- **Chapter 5** A second FSI problem is considered. Fluid flows through an elastic tube, deforming it. The solution to the problem is found through an MDA. Then, the material properties of the tube are optimized, using the MDF architecture, in order to control the maximum displacement of the tube.

- **Chapter 6** A typical MDAO problem, the aerostructural analysis and optimization of an aircraft wing is solved. The wing used is the ONERA M6. Since no structural model exists for the wing, a finite-element beam model is developed, which computes the bending of the wing. Using an MDA, the deformations and stresses that the wing undergoes during flight are computed. Then, the aerostructural model of the wing is optimized, using the MDF architecture. Two objective functions are used.

Chapter 2

MDAO Theory

2.1 Multidisciplinary Analysis (MDA)

A Multidisciplinary Analysis or MDA is the process of solving the equations of the multidisciplinary model. This is equivalent to driving the residuals of all disciplines to zero simultaneously, or converging the coupling variables to a state of equilibrium, effectively making all discipline states compatible. When considering the residual form of the multidisciplinary model (eq. 1.3), any conventional solver for systems of nonlinear equations will suffice, since this approach essentially results in one large set of nonlinear equations, which are solved in a monolithic fashion. For the functional form, the system of equations (eq. 1.5) is typically solved by a fixed-point method in segregated manner. Generic nonlinear solvers can also be used for functional form models, resulting in a semi-segregated, semi-monolithic approach.

2.1.1 Fixed-point methods

Fixed-point methods use appropriate iterates to arrive at the solution that satisfies the governing equations of all disciplines. In general, the iterate is of the form:

$$\hat{Y}_i^k = \hat{Y}_i(\hat{Y}_{j\neq i}^{k-1}),\ i \in (1, m) \tag{2.1}$$

where index k refers to the current MDA iteration. The MDA analog of the Jacobi method used for systems of linear equations, called Jacobi MDA, is obtained by eq. (2.1) for:

$$\hat{Y}_{j\neq i}^k = \begin{bmatrix} \hat{Y}_1^k & \dots & \hat{Y}_{i-1}^{k-1} & \hat{Y}_{i+1}^{k-1} & \dots & \hat{Y}_m^{k-1} \end{bmatrix}^\top \tag{2.2}$$

Essentially, at each Jacobi MDA iteration all disciplinary analyses are executed using the previous coupling variable values $\hat{Y} = \begin{bmatrix} \hat{Y}_1^{k-1} & \dots & \hat{Y}_{i-1}^k & \hat{Y}_{i+1}^{k-1} & \dots & \hat{Y}_m^{k-1} \end{bmatrix}^\top$,

which were computed during the last iteration. This allows the concurrent execution of all disciplines, as is the case with the Jacobi method for linear systems. The process is shown in algorithm 1.

Algorithm 1 Jacobi MDA

1: Initialize all coupling variables: $\hat{Y}^0 = \begin{bmatrix} \hat{Y}_1^0 & \hat{Y}_2^0 & \dots & \hat{Y}_m^0 \end{bmatrix}^\top$
2: **while** $||\hat{Y}^k - \hat{Y}^{k-1}|| < \epsilon$ **do**
3: **for** i = 1, m **do**
4: $\hat{Y}_i^k = \hat{Y}_i(\hat{Y}_1^{k-1}, \dots, \hat{Y}_{i-1}^{k-1}, \hat{Y}_{i+1}^{k-1}, \dots, \hat{Y}_m^{k-1})$
5: **end for**
6: k = k + 1
7: **end while**

Similarly, the iterate for the Gauss-Seidel MDA is given by:

$$\hat{Y}_{j\neq i}^k = \begin{bmatrix} \hat{Y}_1^k & \dots & \hat{Y}_{i-1}^k & \hat{Y}_{i+1}^{k-1} & \dots & \hat{Y}_m^{k-1} \end{bmatrix}^\top \tag{2.3}$$

The Gauss-Seidel MDA uses the latest coupling variable values, meaning that disciplinary analyses cannot be executed in parallel. However, the convergence properties are generally better than those of the Jacobi MDA, as shown in [2]. The process for the Gauss-Seidel MDA is shown in algorithm 2.

Algorithm 2 Gauss-Seidel MDA

1: Initialize all coupling variables: $\hat{Y}^0 = \begin{bmatrix} \hat{Y}_1^0 & \hat{Y}_2^0 & \dots & \hat{Y}_m^0 \end{bmatrix}^\top$
2: **while** $||\hat{Y}^{k+1} - \hat{Y}^k|| < \epsilon$ **do**
3: **for** i = 1, m **do**
4: $\hat{Y}_i^k = \hat{Y}_i(\hat{Y}_1^{k+1}, \dots, \hat{Y}_{i-1}^k, \hat{Y}_{i+1}^{k-1}, \dots, \hat{Y}_m^{k-1})$
5: **end for**
6: k = k + 1
7: **end while**

Both methods can make use of relaxation, namely:

$$\hat{Y}_i^k = (1-a)\hat{Y}_i^{k-1} + a\ \hat{Y}_i(\hat{Y}_{j\neq i}^k) \tag{2.4}$$

The use of under-relaxation ($a < 1$) can be very beneficial for MDA applications, in order to facilitate convergence[1].

[1] An MDA (for a functional form model) is considered converged when an aggregate scalar quantity of the functional residuals of all disciplines, called residual metric, is lower than some (MDA) tolerance ϵ. This, of course, is implementation dependant but in mSense, the MDA residual metric is defined as follows: $||\hat{Y}^{k+1} - \hat{Y}^k|| = \sum_{i=1}^m \frac{\|Y_i^{k+1} - Y_i^k\|}{1+\|Y_i^{k+1}\|}$

2.1.2 Newton's method

Newton's method is a popular approach for solving systems of nonlinear equations. It starts from an initial guess, and at every iteration k it constructs a linear approximation to the solution of the nonlinear system of equations. By solving the resulting linear system, it produces a correction by which it updates the actual solution. For the residual form of the multidisciplinary model (eq. 1.3), Newton's method yields:

$$\begin{bmatrix} \frac{\partial R_1}{\partial Y_1}^{k-1} & \frac{\partial R_1}{\partial Y_2}^{k-1} & \cdots & \frac{\partial R_1}{\partial Y_m}^{k-1} \\ \frac{\partial R_2}{\partial Y_1}^{k-1} & \frac{\partial R_2}{\partial Y_2}^{k-1} & \cdots & \frac{\partial R_2}{\partial Y_m}^{k-1} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial R_m}{\partial Y_1}^{k-1} & \frac{\partial R_m}{\partial Y_2}^{k-1} & \cdots & \frac{\partial R_m}{\partial Y_m}^{k-1} \end{bmatrix} \begin{bmatrix} \Delta Y_1^k \\ \Delta Y_2^k \\ \vdots \\ \Delta Y_N^k \end{bmatrix} = \begin{bmatrix} R_1^{k-1} \\ R_2^{k-1} \\ \vdots \\ R_m^{k-1} \end{bmatrix} \tag{2.5}$$
$$Y_i^k = Y_i^{k-1} - \Delta Y_i^k, \ i \in (1, m)$$

At iteration k, the residuals and their derivatives are computed using the state variable values from the previous iteration. Specifically, the residuals are computed as $R_i^{k-1} = R_i(Y_1^{k-1}, Y_2^{k-1}, \ldots, Y_m^{k-1})$ and their derivatives as $\frac{\partial R_i}{\partial Y_j}^{k-1}(Y_1^{k-1}, Y_2^{k-1}, \ldots, Y_m^{k-1})$. It is important to note that in the above expression all entries are either vectors or matrices. For example, entry $\frac{\partial R_i}{\partial Y_j}$ is a vector of size $n_i \times n_j$, where n_i and n_j are the sizes of the state vectors of disciplines i and j respectively. This means that each discipline has to provide the derivative of n_{R_i} residuals w.r.t $\sum_{j=1}^{m} n_{Y_j}$ state variables, which basically forms the i-th row of the Jacobian matrix of 2.5. Of course, this can only be done if each and every disciplinary solver gives access to its internals, and specifically the computation of its residuals. The process for Newton's method for multidisciplinary models in residual form is shown in algorithm 3.

Applying Newton's method to residual form models results in an entirely monolithic solution process. In contrast, the fixed-point methods discussed previously solve the MDA problem in an segregated manner. A third approach is produced by applying Newton's method to the functional form of a multidisciplinary system. The functional form residual of each discipline is defined as $\hat{R}_i = \hat{Y}_i - \hat{Y}_i(\hat{Y}_{j \neq m})$. Applying Newton's method produces the following system:

$$\begin{bmatrix} \frac{\partial \hat{R}_1}{\partial \hat{Y}_1}^{k-1} & \frac{\partial \hat{R}_1}{\partial \hat{Y}_2}^{k-1} & \cdots & \frac{\partial \hat{R}_1}{\partial \hat{Y}_m}^{k-1} \\ \frac{\partial \hat{R}_2}{\partial \hat{Y}_1}^{k-1} & \frac{\partial \hat{R}_2}{\partial \hat{Y}_2}^{k-1} & \cdots & \frac{\partial \hat{R}_2}{\partial \hat{Y}_m}^{k-1} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial \hat{R}_m}{\partial \hat{Y}_1}^{k-1} & \frac{\partial \hat{R}_m}{\partial \hat{Y}_2}^{k-1} & \cdots & \frac{\partial \hat{R}_m}{\partial \hat{Y}_m}^{k-1} \end{bmatrix} \begin{bmatrix} \Delta \hat{Y}_1^k \\ \Delta \hat{Y}_2^k \\ \vdots \\ \Delta \hat{Y}_m^k \end{bmatrix} = \begin{bmatrix} \hat{R}_1^{k-1} \\ \hat{R}_2^{k-1} \\ \vdots \\ \hat{R}_m^{k-1} \end{bmatrix} \tag{2.6}$$
$$\hat{Y}_i^k = \hat{Y}_i^{k-1} - \Delta \hat{Y}_i^k, \ i \in (1, m)$$

Algorithm 3 Newton MDA (residual form)

Initialize all state variables: $Y^0 = \begin{bmatrix} Y_1^0 & Y_2^0 \dots Y_m^0 \end{bmatrix}^\top$
while $||R^k|| < \epsilon$ **do**
  for i = 1, m **do**
    Compute R_i^{k-1}
    for j = 1, m **do**
      Compute $\frac{\partial R_i}{\partial Y_j}^{k-1}$
    end for
  end for
  Solve $(\frac{\partial R}{\partial Y})^{k-1} \Delta Y^k = R^{k-1}$ for ΔY^k
  $Y^k = Y^{k-1} - \Delta Y^k$
  k = k + 1
end while

From the definition of the functional form residual, it follows that:

$$\frac{\partial \hat{R}_i}{\partial \hat{Y}_j} = \begin{cases} I, \ i = j \\ -\frac{\partial \hat{Y}_i}{\partial \hat{Y}_j}, \ i \neq j \end{cases} \tag{2.7}$$

Eq. 2.6 can be re-written as:

$$\begin{bmatrix} I & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_2}^{k-1} & \dots & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_m}^{k-1} \\ -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_1}^{k-1} & I & \dots & -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_m}^{k-1} \\ \vdots & \vdots & \dots & \vdots \\ -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_1}^{k-1} & -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_2}^{k-1} & \dots & I \end{bmatrix} \begin{bmatrix} \Delta \hat{Y}_1^k \\ \Delta \hat{Y}_2^k \\ \vdots \\ \Delta \hat{Y}_m^k \end{bmatrix} = \begin{bmatrix} \hat{R}_1^{k-1} \\ \hat{R}_2^{k-1} \\ \vdots \\ \hat{R}_m^{k-1} \end{bmatrix} \tag{2.8}$$

$$\hat{Y}_i^k = \hat{Y}_i^{k-1} - \Delta \hat{Y}_i^k, \ i \in (1, m)$$

Similar to eq. 2.8 all entries of the Jacobian matrix of 2.8 are also matrices. Each discipline has to provide the derivatives of $n_{\hat{Y}_i}$ output variables w.r.t $\sum_{j=1, j\neq i}^{m} n_{\hat{Y}_j}$. The process for Newton's method for functional form multidisciplinary models is shown in algorithm 4.

The system of eq. 2.5 will always be equal to or greater in size, compared to that of eq. 2.8, since for every discipline i, $n_{Y_i} \geq n_{\hat{Y}_i}$. However, assembling the Jacobian of the residual form (eq. 2.5) can be cheaper than its functional form counterpart (eq. 2.8). This is because obtaining the functional derivatives $\frac{\partial \hat{Y}_i}{\partial \hat{Y}_j}$ is analogous to obtaining total derivatives at the discipline level, the cost of which might not negligible.

Algorithm 4 Newton MDA (functional form)

Initialize all coupling variables: $\hat{Y}^0 = \begin{bmatrix} \hat{Y}_1^0 & \hat{Y}_2^0 & \dots & \hat{Y}_m^0 \end{bmatrix}^\top$
while $||\hat{Y}^k - \hat{Y}^{k-1}|| < \epsilon$ **do**
  for i = 1, m **do**
    Compute $\hat{R}_i^{k-1}$
    for j = 1, m **do**
      Compute $\frac{\partial \hat{Y}_i}{\partial \hat{Y}_j}^{k-1}$
    end for
  end for
  Solve $(\frac{\partial \hat{Y}}{\partial \hat{Y}})^{k-1} \Delta \hat{Y}^k = \hat{R}^{k-1}$ for $\Delta \hat{Y}^k$
  $\hat{Y}^k = \hat{Y}^{k-1} - \Delta \hat{Y}^k$
  k = k + 1
end while

2.2 Derivative computation for gradient-based MDO

In order to perform gradient-based optimization the derivatives or sensitivities of the objective function and the constraints w.r.t the design variables are required. In the context of MDAO, computation of these derivatives must also take the coupling between the disciplines into account. This is specifically required for MDO architectures that use an MDA to ensure multidisciplinary feasibility, such as MDF.

2.2.1 Derivatives of single-discipline models

Before presenting the various method of computing coupled derivatives of multi-disciplinary models, it can be useful to discuss derivative computation in single-discipline models. Consider a vector valued function F of size n_F and vector X of size n_X. F represents all the objectives and constraints of the system, and X the design variables vector. The (matrix) quantity that must be computed is the Jacobian matrix of F w.r.t X, namely:

$$\frac{dF}{dX} = \begin{bmatrix} \frac{dF_1}{dX_1} & \frac{dF_1}{dX_2} & \dots & \frac{dF_1}{dX_{n_X}} \\ \frac{dF_2}{dX_1} & \frac{dF_2}{dX_2} & \dots & \frac{dF_2}{dX_{n_X}} \\ \vdots & \vdots & \dots & \vdots \\ \frac{dF_{n_F}}{dX_1} & \frac{dF_{n_F}}{dX_2} & \dots & \frac{dF_{n_F}}{dX_{n_X}} \end{bmatrix} \tag{2.9}$$

In engineering applications it is more common for F to be directly computed from the state variables of the corresponding model, rather than from X directly.

In general, the dependence of F on X can be both direct and indirect. For a single-discipline model with state vector Y, of size n_Y, and through the chain-rule of differentiation, it follows that:

$$\frac{dF}{dX} = \frac{\partial F}{\partial X} + \frac{\partial F}{\partial Y}\frac{dY}{dX} \tag{2.10}$$

For a given design variable vector X and a state vector Y that satisfies the model equations, the residuals become zero, namely $R(X, Y) = 0$, where R is the residual vector of the system. Again, through the chain-rule of differentiation and noting that $\frac{dR}{dX} = 0$, since $R = 0$, it follows that:

$$\frac{\partial R}{\partial Y}\frac{dY}{dX} = -\frac{\partial R}{\partial X} \tag{2.11}$$

By solving eq. 2.11 for $\frac{dY}{dX}$ and substituting the result in eq. 2.10, the total sensitivity of F w.r.t X is computed. Eqs. 2.10 and 2.11 constitute the **direct differentiation** method. In order to compute $\frac{dF}{dX}$, this method requires the solution of n_X linear systems (of size $n_Y \times n_Y$ each), i.e. its cost scales with the number of design variables. Through substitution and clever rearrangement of operations, a second method can be derived. By substituting eq. 2.11 in eq. 2.10 the following is obtained:

$$\frac{dF}{dX} = \frac{\partial F}{\partial X} - \frac{\partial F}{\partial Y}\left(\frac{\partial R}{\partial Y}^{-1}\frac{\partial R}{\partial X}\right) \tag{2.12}$$

The order of operations in the last term of eq. 2.12 is slightly modified such that:

$$\begin{aligned} &\frac{dF}{dX} = \frac{\partial F}{\partial X} - \left(\frac{\partial F}{\partial Y}\frac{\partial R}{\partial Y}^{-1}\right)\frac{\partial R}{\partial X} \implies \\ &\frac{dF}{dX} = \frac{\partial F}{\partial X} - \psi^\top \frac{\partial R}{\partial X} \end{aligned}$$

Term $\psi^\top = \frac{\partial F}{\partial Y}\frac{\partial R}{\partial Y}^{-1}$ is the vector of adjoint variables. It can be computed by solving $\frac{\partial R}{\partial Y}^\top \psi = \frac{\partial F}{\partial Y}^\top$. This yields the **adjoint method**:

$$\begin{aligned} &\frac{dF}{dX} = \frac{\partial F}{\partial X} - \psi^\top \frac{\partial R}{\partial X} \\ &\frac{\partial R}{\partial Y}^\top \psi = \frac{\partial F}{\partial Y}^\top \end{aligned} \tag{2.13}$$

This approach requires the solution of n_F linear systems (also of size $n_Y \times n_Y$ each), meaning that it scales with the number of objectives and constraints. The adjoint method is often preferred in engineering design optimization, since often

$n_X >> n_F$, meaning that the number of design variables is far greater than the number of constraints and objectives.

Another approach of obtaining the total sensitivities $\frac{dF}{dX}$ is through approximation. Perhaps the most common approximation technique is **finite-differences**. A first-order, forward, finite-difference approximation of $\frac{dF}{dX}$ is computed as:

$$\frac{dF}{dX_i} = \frac{F(X + h\hat{e}_i) - F(X)}{h} \tag{2.14}$$

In the above equation $\hat{e}_i$ is a unit vector of size n_X, pointing in the i-th direction, and h is the approximation step-size. The step-size has to be small relative to the magnitude of X_i. The presented finite-difference scheme is only first order accurate, and other more precise schemes exist. A much more accurate variant of the finite-difference approximation, is the **complex-step** method [27]. As the name suggests, instead of taking a real step in the i-th direction, the complex-step method perturbs the function in the imaginary plane, i.e.:

$$\frac{dF}{dX_i} = \frac{Imag(F(X + \hat{j}h\hat{e}_i))}{h} \tag{2.15}$$

Although the complex-step method is much more accurate than any finite-difference scheme, it requires that the underlying numerical model supports complex-number arithmetic. Accuracy notwithstanding, all approximation methods suffer from the fact that they scale linearly with the number of design variables n_x, similar to the direct differentiation approach of eqs. 2.10 and 2.11.

2.2.2 Derivatives of multidisciplinary models

The methods presented in the previous section can also be applied to compute the (coupled) derivatives of multidisciplinary models with some adjustments. Consider again a vector valued function F, of size n_F, containing the objectives and constraints, and a design variables vector X, of size n_X. In the context of MDAO, the derivatives of F w.r.t X are referred to as coupled derivatives, or coupled sensitivities. Likewise, the direct differentiation and adjoint methods for both the residual and functions forms of the multidisciplinary model are referred to as the **coupled direct differentiation** and **coupled adjoint method** respectively. The coupled direct differentiation and adjoint methods were first presented by Sobieski in [34]. Martins [25] used the coupled adjoint method for a two-discipline, aerostructural system (expressed in the residual form), and compared its performance and accuracy with that of complex-step approximation.

For a model of m disciplines expressed in the residual form (eq. 1.3), the residual vector R_i of a discipline i is a function of all discipline state variables and the design variables, namely: $R_i(X, Y_1, \ldots, Y_m)$. Differentiating R_i w.r.t X yields:

$$\frac{dR_i}{dX} = \frac{\partial R_i}{\partial X} + \sum_{j=1}^{m} \frac{\partial R_i}{\partial Y_j} \frac{dY_j}{dX} = 0 \tag{2.16}$$

By rearranging, and writing the above for every discipline residual vector, the following linear system of equations is produced:

$$\begin{bmatrix} \frac{\partial R_1}{\partial Y_1} & \frac{\partial R_1}{\partial Y_2} & \cdots & \frac{\partial R_1}{\partial Y_m} \\ \frac{\partial R_2}{\partial Y_1} & \frac{\partial R_2}{\partial Y_2} & \cdots & \frac{\partial R_2}{\partial Y_m} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial R_m}{\partial Y_1} & \frac{\partial R_m}{\partial Y_2} & \cdots & \frac{\partial R_m}{\partial Y_m} \end{bmatrix} \begin{bmatrix} \frac{dY_1}{dX} \\ \frac{dY_2}{dX} \\ \vdots \\ \frac{dY_m}{dX} \end{bmatrix} = - \begin{bmatrix} \frac{\partial R_1}{\partial X} \\ \frac{\partial R_2}{\partial X} \\ \vdots \\ \frac{\partial R_m}{\partial X} \end{bmatrix} \tag{2.17}$$

The objectives and constraints vector F can now be differentiated w.r.t to X:

$$\frac{dF}{dX} = \frac{\partial F}{\partial X} + \begin{bmatrix} \frac{\partial F}{\partial Y_1} & \frac{\partial F}{\partial Y_2} & \cdots & \frac{\partial F}{\partial Y_m} \end{bmatrix} \begin{bmatrix} \frac{dY_1}{dX} \\ \frac{dY_2}{dX} \\ \vdots \\ \frac{dY_m}{dX} \end{bmatrix} \tag{2.18}$$

Eqs. 2.17 and 2.18 are the equivalent of the **direct differentiation** method (eqs. 2.10 and 2.11) for the residual form of multidisciplinary models. The adjoint approach is also obtainable, and is expressed as follows:

$$\frac{dF}{dX} = \frac{\partial F}{\partial X} - \begin{bmatrix} \Psi_1^\top & \Psi_2^\top & \cdots & \Psi_m^\top \end{bmatrix} \begin{bmatrix} \frac{\partial R_1}{\partial X} \\ \frac{\partial R_2}{\partial X} \\ \vdots \\ \frac{\partial R_m}{\partial X} \end{bmatrix} \tag{2.19}$$

The vector of the adjoint variables $\begin{bmatrix} \Psi_1^\top & \Psi_2^\top & \cdots & \Psi_m^\top \end{bmatrix}$ is computed from the solution of the following linear system:

$$\begin{bmatrix} \frac{\partial R_1}{\partial Y_1}^\top & \frac{\partial R_1}{\partial Y_2}^\top & \cdots & \frac{\partial R_1}{\partial Y_m}^\top \\ \frac{\partial R_2}{\partial Y_1}^\top & \frac{\partial R_2}{\partial Y_2}^\top & \cdots & \frac{\partial R_2}{\partial Y_m}^\top \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial R_m}{\partial Y_1}^\top & \frac{\partial R_m}{\partial Y_2}^\top & \cdots & \frac{\partial R_m}{\partial Y_m}^\top \end{bmatrix} \begin{bmatrix} \Psi_1 \\ \Psi_2 \\ \vdots \\ \Psi_m \end{bmatrix} = \begin{bmatrix} \frac{\partial F}{\partial Y_1}^\top \\ \frac{\partial F}{\partial Y_2}^\top \\ \vdots \\ \frac{\partial F}{\partial Y_m}^\top \end{bmatrix} \tag{2.20}$$

Eqs. 2.19 and 2.20 are the analogous to the **adjoint method** (eq. 2.13) for the residual form of multidisciplinary models. For residual form models, the size of each linear system that has to be solved, both for the coupled direction differentiation and coupled adjoint methods, is of size $n_Y \times n_Y$, where $n_Y = \sum n_{Y_i}{}_{i=1}^{m}$.

Both the direct differentiation and the adjoint approaches can also be obtained for models expressed in the functional form. For a model of m disciplines, the

output of discipline i is computed as $\hat{Y}_i = \hat{Y}_i(X, \hat{Y}_{j\neq i})$, being a function of all discipline outputs and the design variables. Differentiating the $\hat{Y}_i$ w.r.t X yields:

$$\frac{d\hat{Y}_i}{dX} - \sum_{j=1, j\neq i}^{m} \frac{\partial \hat{Y}_i}{\partial \hat{Y}_j} \frac{d\hat{Y}_i}{d\hat{X}} = \frac{\partial \hat{Y}_i}{\partial X} \tag{2.21}$$

Writing the above equation for all m disciplines, the following linear system is obtained:

$$\begin{bmatrix} I & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_2} & \dots & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_m} \\ -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_1} & I & \dots & -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_m} \\ \vdots & \vdots & \dots & \vdots \\ -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_1} & -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_2} & \dots & I \end{bmatrix} \begin{bmatrix} \frac{d\hat{Y}_1}{dX} \\ \frac{d\hat{Y}_2}{dX} \\ \vdots \\ \frac{d\hat{Y}_m}{dX} \end{bmatrix} = \begin{bmatrix} \frac{\partial \hat{Y}_1}{\partial X} \\ \frac{\partial \hat{Y}_2}{\partial X} \\ \vdots \\ \frac{\partial \hat{Y}_m}{\partial X} \end{bmatrix} \tag{2.22}$$

Differentiating the objective and constraints vector F w.r.t the design variables vector X yields:

$$\frac{dF}{dX} = \frac{\partial F}{\partial X} + \begin{bmatrix} \frac{\partial F}{\partial \hat{Y}_1} & \frac{\partial F}{\partial \hat{Y}_2} & \dots & \frac{\partial F}{\partial \hat{Y}_m} \end{bmatrix} \begin{bmatrix} \frac{d\hat{Y}_1}{dX} \\ \frac{d\hat{Y}_2}{dX} \\ \vdots \\ \frac{d\hat{Y}_m}{dX} \end{bmatrix} \tag{2.23}$$

Eqs. 2.22 and 2.23 represent the **direct differentiation method** for multidisciplinary systems in functional form. Again, by rearrangement, the adjoint method can be obtained:

$$\frac{dF}{dX} = \frac{\partial F}{\partial X} - \begin{bmatrix} \Phi_1^\top & \Phi_2^\top & \dots & \Phi_m^\top \end{bmatrix} \begin{bmatrix} \frac{\partial \hat{Y}_1}{\partial X} \\ \frac{\partial \hat{Y}_2}{\partial X} \\ \vdots \\ \frac{\partial \hat{Y}_m}{\partial X} \end{bmatrix} \tag{2.24}$$

The vector of adjoint variables is now denoted by Φ, to distinguish it from its residual form counterpart. It is obtained from the solution of the following linear system:

$$\begin{bmatrix} I & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_2}^\top & \dots & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_m}^\top \\ -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_1}^\top & I & \dots & -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_m}^\top \\ \vdots & \vdots & \dots & \vdots \\ -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_1}^\top & -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_2}^\top & \dots & I \end{bmatrix} \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_m \end{bmatrix} = \begin{bmatrix} \frac{\partial F}{\partial \hat{Y}_1}^\top \\ \frac{\partial F}{\partial \hat{Y}_2}^\top \\ \vdots \\ \frac{\partial F}{\partial \hat{Y}_m}^\top \end{bmatrix} \tag{2.25}$$

Eqs. 2.24 and 2.25 represent the **adjoint method** for multidisciplinary models in the functional form. For functional form models, the size of each linear system that has to be solved, both for the coupled direction differentiation and coupled adjoint methods, is of size $n_{\hat{Y}} \times n_{\hat{Y}}$, where $n_{\hat{Y}} = \sum {n_{\hat{Y}_i}}_{i=1}^{m}$.

The Jacobian matrices $\left[\frac{\partial Y_i}{\partial Y_j}\right]$ and $\left[\frac{\partial \hat{Y}_i}{\partial \hat{Y}_j}\right]$, present in the linear systems for both the coupled direct differentiation and adjoint approaches, are equivalent to the matrices in the left-hand-side of Newton's method for the residual (eq. 2.5) and functional (eq. 2.8) forms respectively. As it was noted when comparing the residual and functional Newton forms, the system arising from the functional form is typically significantly smaller in size, since for every discipline $n_{Y_i} \geq n_{\hat{Y}_i}$. However, obtaining the partial derivatives required in the functional direct differentiation and adjoint methods can be much more costly than the residual form counterparts. The terms $\frac{\partial \hat{Y}_i}{\partial \hat{Y}_j}$ and $\frac{\partial \hat{Y}_i}{\partial X}$ are partial derivatives at the multidisciplinary level, but are total derivatives at the discipline level. Both $\frac{\partial \hat{Y}_i}{\partial \hat{Y}_j}$ and $\frac{\partial \hat{Y}_i}{\partial X}$ are matrices of size $n_{\hat{Y}_i} \times n_{\hat{Y}_j}$ and $n_{\hat{Y}_i} \times n_X$ respectively. Therefore, each discipline has to provide the derivatives of $n_{\hat{Y}_i}$ objectives w.r.t $\sum_{j=1,j\neq i}^{m} n_{\hat{Y}_j} + n_X$ design variables. If a discipline uses the direct differentiation or adjoint methods presented in the previous section, providing these derivatives requires the solution of either $\sum_{j=1,j\neq i}^{m} n_{\hat{Y}_j} + n_X$ direct differentiation or $n_{\hat{Y}_i}$ adjoint linear systems. This becomes intractable even for moderately sized applications. The partial derivative terms $\frac{\partial R_i}{\partial Y_j}$ and $\frac{\partial R_i}{\partial X}$ appearing in the direct and adjoint methods for the residual form of the multidisciplinary model are often available or cheaply computed at the discipline level, and more importantly do not require any linear system solutions. Hence, in large multidisciplinary models, with computationally demanding disciplinary solvers, the residual form is better suited, at least for the computation of the coupled derivatives.

It should be noted that both finite-difference and complex-step approximations can also be used for evaluating the coupled derivatives in multidisciplinary models. Their poor scaling is much more pronounced here, since approximating the sensitivity to each design variable requires a separate MDA to be performed.

2.3 Multidisciplinary Design Optimization (MDO)

Multidisciplinary Design Optimization or MDO differs from single-discipline design optimization in that MDO has to take the interactions between the disciplines of the multidisciplinary model into account. The compatibility of discipline states is often referred to as multidisciplinary feasibility in the context of MDAO, and should not be confused with the notion of feasibility in classical optimization, which describes whether a certain design vector satisfies all constraints. Feasibility is a

requirement of any MDO formulation or architecture, and different formulations achieve it in different ways. These architectures can be placed into two broad categories: **monolithic** and **distributed**. Monolithic architectures formulate the MDO problem so that it can be solved as a single optimization problem. On the other hand, distributed architectures decompose the original problem into smaller optimization problems, with the hope of reducing the total computational cost, by exploiting the structure of the problem. In the following sections, examples of each category are presented.

2.3.1 Monolithic architectures

Multidisciplinary Feasible (MDF)

All monolithic architectures formulate and solve a single optimization problem. Perhaps the most straight forward monolithic architecture is the **Multidisciplinary Feasible (MDF)** architecture [24]. MDF constructs an optimization problem most similar to classic single-discipline optimization. Each MDF cycle begins by solving the multidisciplinary model's equations through an MDA. After all discipline states are compatible, namely after the MDA has converged, the objective and the constraints can be evaluated. Then, the coupled derivatives are computed, through the methods described in chapter 2.2.2. The values of the objective, the constraints and the derivatives are passed to the optimizer, which updates the design variables vector. For a model of m disciplines in functional form, the MDF architecture is expressed mathematically as:

$$\begin{aligned} \text{minimize} \quad & f(X, \hat{Y}) \\ \text{with respect to} \quad & X \\ \text{subject to} \quad & G(X, \hat{Y})) \leq 0 \\ & H(X, \hat{Y})) = 0 \\ \text{while solving} \quad & \hat{R}(X, \hat{Y}) = 0, \text{ for } \hat{Y} \end{aligned} \tag{2.26}$$

In the above expression f is the objective function, G and H are the vectors of all inequality and equality constraints respectively, and $\hat{R} = \left[\hat{R}_1, \hat{R}_2, ..., \hat{R}_m\right]^\top$ and $\hat{Y} = \left[\hat{Y}_1, \hat{Y}_2, ..., \hat{Y}_m\right]^\top$ are the concatenation of residual and output vectors of all disciplines. Specifically, $\hat{R}_i$ is the functional form residual for discipline i, defined as follows:

$$\hat{R}_i = \hat{Y}_i - \hat{Y}_i(\hat{Y}_{j \neq i}) \tag{2.27}$$

One major advantage of the MDF formulation is that feasibility is ensured at every optimization cycle, so even if optimization stops prematurely, the discipline states are compatible and the optimization result is meaningful. The disadvantage of MDF is that, at each cycle, a fully converged MDA solution is required, which can be costly. However, the MDF architecture is often the choice for large, PDE-constrained optimization problems, since it is the most effective architecture for systems with a large number of coupling variables and computationally expensive disciplinary solvers. This is because MDF can handle multidisciplinary models expressed both in the residual and functional forms, and leverage the efficiency of the coupled adjoint method (in the residual form). It is therefore commonly used in high-fidelity aerostructural optimization [16, 17], but is also used effectively in other MDO problems, such as the design and trajectory optimization of small satellite [15]. The architecture is described in algorithm 5.

Algorithm 5 Multidisciplinary Feasible (MDF)

1: Initialize $X, \hat{Y}$
2: **repeat**
3: Solve $\hat{R}(X, \hat{Y})) = 0,$ for $\hat{Y}$ (MDA)
4: Compute $f(X, \hat{Y}),\ G(X, \hat{Y}),\ H(X, \hat{Y})$
5: Compute $\frac{df}{dX}, \frac{dG}{dX}, \frac{dH}{dX}$ (Coupled derivatives, see Chapter 2.2.2)
6: Update X using $f, G, H, \frac{df}{dX}, \frac{dG}{dX}, \frac{dH}{dX}$
7: **until** Optimization has converged

Individual Discipline Feasible (IDF)

Another popular monolithic formulation is the **Individual Discipline Feasible or (IDF)** [24]. Instead of an MDA, the IDF architecture achieves feasibility by using target variables. The target variables, denoted by $\hat{Y}_i^t$, are estimates of the coupling variables $\hat{Y}_i$ which are entirely controlled by the optimizer and should match $\hat{Y}_i$ upon convergence. Each discipline i computes its outputs using as input not the outputs of other disciplines $\hat{Y}_{j\neq i}$, but the target variables $\hat{Y}_{j\neq i}^t = \begin{bmatrix}\hat{Y}_1^t & \dots & \hat{Y}_{i-1}^t & \hat{Y}_{i+1}^t & \hat{Y}_m^t\end{bmatrix}^\top$. This effectively decouples the disciplines, allowing the disciplinary solvers to execute independently. For every target variable vector $\hat{Y}_i^t$ an equality constraint $H_{\hat{Y}_i}^t(\hat{Y}_i^t, \hat{Y}_i) = \hat{Y}_i^t - \hat{Y}_i = 0$ is added. Through these constraints, the optimizer drives the value of $\hat{Y}_i^t$ to equal that of $\hat{Y}_i$. At optimization convergence and through satisfaction of the these constraints, multidisciplinary feasibility is ensured, since then $\hat{Y}_i^t = \hat{Y}_i$. Due to this, they are appropriately named **feasibility constraints**. For a multidisciplinary model of m disciplines in the functional form, IDF is expressed mathematically as:

$$
\begin{aligned}
\text{minimize} \quad & f(X, \hat{Y}^t) \\
\text{with respect to} \quad & X, \hat{Y}^t \\
\text{subject to} \quad & G(X, \hat{Y}^t) \leq 0 \\
& H(X, \hat{Y}^t) = 0 \\
& H_i^t(\hat{Y}_i^t, \hat{Y}_i) = \hat{Y}_i^t - \hat{Y}_i = 0, \text{ for} \in (1, m) \\
\text{while solving} \quad & \hat{R}_i(X, \hat{Y}_i, \hat{Y}_{j \neq i}^t) = 0, \text{ for } \hat{Y}_i, \ i \in (1, m)
\end{aligned}
\tag{2.28}
$$

The main advantage of IDF is that all disciplines can evaluate their outputs independently and in parallel. However, unlike MDF, the IDF formulation does not guarantee multidisciplinary feasibility at all optimization cycles. Therefore, if the optimization has to be stopped early, the discipline states might not be compatible and the optimization result might not be meaningful. A further disadvantage of IDF is that the optimizer most solve a much larger system. The introduction of the target variables forces the optimizer to handle $\sum_{i=1}^{m} n_{\hat{Y}_i}$ more "design" variables and constraints, since for the optimizer handles the target variables as if they were regular design/decision variables. This creates the need for efficient optimizers designed for large problems, when using this architecture. IDF is described in algorithm 6.

Algorithm 6 Individual Discipline Feasible (IDF)

1: Initialize $X, \hat{Y}^t$
2: **repeat**
3: **for** i = 1, m **do**
4: $\hat{Y}_i = \hat{Y}_i(X, \hat{Y}_{j \neq i}^t))$
5: $H_{\hat{Y}_i}^t(\hat{Y}_i^t, \hat{Y}_i) = \hat{Y}_i^t - \hat{Y}_i$
6: **end for**
7: Compute $f(X, \hat{Y}^t), G(X, \hat{Y}^t), \ H(X, \hat{Y}^t), \ H(X, \hat{Y}^t)$
8: Compute $\frac{df}{dX}, \frac{dG}{dX}, \frac{dH}{dX}, \frac{dH^t}{dX}, \frac{df}{d\hat{Y}^t}, \frac{dG}{d\hat{Y}^t}, \frac{dH}{d\hat{Y}^t}, \frac{dH^t}{d\hat{Y}^t}$
9: Update $X, \hat{Y}^t$ using $f, G, H, H^t, \frac{df}{dX}, \frac{dG}{dX}, \frac{dH}{dX}, \frac{dH^t}{dX}, \frac{df}{d\hat{Y}^t}, \frac{dG}{d\hat{Y}^t}, \frac{dH}{d\hat{Y}^t}, \frac{dH^t}{d\hat{Y}^t}$
10: **until** Optimization has converged

2.3.2 Distributed architectures

The second broad category of MDO architectures are distributed ones. Instead of solving a single optimization problem, they solve multiple smaller problems. Based on how they achieve feasibility, distributed architectures can be classified

as variants of the MDF and IDF (monolithic) architectures. Distributed MDF architectures use an MDA to ensure the compatibility of discipline states (similar to MDF), whereas distributed IDF architectures use target variables and feasibility constraints (similar to IDF). The Collaborative Optimization (CO) architecture is presented below, which is the only distributed architecture implemented in mSense.

Collaborative Optimization (CO)

Collaborative Optimization (CO) is a popular distributed architecture introduced by Braun [5]. CO decomposes the optimization problem into a system-level optimization problem and a separate optimization subproblem for each discipline. To decouple both the execution of the disciplinary solvers and the solution of the discipline subproblems, CO introduces target variables not only for the discipline outputs, but also for the shared and local design variables. Specifically, the system-level problem controls the shared design variables Z and target variables for the local design variables X_i and outputs $\hat{Y}_i$ of each discipline, denoted by X_i^t and $\hat{Y}_i^t$ respectively. The discipline i subproblem controls target variables for the shared design variables Z, denoted by Z_i^t, and its local design variables X_i. The mathematical formulation of the system-level problem is as follows:

$$\begin{aligned} \text{minimize} \quad & f(Z, X^t, \hat{Y}^t) \\ \text{with respect to} \quad & Z, X^t, \hat{Y}^t \\ \text{subject to} \quad & G_0(Z, X^t, \hat{Y}^t) \leq 0 \\ & H_0(Z, X^t, \hat{Y}^t) = 0 \\ \text{and} \quad & J_i^* = 0, \ i \in (1, m) \end{aligned} \tag{2.29}$$

In the above expression, X^t is a concatenation of the target variables for all local design variables, namely $X^t = \begin{bmatrix} X_1^t & X_2^t \dots & X_m^t \end{bmatrix}^\top$, similar to $\hat{Y}^\top = \begin{bmatrix} \hat{Y}_1^\top & \hat{Y}_2^\top & \dots & \hat{Y}_m^\top \end{bmatrix}^\top$ which includes the target variables for all discipline outputs. A set of equality constraints J_i^* is added, which enforce multidisciplinary feasibility at the optimum. These are feasibility constraints, similar in function to those used by IDF. The term J_i^* corresponds to the optimized solution of the discipline i subproblem, which is stated as follows:

$$
\begin{aligned}
\text{minimize} \quad & J_i(Z_i^t, X_i, \hat{Y}_i) = \|X_i^t - X_i\|^2 \\
& \qquad + \|Z_i^t - Z\|^2 \\
& \qquad + \|\hat{Y}_i^t - \hat{Y}_i\|^2 \\
\text{with respect to} \quad & Z_i^t,\ X_i \\
\text{subject to} \quad & G_i(Z_i^t, X_i, \hat{Y}_i) \leq 0 \\
& H_i(Z_i^t, X_i, \hat{Y}_i) = 0 \\
\text{while solving} \quad & \hat{R}_i(Z_i^t, X_i, \hat{Y}_{j\neq i}^t) \text{ for } \hat{Y}_i
\end{aligned}
\tag{2.30}
$$

The derivatives of the optimized discipline-level objectives J_i, denoted by J_i^*, are required by the system-level optimizer optimizer. Braun [6] showed that these post-optimality sensitivities can be computed as:

$$
\begin{aligned}
\frac{dJ_i^*}{dX_i^t} &= 2(X_i^t - X_i^*) \\
\frac{dJ_i^*}{dZ} &= -2((Z_i^t)^* - Z) \\
\frac{dJ_i^*}{dY_i^t} &= 2(Y_i^t - Y_i^*)
\end{aligned}
\tag{2.31}
$$

where $X_i^*, (Z_i^t)^*$ and Y_i^* are the optimized values of X_i, Z_i^t and Y_i, namely their values at the end of the i-th discipline-level optimization.

In essence, the concept behind CO is that the system-level problem minimizes the objective function, while the discipline subproblems minimize the interdisciplinary inconsistency. Since $J_i^* = 0$ at the optimum, the values of all targets match the values of the variables they are trying to estimate, namely $(X_i^t)^* = X_i^*$, $(Y_i^t)^* = Y_i^*$ and $(Z_i^t)^* = Z^*$.

CO has the obvious advantage that all disciplinary analyses and optimizations are completely separated. However, the formulation faces major convergence issues, especially when gradient-based optimizers are used [1]. This is because, at the optimum both the feasibility constraints J_i^* and their gradients are zero, which is a violation of the Karush-Khan-Tucker optimality conditions, significantly hindering convergence. Despite these drawbacks, the architecture has been used extensively in a wide range of MDO problems, including the design of flight trajectories [23], satellite constellations [7] and even a scanning optical microscope [29]. The architecture is described in algorithm 7.

Algorithm 7 Collaborative Optimization (CO)

Initialize $Z, Z_i^t, X_i, X_i^t, \hat{Y}_i, \hat{Y}_i^t$ for $i \in (1, m)$
repeat
  for i = 1, m **do**
    repeat
      Solve $\hat{R}_i(Z_i^t, X_i, \hat{Y}_{i\neq j}^t) = 0$ for $\hat{Y}_i$
      Compute $J_i(Z_i^t, X_i, \hat{Y}_i),\ G_i(Z_i^t, X_i, \hat{Y}_i),\ H_i(Z_i^t, X_i, \hat{Y}_i)$
      Compute $\frac{dJ_i}{dZ_i^t}, \frac{dG_i}{dZ_i^t}, \frac{dH_i}{dZ_i^t}, \frac{dJ_i}{dX_i}, \frac{dG_i}{dX_i}, \frac{dH_i}{dX_I}$,
      Update Z_i^t, X_i using $J_i, g_i, \frac{dJ_i}{dZ_i^t}, \frac{dG_i}{dZ_i^t}, \frac{dJ_i}{dX_i}, \frac{dG_i}{dX_i}$
    until Discipline-level optimization has converged
  end for
  Compute $f(Z, X^t, \hat{Y}^t), G_0(Z, X^t, \hat{Y}^t), H_0(Z, X^t, \hat{Y}^t)$
  Compute $\frac{df}{dZ}, \frac{dG_0}{dZ}, \frac{dH_0}{dZ}, \frac{dJ^*}{dZ}, \frac{df}{d\hat{X}^t}, \frac{dG_0}{d\hat{X}^t}, \frac{dH_0}{d\hat{X}^t}, \frac{dJ^*}{dX^t}, \frac{df}{d\hat{Y}^t}, \frac{dG_0}{d\hat{Y}^t}, \frac{dH_0}{d\hat{Y}^t}, \frac{dJ^*}{d\hat{Y}^t}$
  Update $Z, X^t, \hat{Y}^t$ using $f, G_0, J^*, \frac{df}{dZ}, \ldots$
until System-level optimization has converged

Chapter 3

Benchmarking different MDO architectures

In chapter 2.3 various MDO architectures were presented. The question that naturally arises is which architecture to use for a given problem. The ultimate goal of any architecture is to find the optimum of the MDO problem, without violating the constraints or multidisciplinary feasibility. However, when comparing architectures, it is not enough that an architecture can successfully arrive at the optimum. It is also necessary that it does so in a reasonable amount of time, or for a reasonable number of disciplinary evaluations. Many benchmark problems have been proposed, with the earliest example of such work perhaps being the NASA MDO test suite released in 1996 [28]. In this chapter, several MDO benchmark problems are presented and solved, with the goal of comparing the performance of the MDO architectures available in mSense.

3.1 Sellar's problem

Sellar's problem [32] is a simple mathematical problem comprised of two disciplines. It considers the minimization of a scalar function f w.r.t three design variables x_1, z_1, z_2. The problem is constrained by two inequality constraints g_1 and g_2. Design variable x_1 and both constraints are local, while z_1 and z_2 are shared. The outputs of the two disciplines are y_1 and y_2 respectively. The mathematical formulation of the problem follows, while the problem's XDSM is shown in fig. 3.1.

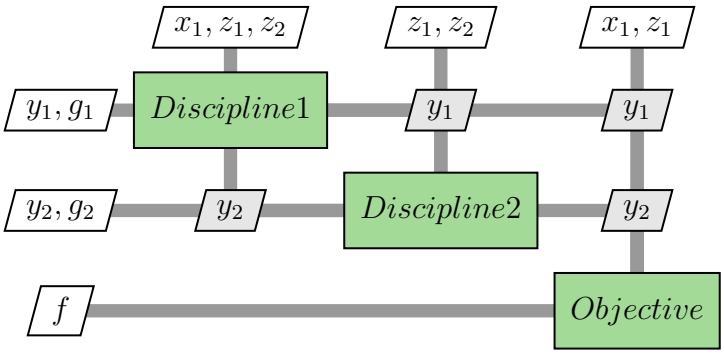


Figure 3.1: Sellar's problem: XDSM

$$
\begin{aligned}
\text{minimize} \quad & f = x_1^2 + z_1^2 + y_1^2 + e^{-y_2} \\
\text{with respect to} \quad & x_1, z_1, z_2 \\
\text{subject to} \quad & g_1 = 3.16 - y_1^2 \leq 0 \\
& g_2 = y_2 - 24 \leq 0 \\
\text{while satisfying} \quad & R_1 = y_1 - \sqrt{x_1^2 + z_2 + x_1 - 0.2 * y_2} = 0 \\
& R_2 = y_2 - |y_1| - z_1 - z_2 = 0
\end{aligned}
\tag{3.1}
$$

The three MDO architectures implemented in mSense are tested: MDF, IDF, and CO. For MDF, the Gauss-Seidel MDA is used and its termination tolerance is set to 10^{-4}. The coupled derivatives are computed using the coupled adjoint approach of eqs. 2.24 and 2.25. To make the application of the coupled adjoint method clear, eq. 2.25 yields:

$$
\begin{bmatrix} 1 & -\frac{\partial y_1}{\partial y_2} \\ -\frac{\partial y_2}{\partial y_1} & 1 \end{bmatrix}
\begin{bmatrix} \Phi_1 \\ \Phi_2 \end{bmatrix} =
\begin{bmatrix} \frac{\partial F}{\partial y_1}^\top \\ \frac{\partial F}{\partial y_2}^\top \end{bmatrix}
$$

where $F = \begin{bmatrix} f & g_1 & g_2 \end{bmatrix}^\top$. Since $n_F = 3$, the above 2×2 linear system has to be solved three times, before the coupled derivatives can be computed. For clarity's sake, by substituting F the above equation is written as:

$$
\begin{bmatrix} 1 & -\frac{\partial y_1}{\partial y_2} \\ -\frac{\partial y_2}{\partial y_1} & 1 \end{bmatrix}
\begin{bmatrix} \Phi_1^\top \\ \Phi_2^\top \end{bmatrix} =
\begin{bmatrix} \frac{\partial f}{\partial y_1} & \frac{\partial g_1}{\partial y_1} & \frac{\partial g_2}{\partial y_1} \\ \frac{\partial f}{\partial y_2} & \frac{\partial g_1}{\partial y_2} & \frac{\partial g_2}{\partial y_2} \end{bmatrix}
$$

Each column of the matrix on the right corresponds to the solution of a different linear system. Similarly, eq. 2.24 yields:

$$\frac{dF}{dX} = \frac{\partial F}{\partial X} - \begin{bmatrix}\Phi_1^\top & \Phi_2^\top\end{bmatrix} \begin{bmatrix}\frac{\partial y_1}{\partial X} \\ \frac{\partial y_2}{\partial X}\end{bmatrix}$$

where $X = \begin{bmatrix}z_1 & z_2 & x_1\end{bmatrix}^\top$. By substituting X and F, the equation is written as:

$$\begin{bmatrix}\frac{df}{dz_1} & \frac{df}{dz_2} & \frac{df}{dx_1} \\ \frac{dg_1}{dz_1} & \frac{dg_1}{dz_2} & \frac{dg_1}{dx_1} \\ \frac{dg_2}{dz_1} & \frac{dg_2}{dz_2} & \frac{dg_2}{dx_1}\end{bmatrix} = \begin{bmatrix}\frac{\partial f}{\partial z_1} & \frac{\partial f}{\partial z_2} & \frac{\partial f}{\partial x_1} \\ \frac{\partial g_1}{\partial z_1} & \frac{\partial g_1}{\partial z_2} & \frac{\partial g_1}{\partial x_1} \\ \frac{\partial g_2}{\partial z_1} & \frac{\partial g_2}{\partial z_2} & \frac{\partial g_2}{\partial x_1}\end{bmatrix} - \begin{bmatrix}\Phi_1^\top & \Phi_2^\top\end{bmatrix} \begin{bmatrix}\frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \frac{\partial y_1}{\partial x_1} \\ \frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \frac{\partial y_1}{\partial x_1}\end{bmatrix}$$

IDF solves a slightly larger optimization problem, as the optimizer is responsible not only for the design variables x_1, z_1, z_2, but also the target variables y_1^t and y_2^t. There is also the addition of the feasibility constraints, which are $h_1^t = y_1^t - y_1 = 0$ and $h_2^t = y_2^t - y_2 = 0$. Using these constraints the optimizer will drive the values of y_1^t and y_2^t to match the values of y_1 and y_2 at the optimum, namely $(y_1^t)^* = y_1^*$ and $(y_2^t)^* = y_2^*$, effectively enforcing multidisciplinary feasibility. For the sake of clarity, the optimization problem that IDF solves is the following:

$$\begin{aligned}
\text{minimize} \quad & f = x_1^2 + z_1^2 + (y_1^t)^2 + e^{-y_2^t} \\
\text{with respect to} \quad & x_1, z_1, z_2, y_1^t, y_2^t \\
\text{subject to} \quad & g_1 = 3.16 - (y_1^t)^2 \leq 0 \\
& g_2 = (y_2^t) - 24 \leq 0 \\
& h_1^t = y_1^t - y_1 = 0 \\
& h_2^t = y_2^t - y_2 = 0 \\
\text{while satisfying} \quad & R_1 = y_1 - \sqrt{x_1^2 + z_2 + x_1 - 0.2 * y_2^t} = 0 \\
& R_2 = y_2 - |y_1^t| - z_1 - z_2 = 0
\end{aligned}$$

CO formulates and solves a system-level problem and two discipline subproblems. The system level problem handles the shared design variables z_1 and z_2, and target variables for the local design variable x_1 and disciplinary outputs y_1 and y_2, denoted by x_1^t, y_1^t and y_2^t respectively. It is written as:

$$\begin{aligned}
\text{minimize} \quad & f = (x_1^t)^2 + z_1^2 + (y_1^t)^2 + e^{-y_2^t} \\
\text{with respect to} \quad & x_1^t, z_1, z_2, y_1^t, y_2^t \\
\text{subject to} \quad & J_1^* = 0 \\
& J_2^* = 0
\end{aligned}$$

where J_1^* and J_2^* are the optimized feasibility constraints of each subproblem. The subproblem for the first discipline handles x_1 and target variables for the shared design variables $z_{1,1}^t$ and $z_{2,1}^t$. It is formulated as follows:

$$
\begin{aligned}
\text{minimize} \quad & J_1 = (x_1^t - x_1)^2 \\
& \quad + (z_{1,1}^t - z_1)^2 \\
& \quad + (z_{2,1}^t - z_2)^2 \\
& \quad + (y_1^t - y_1)^2 \\
\text{with respect to} \quad & x_1, z_{1,1}^t, z_{2,1}^t \\
\text{subject to} \quad & g_1 = 3.16 - (y_1^t)^2 \leq 0 \\
\text{while satisfying} \quad & R_1 = y_1 - \sqrt{x_1^2 + z_{2,1}^t + x_1 - 0.2 * y_2^t} = 0
\end{aligned}
$$

The second discipline subproblem handles only target variables for the shared design variables $z_{1,2}^t$ and $z_{2,2}^t$. It is written as:

$$
\begin{aligned}
\text{minimize} \quad & J_2 = (z_{1,2}^t - z_1)^2 \\
& \quad + (z_{2,2}^t - z_2)^2 \\
& \quad + (y_2^t - y_2)^2 \\
\text{with respect to} \quad & z_{1,2}^t, z_{2,2}^t \\
\text{subject to} \quad & g_2 = (y_2^t) - 24 \leq 0 \\
\text{while satisfying} \quad & R_2 = y_2 - |y_1^t| - z_{1,2} - z_{2,2} = 0
\end{aligned}
$$

The system-level problem minimizes the objective function f, while respecting the $J_1^* = 0$ and $J_2^* = 0$ feasibility constraints. The discipline subproblems minimize J_i, while respecting g_i. The feasibility constraints $J_1^* = 0$ and $J_2^* = 0$ guarantee that at the optimum the values of all target variables match the value of their corresponding variable, for example $(x_1^t)^* = x_1^*$. The XDSMs for MDF, IDF, and CO are shown in figures 3.2,3.3 and 3.4 respectively.

The three architectures are compared in their ability to reach the optimum and the number of function calls required. In order to compute the disciplinary derivatives, for example $\frac{\partial y_1}{\partial x_1}$, $\frac{\partial y_1}{\partial y_2}$ or $\frac{\partial f}{\partial x_1}$, finite-differences with a step size of 10^{-4} are used for all architectures. The resulting MDO problems are solved using a Sequential Quadratic Programming (SQP) algorithm, namely SLSQP [20]. In the case of CO, SLSQP is used not only for the system-level problem, but also for the discipline-level subproblems. The termination tolerance for SLSQP is set to 10^{-8} and the maximum amount of cycles to be performed is set to 30. The starting and optimal values for the problem are shown in table 3.1. Reference optimal values are obtained from Sellar's original paper [32]. The evolution of the relative error for each architecture is shown in fig. 3.5. The relative error is defined as $\epsilon_{Relative} = \frac{f - f^*}{f^*}$, where f^* is the reference optimal objective value. The

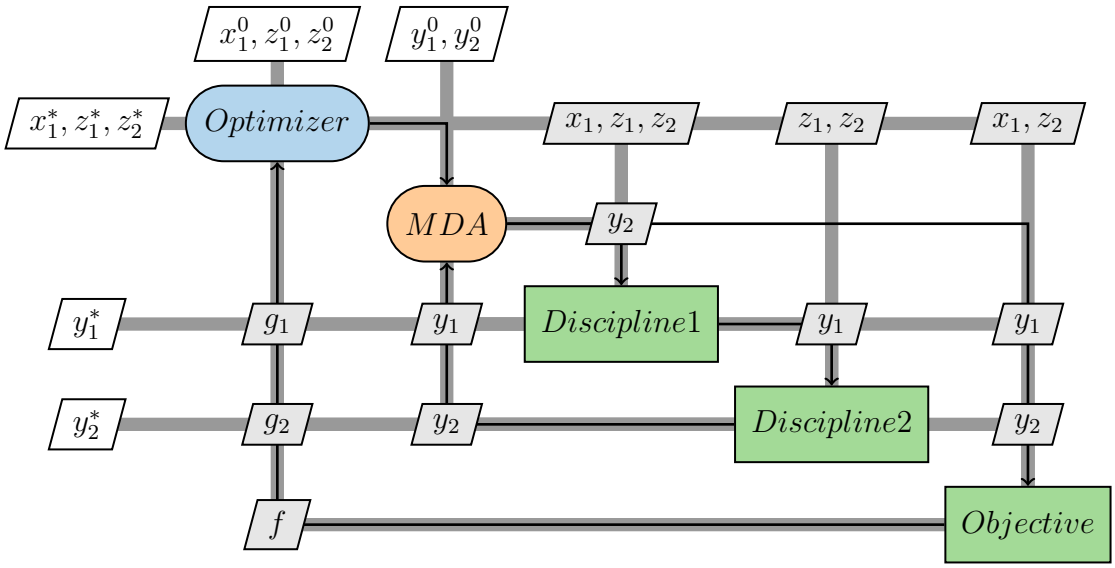


Figure 3.2: Sellar's problem: XDSM for the solution of the problem using MDF.

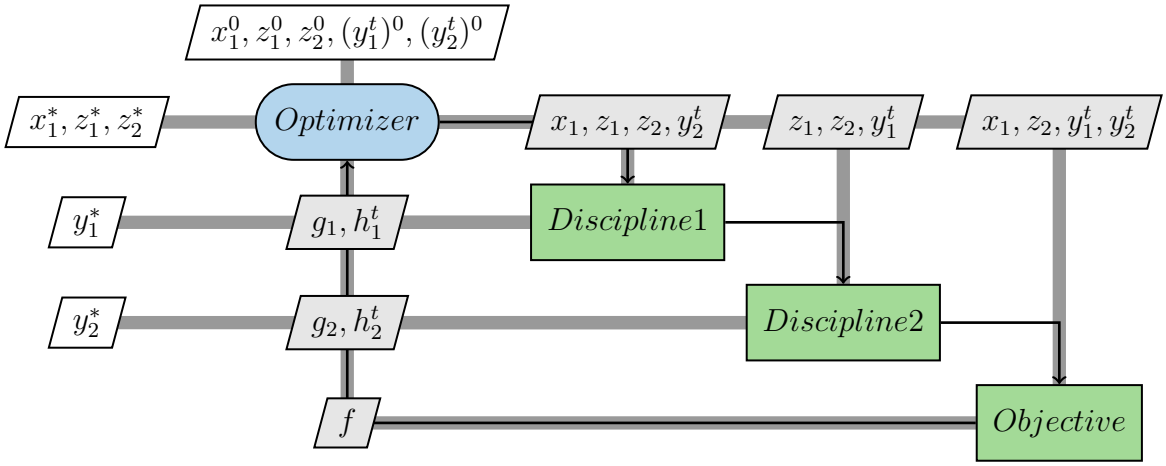


Figure 3.3: Sellar's problem: XDSM for the solution of the problem using IDF.

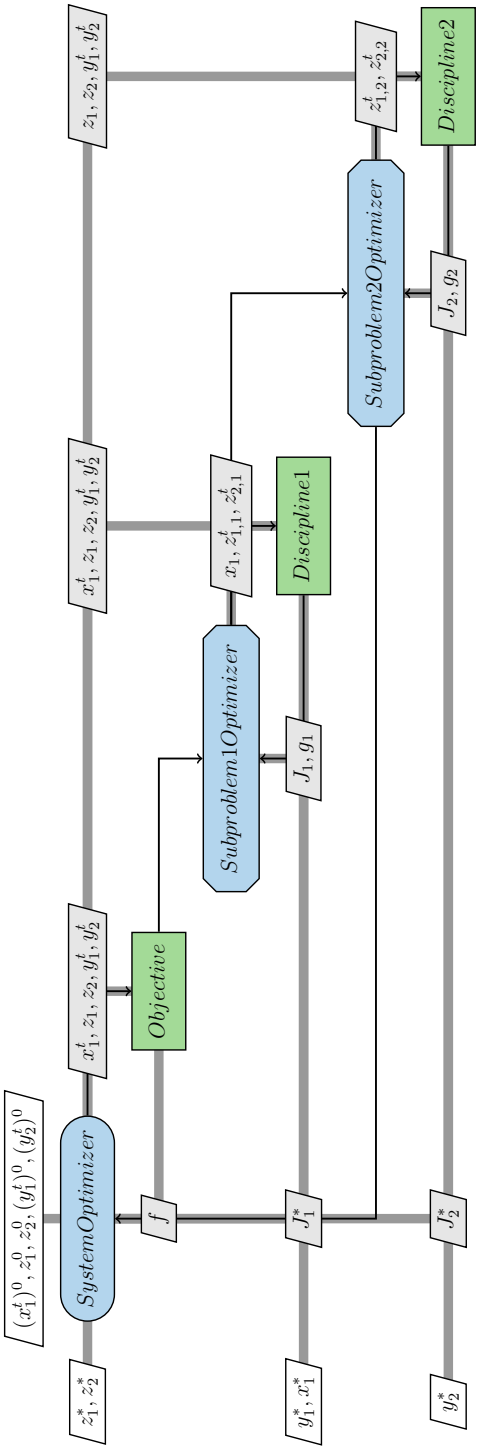


Figure 3.4: Sellar's problem: XDSM for the solution of the problem using CO.

Variable	Starting value	Optimal value
z_1	4	1.978
z_2	3	0
x	1.0	0
y_1	0.8	1.776
y_2	0.9	3.775
f	5.0465	3.1834

Table 3.1: Sellar's problem: Starting and reference optimal values, obtained from [32].

Architecture	MDF	IDF	CO
Discipline 1 evaluations	61	30	1210
Discipline 2 evaluations	53	24	601
Objective evaluations	61	30	168
Optimization cycles	9	7	30
Optimized objective value	3.18339	3.18339	3.1828

Table 3.2: Sellar's problem: Performance comparison between MDO architectures for. The number of evaluations also includes evaluations required for computing disciplinary derivatives through finite-differences.

number of evaluations for each discipline and the objective, as well as the number of optimization cycles required by each architecture are shown in table 3.2.

By observing the results, it is clear that IDF achieves the best performance, while CO performs worse than the two monolithic architectures. Because the selected starting values do not satisfy multidisciplinary feasibility, IDF and CO have to gradually drive the values of their respective feasibility constraints to zero. Both architectures are able to guide the solution to feasibility, although IDF does this in much fewer cycles, as seen when comparing figures 3.6 and 3.7. The performance of MDF is comparable to that of IDF, but the MDA solved at each iteration increases the disciplinary evaluations substantially. The poor convergence properties of CO are evident, even for this small mathematical problem. CO requires two orders of magnitude more disciplinary evaluations than IDF and MDF, many more optimizer iterations, and is still unable to precisely reach the optimum. The results are in agreement with those present in [12].

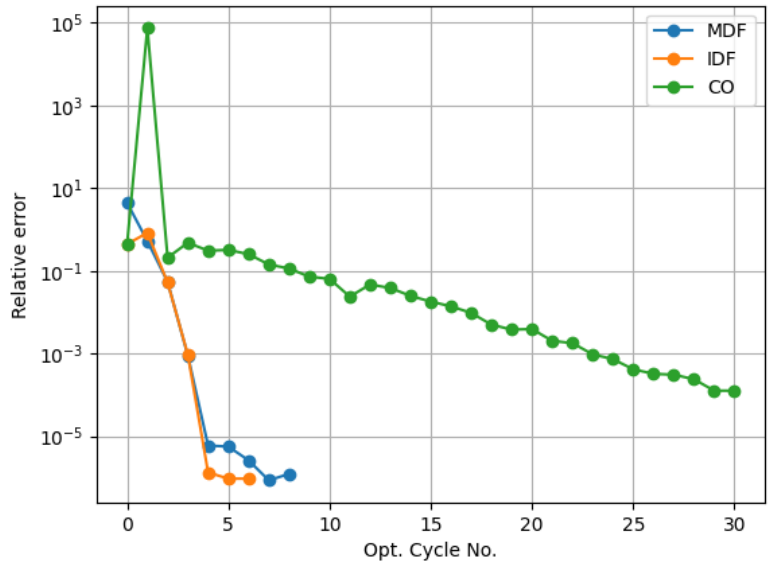


Figure 3.5: Sellar's problem: Evolution of the relative error. Comparison of the MDF, IDF and CO architectures. The relative error is defined as $\epsilon_{Relative} = \frac{f-f^*}{f^*}$, where f^* is the reference optimal objective value.

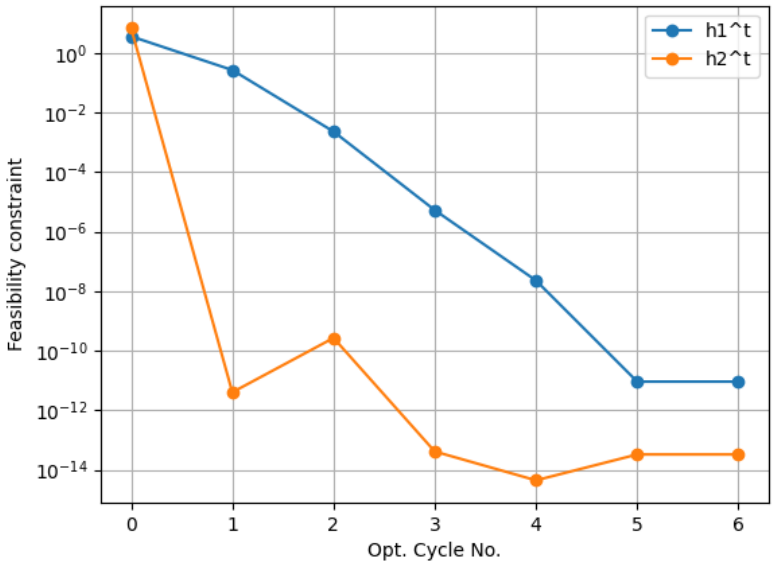


Figure 3.6: Sellar's problem: Evolution of IDF's feasibility constraints.

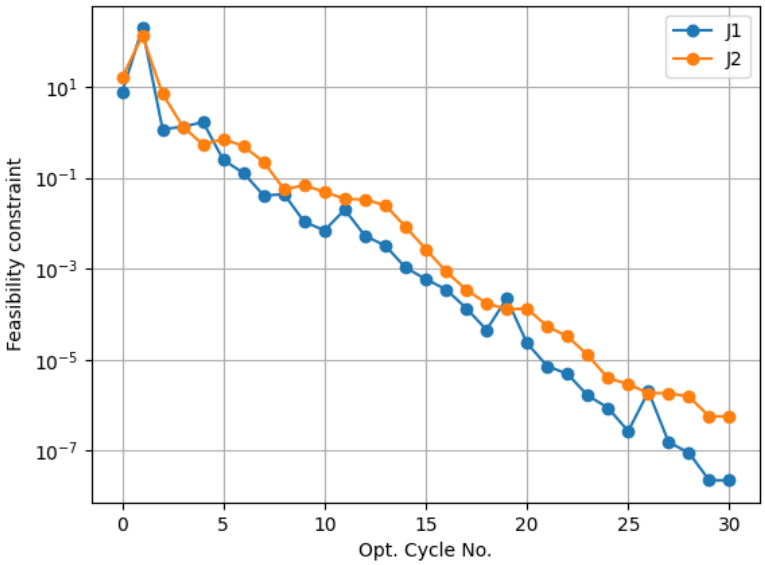


Figure 3.7: Sellar's problem: Evolution of CO's feasibility constraints.

3.2 Martins' scalable problem

This MDO problem has the particular feature that its dimensionality can be selected arbitrarily. Many of its parameters can be varied, resulting essentially in unique problems. These are:

- The number of disciplines, m
- The number of global design variables, n_Z
- The number of local design variables for discipline, i n_{X_i}
- The number of output variables for discipline, i n_{Y_i}

The problem is formulated mathematically as follows:

$$
\begin{aligned}
\text{minimize} \quad & f = \lambda_f (Z^\top Z + \sum_{i=0}^{m} Y_i^\top Y_i) \\
\text{with respect to} \quad & Z, X_i, \ i \in (1, m) \\
\text{subject to} \quad & G_i = Y_i - 1 \leq 0 \\
\text{while satisfying} \quad & R_i = C_{Y_i} Y_i - \lambda_Y (C_{Z_i} Z + C_{X_i} X_i - \sum_{j=0, j \neq i}^{j=m} C_{Y_j} Y_j) = 0, \ \in (1, m)
\end{aligned}
\tag{3.2}
$$

In the above expression Z is the vector of global design variables (size n_Z), X_i is the vector of local design variables for discipline i (size n_{X_i}) and Y_i is the vector of output variables for discipline i (size n_{Y_i}). The constraints G_i are local for each discipline. Coefficients λ_f and λ_Y are used so that the values of f and Y_i are scaled to around unity, and are computed as: $\lambda_f = (\sum_{i=0}^{m} n_{Y_i})^{-1}$ and $\lambda_Y = (m+1)^{-1}$. Matrices C_{Y_i}, C_{Z_i} and C_{X_i} can also be chosen arbitrarily, provided that all of the C_{Y_i} are non-singular. In the original paper [36], Martins chooses these matrices as follows:

- C_{Y_i} is an $n_{Y_i} \times n_{Y_i}$ identity matrix
- C_Z is a $n_{Y_i} \times n_Z$ matrix of ones
- C_{X_i} is a$n_{Y_i} \times n_{X_i}$ matrix of ones

Obviously, the inversion of the identity matrices C_{Y_i} is trivial, which makes solving the equation $R_i = 0$ very cheap. Although Martins uses finite-differences for computing the disciplinary partial derivatives $\frac{\partial Y_i}{\partial Z}$, $\frac{\partial Y_i}{\partial X_i}$ and $\frac{\partial Y_i}{\partial Y_j}$, here the differentiation is performed analytically as:

$$
\begin{aligned}
\frac{\partial Y_i}{\partial Z} &= C_{Y_i}^{-1} C_Z \\
\frac{\partial Y_i}{\partial X_i} &= C_{Y_i}^{-1} C_{X_i} \\
\frac{\partial Y_i}{\partial Y_j} &= C_{Y_i}^{-1} C_{Y_j}
\end{aligned}
\tag{3.3}
$$

Again, in the general case this requires inverting the matrix C_{Y_i} for each differentiation (or solving $m+1$ linear systems), but if C_{Y_i} is an identity matrix the cost is negligible. The XDSM for the scalable problem with $m = 3$ is shown in fig. 3.8.

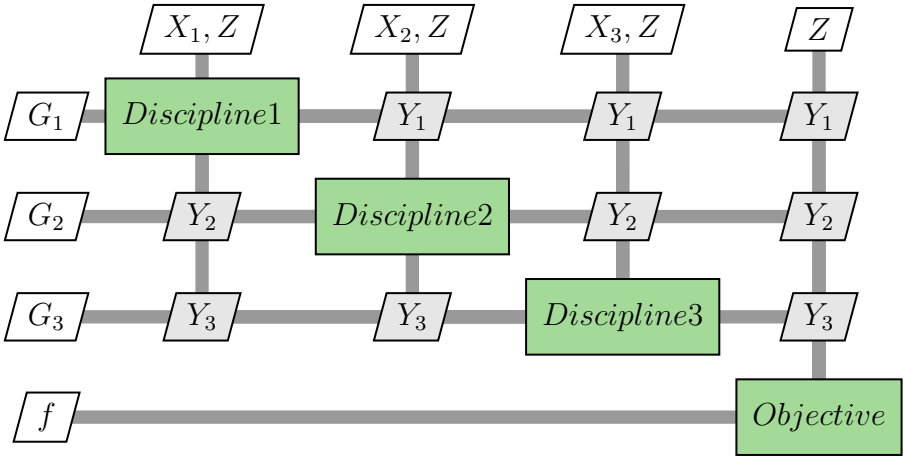


Figure 3.8: Martins' scalable problem: XDSM for three disciplines

The MDO problem is solved using the MDF, IDF and CO architectures. For MDF, the Gauss-Seidel MDA is used, with a termination tolerance of 10^{-4} and no relaxation. The coupled derivatives are again computed using the coupled adjoint method, similar to Sellar's problem. For $m = 3$, eq. 2.25 yields:

$$\begin{bmatrix} I & -\frac{\partial Y_1}{\partial Y_2}^\top & -\frac{\partial Y_1}{\partial Y_3}^\top \\ -\frac{\partial Y_2}{\partial Y_1}^\top & I & -\frac{\partial Y_2}{\partial Y_3}^\top \\ -\frac{\partial Y_3}{\partial Y_1}^\top & -\frac{\partial Y_3}{\partial Y_2}^\top & I \end{bmatrix} \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial Y_1}^\top & \frac{\partial G_1}{\partial Y_1}^\top & \frac{\partial G_2}{\partial Y_1}^\top & \frac{\partial G_3}{\partial Y_1}^\top \\ \frac{\partial f}{\partial Y_2}^\top & \frac{\partial G_1}{\partial Y_2}^\top & \frac{\partial G_2}{\partial Y_2}^\top & \frac{\partial G_3}{\partial Y_2}^\top \\ \frac{\partial f}{\partial Y_3}^\top & \frac{\partial G_1}{\partial Y_3}^\top & \frac{\partial G_2}{\partial Y_3}^\top & \frac{\partial G_3}{\partial Y_3}^\top \end{bmatrix}$$

Therefore, computation of the coupled derivatives requires solving a $n_Y \times n_Y$ linear system (where $n_Y = \sum_{i=1}^{3} n_{Y_i}$) a total of n_F times, where $n_F = 1 + \sum_{i=1}^{3} n_{G_i} = 1 + \sum_{i=1}^{3} n_{Y_i}$. After solution of the linear systems, the coupled derivatives are computed using eq. 2.24:

$$\begin{bmatrix} \frac{df}{dZ} & \frac{df}{dX_1} & \frac{df}{dX_2} & \frac{df}{dX_3} \\ \frac{dG_1}{dZ} & \frac{dG_1}{dX_1} & \frac{dG_1}{dX_2} & \frac{dG_1}{dX_3} \\ \frac{dG_2}{dZ} & \frac{dG_2}{dX_1} & \frac{dG_2}{dX_2} & \frac{dG_2}{dX_3} \\ \frac{dG_3}{dZ} & \frac{dG_3}{dX_1} & \frac{dG_3}{dX_2} & \frac{dG_3}{dX_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial Z} & \frac{\partial f}{\partial X_1} & \frac{\partial f}{\partial X_2} & \frac{\partial f}{\partial X_3} \\ \frac{\partial G_1}{\partial Z} & \frac{\partial G_1}{\partial X_1} & \frac{\partial G_1}{\partial X_2} & \frac{\partial G_1}{\partial X_3} \\ \frac{\partial G_2}{\partial Z} & \frac{\partial G_2}{\partial X_1} & \frac{\partial G_2}{\partial X_2} & \frac{\partial G_2}{\partial X_3} \\ \frac{\partial G_3}{\partial Z} & \frac{\partial G_3}{\partial X_1} & \frac{\partial G_3}{\partial X_2} & \frac{\partial G_3}{\partial X_3} \end{bmatrix} - \begin{bmatrix} \Phi_1^\top & \Phi_2^\top & \Phi_3^\top \end{bmatrix} \begin{bmatrix} \frac{\partial Y_1}{\partial Z} & \frac{\partial Y_1}{\partial X_1} & \frac{\partial Y_1}{\partial X_2} & \frac{\partial Y_1}{\partial X_3} \\ \frac{\partial Y_2}{\partial Z} & \frac{\partial Y_2}{\partial X_1} & \frac{\partial Y_2}{\partial X_2} & \frac{\partial Y_2}{\partial X_3} \\ \frac{\partial Y_3}{\partial Z} & \frac{\partial Y_3}{\partial X_1} & \frac{\partial Y_3}{\partial X_2} & \frac{\partial Y_3}{\partial X_3} \end{bmatrix}$$

IDF solves a larger optimization problem, introducing target variables Y_i^t and feasibility constraints $H_i^t = Y_i^t - Y_i = 0$. This means that the optimizer used by IDF has to handle $\sum_{i=0}^{m} n_{Y_i}$ extra design variables and constraints. For the sake of clarity, the optimization problem that IDF solves (for $m = 3$) is the following:

$$
\begin{aligned}
\text{minimize} \quad & f = \lambda_f(Z^\top Z + (Y_1^t)^\top(Y_1^t) + (Y_2^t)^\top(Y_2^t) + (Y_3^t)^\top(Y_3^t)) \\
\text{with respect to} \quad & Z, X_1, X_2, X_3, Y_1^t, Y_2^t, Y_3^t \\
\text{subject to} \quad & G_1 = Y_1 - 1 \leq 0 \\
& G_2 = Y_2 - 1 \leq 0 \\
& G_3 = Y_3 - 1 \leq 0 \\
& H_1^t = Y_1^t - Y_1 = 0 \\
& H_2^t = Y_2^t - Y_2 = 0 \\
& H_3^t = Y_3^t - Y_3 = 0 \\
\text{while satisfying} \quad & R_1 = C_{Y_1}Y_1 - \lambda_Y(C_{Z_1}Z + C_{X_1}X_1 - C_{Y_1}Y_1 - C_{Y_2}Y_2 - C_{Y_3}Y_3) = 0 \\
& R_2 = C_{Y_2}Y_2 - \lambda_Y(C_{Z_2}Z + C_{X_2}X_2 - C_{Y_1}Y_1 - C_{Y_2}Y_2 - C_{Y_3}Y_3) = 0 \\
& R_3 = C_{Y_3}Y_3 - \lambda_Y(C_{Z_3}Z + C_{X_3}X_3 - C_{Y_1}Y_1 - C_{Y_2}Y_2 - C_{Y_3}Y_3) = 0
\end{aligned}
$$

Finally, CO solves a system-level problem and three discipline subproblems. The system-level problem handles the global design variables Z and copies of the local design variables X_i^t and the coupling variables Y_i^t. The system-level problem is defined as follows ($m = 3$):

$$
\begin{aligned}
\text{minimize} \quad & f = \lambda_f(Z^\top Z + (Y_1^t)^\top(Y_1^t) + (Y_2^t)^\top(Y_2^t) + (Y_3^t)^\top(Y_3^t)) \\
\text{with respect to} \quad & Z, X_1^t, X_2^t, X_3^t, Y_1^t, Y_2^t, Y_3^t \\
\text{subject to} \quad & J_1^* = 0 \\
& J_2^* = 0 \\
& J_3^* = 0
\end{aligned}
$$

Each discipline subproblem handles its local design variables X_i and copies of the global design variables Z_i^t. The three discipline subproblems are defined as follows:

$$
\begin{aligned}
\text{minimize} \quad & J_1 = \|X_1^t - X_1\|^2 + \|Z_1^t - Z\|^2 + \|Y_1^t - Y_1\|^2 \\
\text{with respect to} \quad & Z_1^t, X_1 \\
\text{subject to} \quad & G_1 = Y_1 - 1 \leq 0 \\
\text{while satisfying} \quad & R_1 = C_{Y_1}Y_1 - \lambda_Y(C_{Z_1}Z + C_{X_1}X_1 - C_{Y_1}Y_1 - C_{Y_2}Y_2 - C_{Y_3}Y_3) = 0
\end{aligned}
$$

$$
\begin{aligned}
\text{minimize} \quad & J_2 = \|X_2^t - X_2\|^2 + \|Z_2^t - Z\|^2 + \|Y_2^t - Y_2\|^2 \\
\text{with respect to} \quad & Z_2^t, X_2 \\
\text{subject to} \quad & G_2 = Y_2 - 1 \leq 0 \\
\text{while satisfying} \quad & R_2 = C_{Y_2}Y_2 - \lambda_Y(C_{Z_2}Z + C_{X_2}X_2 - C_{Y_1}Y_1 - C_{Y_2}Y_2 - C_{Y_3}Y_3) = 0
\end{aligned}
$$

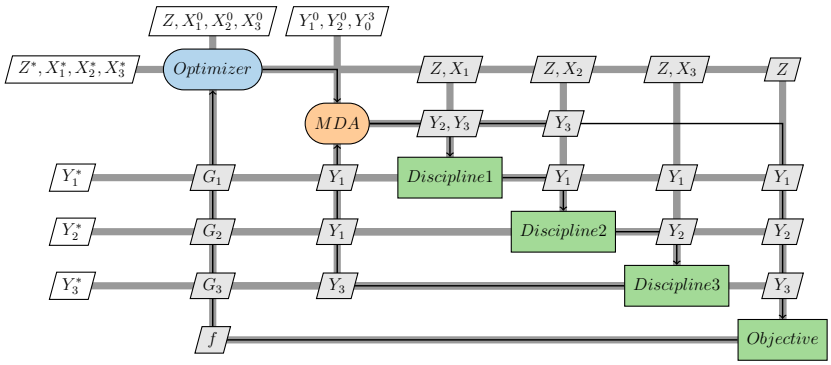


Figure 3.9: Martins' scalable problem: XDSM for the solution of the problem using MDF.

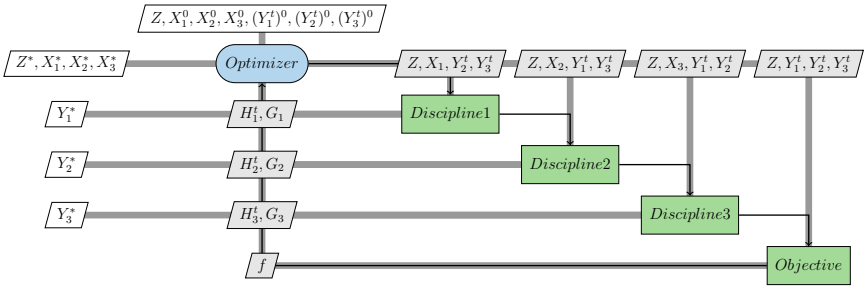


Figure 3.10: Martins' scalable problem: XDSM for the solution of the problem using IDF.

$$
\begin{aligned}
\text{minimize} \quad & J_3 = \|X_3^t - X_3\|^2 + \|Z_3^t - Z\|^2 + \|Y_3^t - Y_3\|^2 \\
\text{with respect to} \quad & Z_3^t, X_3 \\
\text{subject to} \quad & G_3 = Y_3 - 1 \leq 0 \\
\text{while satisfying} \quad & R_3 = C_{Y_3} Y_3 - \lambda_Y (C_{Z_3} Z + C_{X_3} X_3 - C_{Y_1} Y_1 - C_{Y_2} Y_2 - C_{Y_3} Y_3) = 0
\end{aligned}
$$

The system-level problem minimizes f, while the discipline subproblems minimize the disciplinary inconsistencies J_1, J_2 and J_3. The XDSMs for MDF, IDF and CO are shown in figures 3.9, 3.10 and 3.11 respectively.

The three architectures are compared in their ability to reach the optimum and the number of function calls required. Again, SLSQP is used as the optimizer. The termination tolerance is set to 10^{-8} and the maximum allowed number of cycles is 10. The problem is solved for $m = n_Z = n_{X_i} = n_{Y_i} = 3$, and the matrices C_{Y_i}, C_{Z_i} and C_{X_i} are chosen as above. The starting and reference optimal values for the problem are shown in table 3.3. The configuration of the problem and the

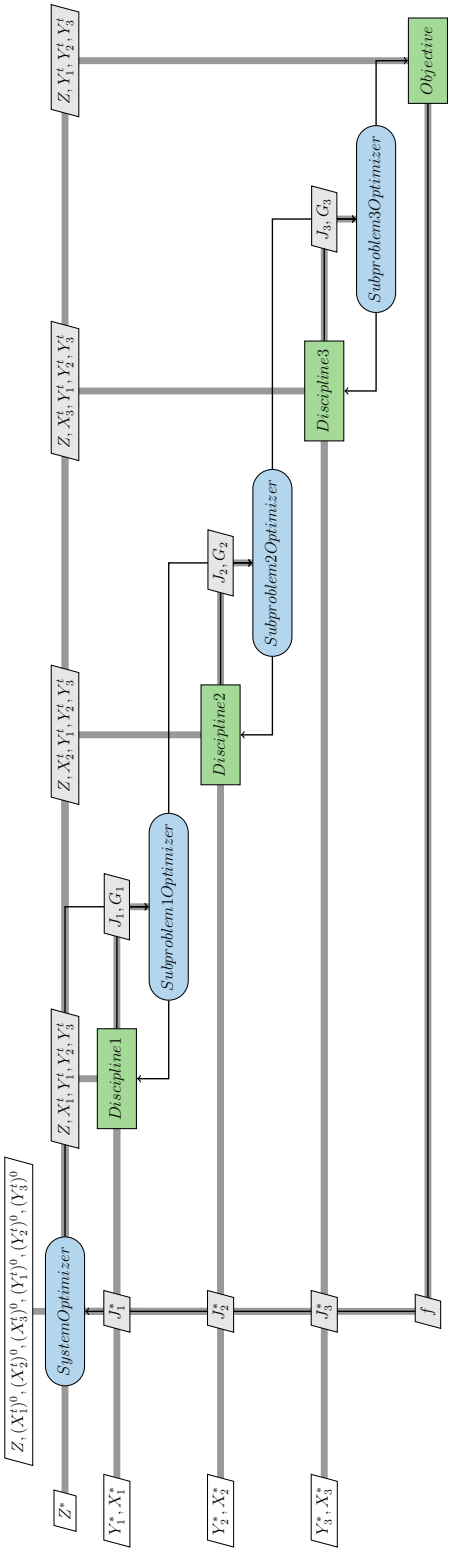


Figure 3.11: Martins' scalable problem: XDSM for the solution of the problem using CO.

Variable	Starting value	Optimal value
Z	-1	0
X_i	-1	-0.667
Y_i	0	1
f	9.334	1

Table 3.3: Martins' scalable problem: Starting and reference optimal values, obtained from [36]. Since Z, X_i and Y_i are vectors in general, the values provided are used for all their entries.

Architecture	MDF	IDF	CO
Total discipline evaluations	236	24	5396
Total discipline differentiations	16	20	2127
Optimization cycles	5	5	11
Optimized objective value	1	1	1.0003

Table 3.4: Martins' scalable problem: Performance comparison between the MDF, IDF and CO architectures.

starting and reference optimal values are the same as the ones used in the original paper by Martins [36]. The evolution of the relative error for each architecture is shown in fig. 3.12. The relative error is defined as $\epsilon_{Relative} = \frac{f-f^*}{f^*}$, where f^* is the reference optimal objective value. The number of evaluations for each discipline and the objective, as well as the number of optimization cycles required by each architecture are shown in table 3.4.

Similar to Sellar's problem, IDF performs the best and CO performs the worst out of the three architectures. MDF requires the same number of optimization cycles as IDF, but many more disciplinary evaluations because of the MDA it solves at every iteration. CO again requires two orders of magnitude more disciplinary evaluations and differentiations, compared to the other architectures. However, all three architectures are able to accurately reach the optimum. Furthermore, despite the starting point being infeasible (in the multidisciplinary sense), both the IDF and CO architectures are able to achieve feasibility, by driving their respective feasibility constraints close to zero. Similar to Sellar's problem, IDF converges to a feasible solution much faster than CO, as shown in figures 3.13 and 3.14. Interestingly, the evolution of all feasibility constraints for both disciplines is nearly identical, possibly because the disciplinary functions are the same.

3.2.1 Scalability study

By exploiting the fact that the size of Martin's problem can be selected arbitrarily, a study can be performed about the scaling characteristics of the used MDO archi-

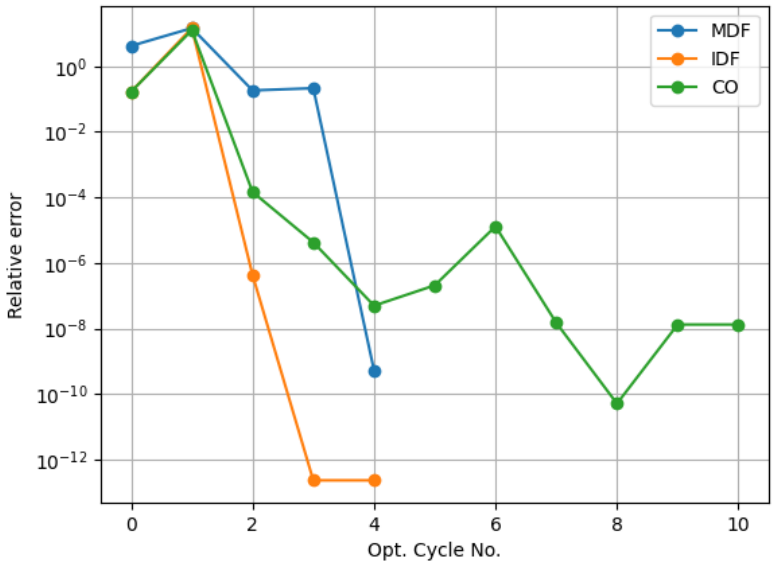


Figure 3.12: Martins' scalable problem: Evolution of the relative error. Comparison of the MDF, IDF and CO architectures. The relative error is defined as $\epsilon_{Relative} = \frac{f - f^*}{f^*}$, where f^* is the reference optimal objective value.

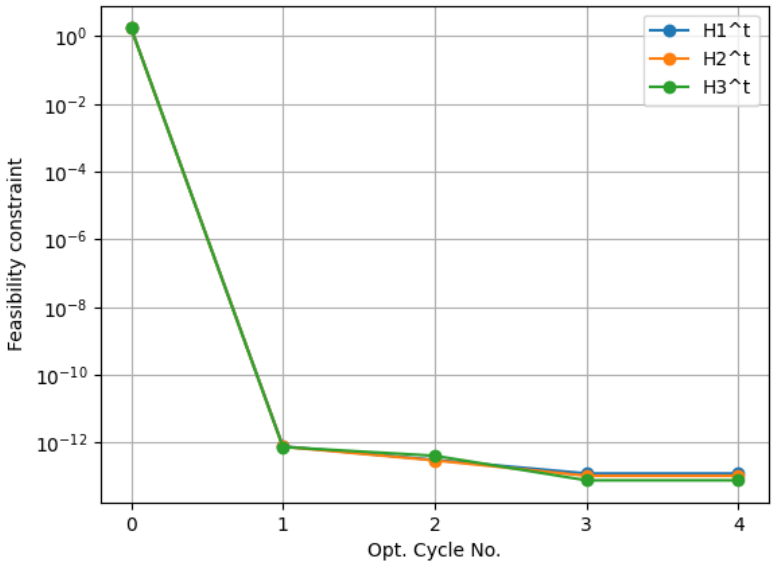


Figure 3.13: Martins' scalable problem: Evolution of IDF's feasibility constraints.

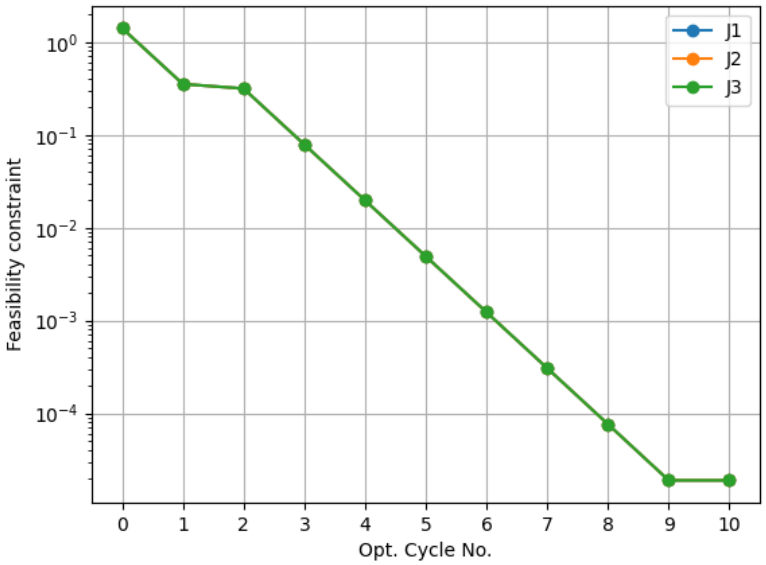


Figure 3.14: Martins' scalable problem: Evolution of CO's feasibility constraints.

tectures. Three disciplines are used ($m = 3$), and the number of design (n_{X_i} and n_Z) and coupling (n_{Y_i}) variables can vary. Only the MDF and IDF architectures are included in this study, since CO performed significantly worse for the baseline configuration of the problem. All studies are performed on laptop with a quad core 11th generation Intel i7 processor.

First, the performance of the two architectures is tested for an increasing number of design variables. The number of local design variables for each discipline (n_{X_i}) and the number of shared design variables (n_Z) are set equal ($n_{X_i} = n_Z$), and range from 10 to 130 with increments of 10. The number of coupling variables is set to 30 ($n_{Y_i} = 30$). Both the MDF and IDF architectures use SLSQP as the optimizer, with the termination tolerance set to 10^{-8} and a maximum of 10 cycles allowed. Figure 3.15 shows the solution time as a function of the total number of design variables. It is clear that the solution time for both architectures scales nonlinearly with the number of design variables. Interestingly, the number of optimization cycles, the number of total disciplinary evaluations and the number of total disciplinary differentiations do not change significantly with the number of design variables (figures 3.16, 3.17 and 3.18 respectively). Since the cost per disciplinary evaluation/differentiation is not heavily influenced by the number of design variables (eqs. 3.2 and 3.3), the increase in solution time comes mainly from the linear system that SLSQP has to solve at each cycle. The size of this system scales

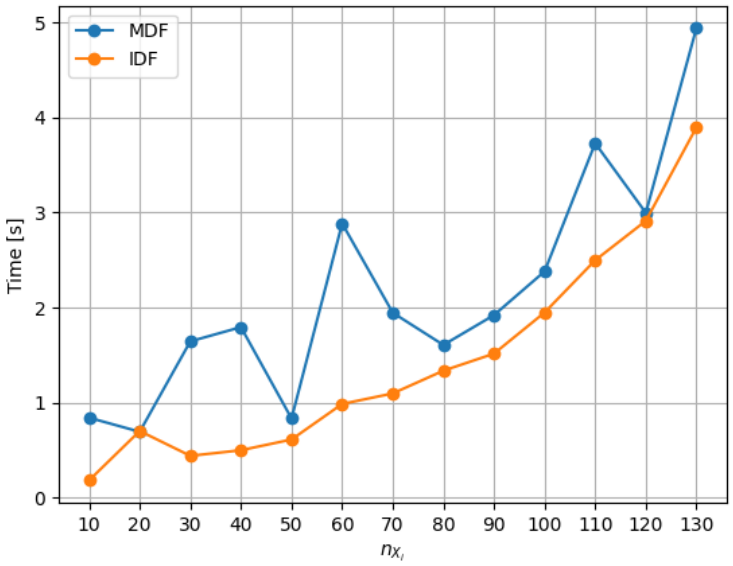


Figure 3.15: Martins' scalable problem: CPU time as a function of the number of design variables (local and shared, $n_{X_i} = n_Z$)).

directly with the number of variables the optimizer is responsible for. However, this is not an inherent drawback of the architectures, but rather, it is relevant to the choice of optimizer.

A second study is performed where the number of coupling variables per discipline n_{Y_i} is increased from 10 to 130 with increments of 10. The number of design variables is set to 30 ($n_{X_i} = n_Z = 30$). Again, SLSQP is used as the optimizer with settings identical to before. The solution time increases nonlinearly with increasing n_{Y_i} (fig. 3.19), but for IDF it seems to increase more rapidly, especially at $n_{Y_i} \geq 90$. Increasing n_{Y_i} directly impacts the time per disciplinary evaluation/differentiation, since the size of the linear system that each discipline solves is equal to $n_{Y_i} \times n_{Y_i}$. Furthermore, for the IDF architecture increasing the number of coupling variables essentially increases the number of design variables (since for each coupling a target variable is assigned), and, consequently, the size of the linear system that SLSQP solves at each cycle. MDF faces a similar problem, in that increasing the number of design variables increases the size of the linear system required for coupled derivative computation (either via the coupled direct eq. 2.22 or the coupled adjoint eq. 2.25 methods). Despite this and requiring many more disciplinary evaluations due to the MDA it solves at each cycle (fig. 3.21), MDF seems to handle the increasing number of design variables noticeably better than

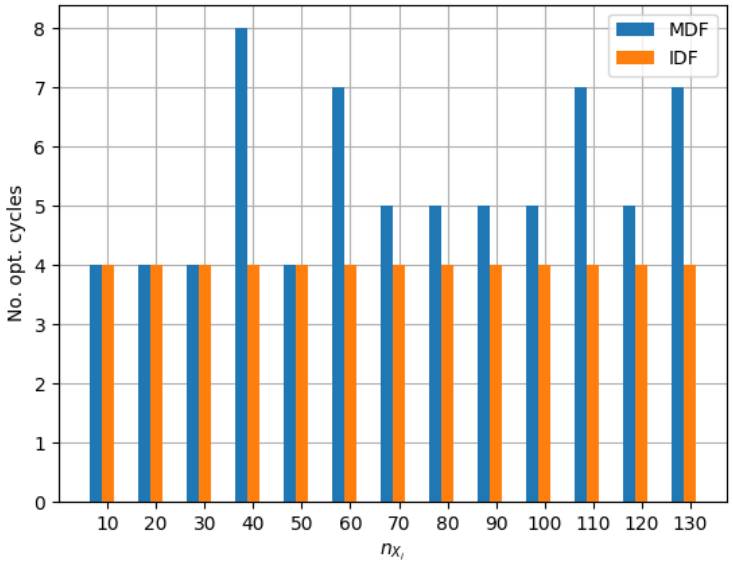


Figure 3.16: Martins' scalable problem: Number of optimization cycles as a function of the number of design variables (local and shared, $n_{X_i} = n_Z$)).

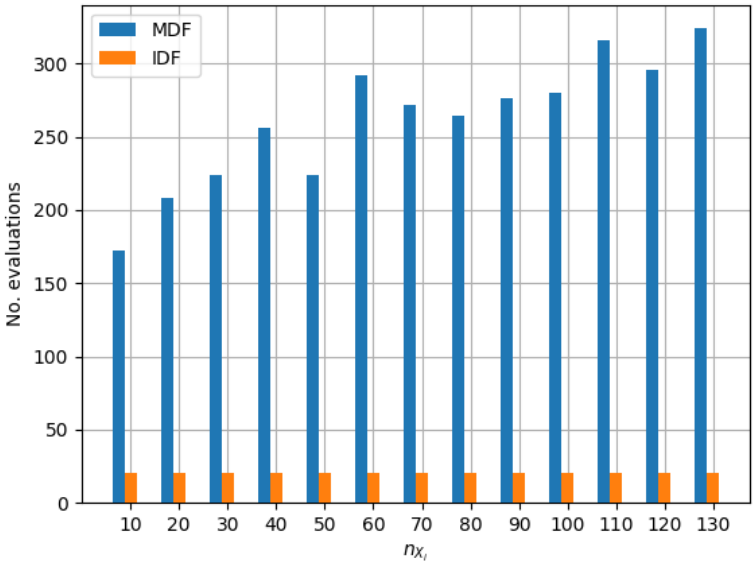


Figure 3.17: Martins' scalable problem: Number of total disciplinary evaluations as a function of the number of design variables (local and shared, $n_{X_i} = n_Z$)).

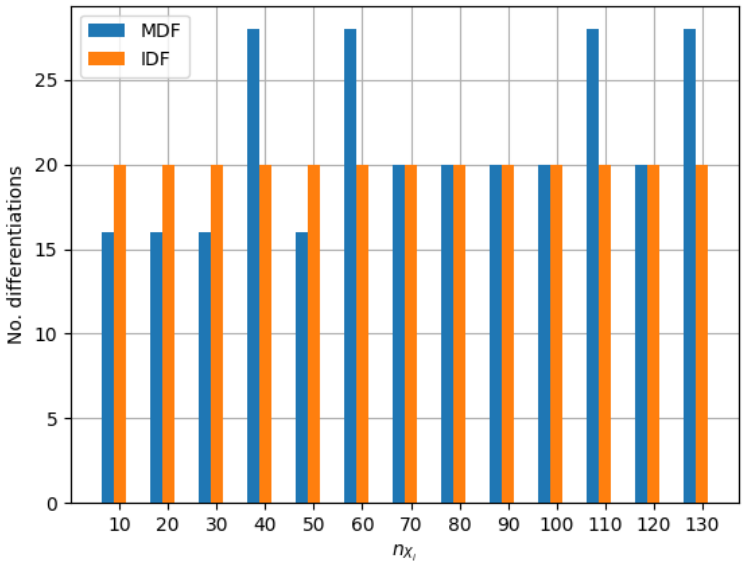


Figure 3.18: Martins' scalable problem: Number of total disciplinary differentiations as a function of the number of design variables (local and shared, $n_{X_i} = n_Z$)).

IDF for this problem, at least with respect to solution time. It should also be noted that besides the difference in disciplinary evaluations, the two architectures require about the same number of the more costly disciplinary differentiations (fig. 3.22) and optimization cycles. It can therefore be concluded that the performance of IDF with respect to the number of coupling heavily relies on the optimizer used. Due to this, MDF is perhaps the better choice for problems with many coupling variables, especially if no efficient, large-scale optimizer is available.

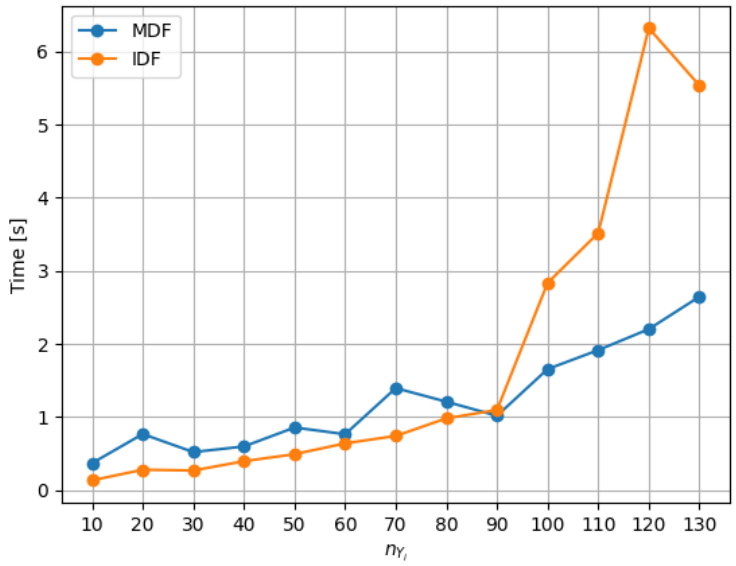


Figure 3.19: Martins' scalable problem: CPU time as a function of the number of coupling variables.

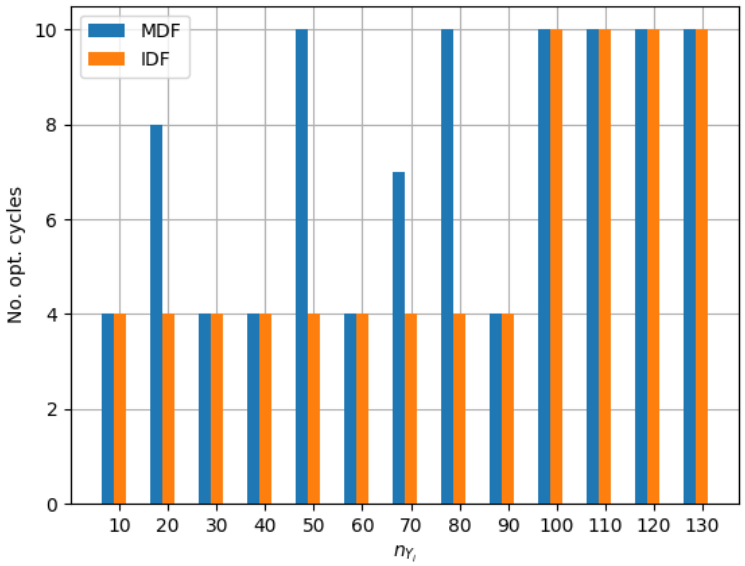


Figure 3.20: Martins' scalable problem: Number of optimization cycles as a function of the number of coupling variables.

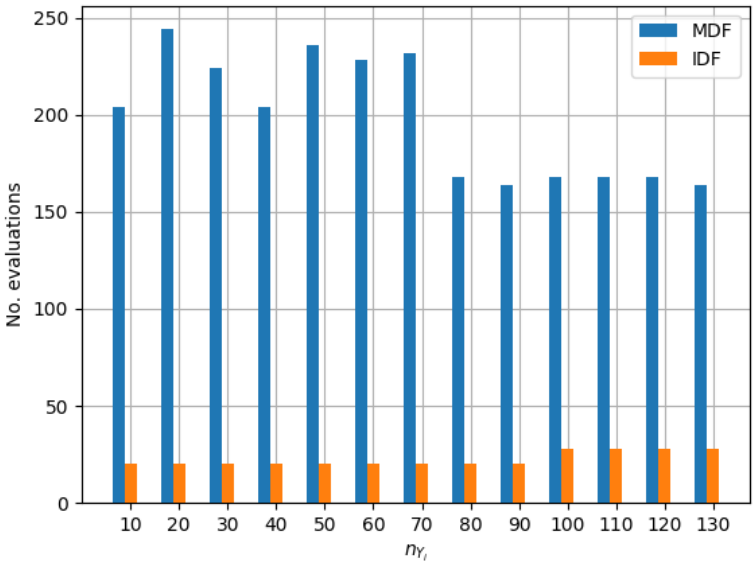


Figure 3.21: Martins' scalable problem: Number of total disciplinary evaluations as a function of the number of coupling variables.

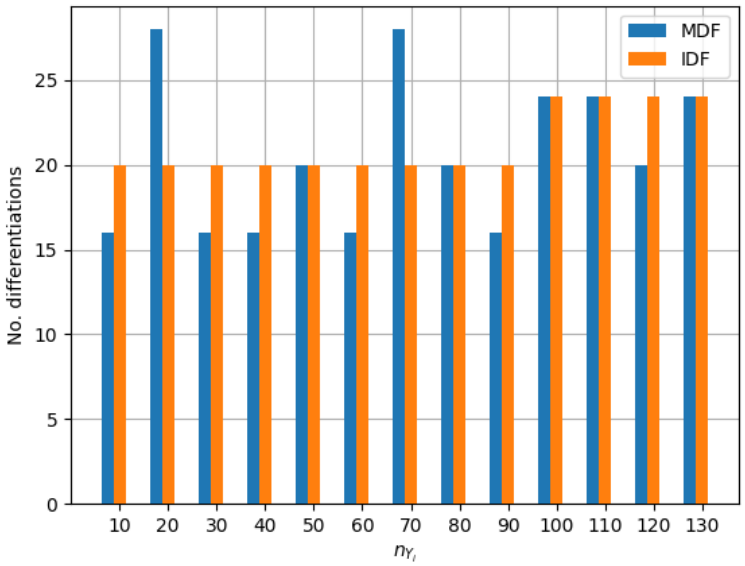


Figure 3.22: Martins' scalable problem: Number of total disciplinary differentiations as a function of the number of coupling variables.

Chapter 4

The airfoil-spring system

4.1 Problem description

A symmetric NACA-0012 airfoil is placed inside of a two-dimensional inviscid flow field. The airfoil is able to rotate about an axis normal to the flow plane, and passing through its quarter chord, where a torsional spring of stiffness J is attached. The freestream is horizontal, and has a magnitude of U_∞. The angle of attack α of the flow relative to the airfoil is:

$$\alpha = \alpha_\infty + \theta = \theta \tag{4.1}$$

where $\alpha_\infty = 0$ is aforementioned freestream angle and θ is the structural angle, namely the angle between the airfoil chord and the horizontal axis. The flow produces a moment M_{aero} about the quarter chord which, at equilibrium, is equal to the spring moment, $M_{spring} = J\theta$. By equating the two moments, the equilibrium angle θ for the airfoil-spring system can be computed:

$$M_{aero} = J\theta \tag{4.2}$$

The moment produced by the flow is, in general, a non-linear function of α_∞,

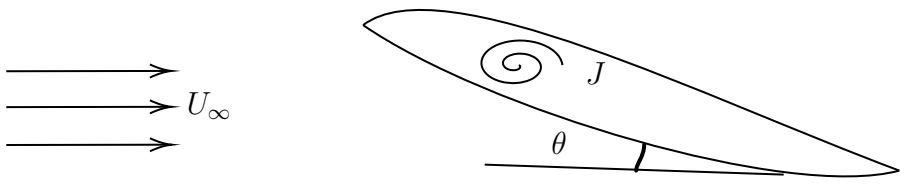


Figure 4.1: Airfoil-spring system: Schematic.

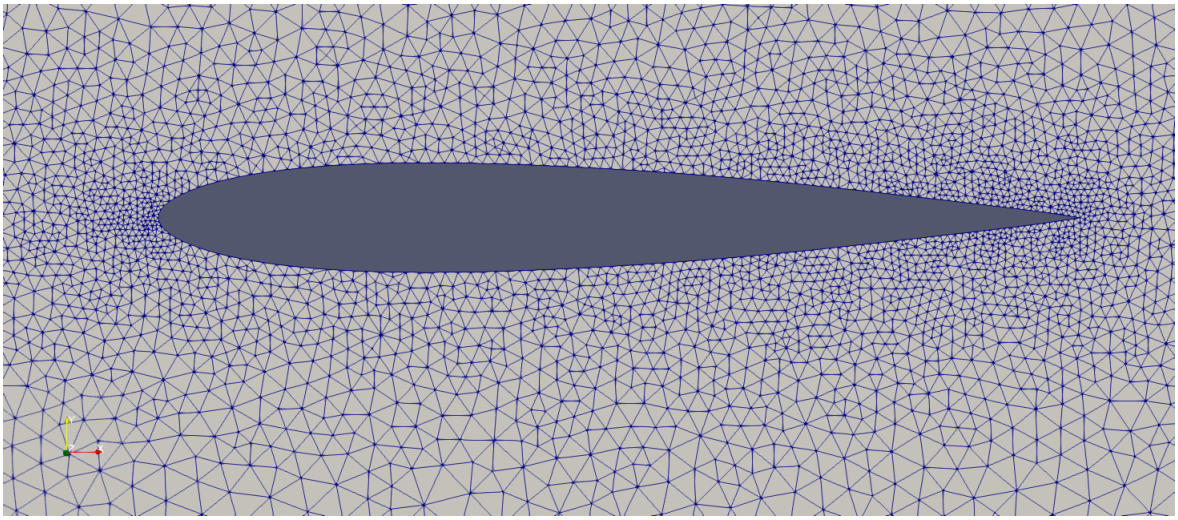


Figure 4.2: Airfoil-spring system: The unstructured mesh around the isolated NACA 0012 airfoil. The mesh is relatively coarse, since the flow is considered inviscid and the Euler equations are used.

or θ, namely: $M_{aero} = M_{aero}(\theta)$. Therefore, the above equation is a nonlinear equation in θ:

$$M_{aero}(\theta) - J\theta = 0 \tag{4.3}$$

The roots of this equation correspond to the equilibrium points of the aerostructural system. The inviscid flow around the airfoil is modelled using the Euler equations. They are discretized through the vertex-centered, finite-volume method and solved by the in-house, GPU-enabled, CFD solver PUMA [3]. An unstructured, triangular mesh is generated around the airfoil, consisting of 7870 nodes and 15489 cells. A close-view of the mesh around the airfoil is seen in fig. 4.2.

The aerostructural system includes two disciplines, namely aerodynamics and structures, where the latter is nothing else but the spring model. The input of aerodynamics is the structural angle θ, used to compute the aerodynamic moment M. The spring model computes the structural angle for a given moment, therefore its input is M and its output is θ. The two disciplines are hence strongly coupled, with the coupling variables being θ and M. The XDSM of the airfoil-spring system is shown in fig. 4.3.

Technically, before every call to the aerodynamics solver, the mesh should be rotated by θ. In order to avoid the extra computational cost, the flow velocity is simply turned by -θ, effectively achieving the same effect.

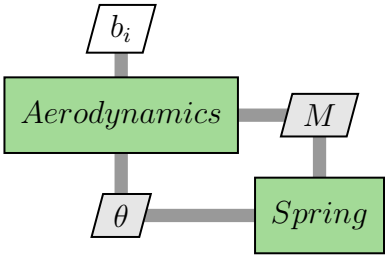


Figure 4.3: Airfoil-spring system: XDSM.

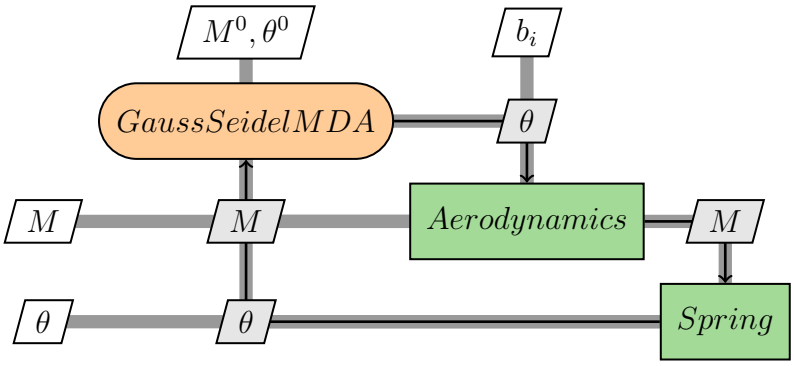


Figure 4.4: Airfoil-spring system: XDSM for the Gauss-Seidel MDA.

4.2 MDA

In order to determine the equilibrium point of the airfoil-spring system, a MDA is performed. The problem is first solved using the Gauss-Seidel approach, the procedure for which is described in algorithm 2. The XDSM for the solution process using the Gauss-Seidel MDA is shown in fig. 4.4.

The spring stiffness J is set to $0.05 N.m/^\circ$ and the inflow velocity U_∞ to $90 m/s$. The starting point is $\theta = 9.1^\circ$ and $M = 0.1\text{N.m}/^\circ$, and no relaxation is used. The termination tolerance of the MDA is set to 10^{-4}. The evolution of the residual metric and the coupling variables during the MDA iterations can be seen in figures 4.5 and 4.6. In total, the MDA requires 8 iterations to reach the specified tolerance, requiring 8 calls to the disciplinary solvers. It converges to the point $\theta = 6.45^\circ$ and $M = 0.322\text{N.m}/^\circ$.

The problem is also solved using the Newton's method (in the functional form), as described in algorithm 4. At each MDA iteration each discipline has to be evaluated and differentiated. The aerodynamics discipline has to provide $\frac{\partial M}{\partial \theta}$, while spring must provide $\frac{\partial \theta}{\partial M}$. The former is computed using first-order finite-

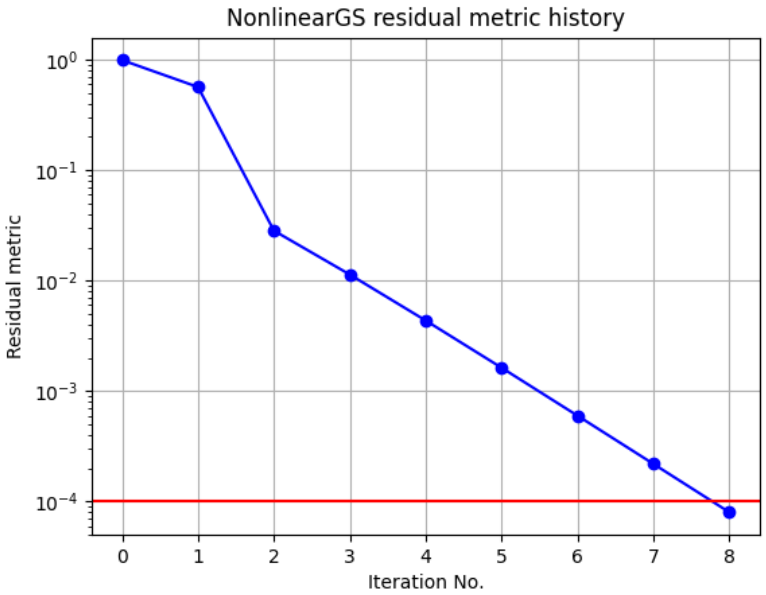


Figure 4.5: Airfoil-spring system: Residual metric evolution for the Gauss-Seidel MDA.

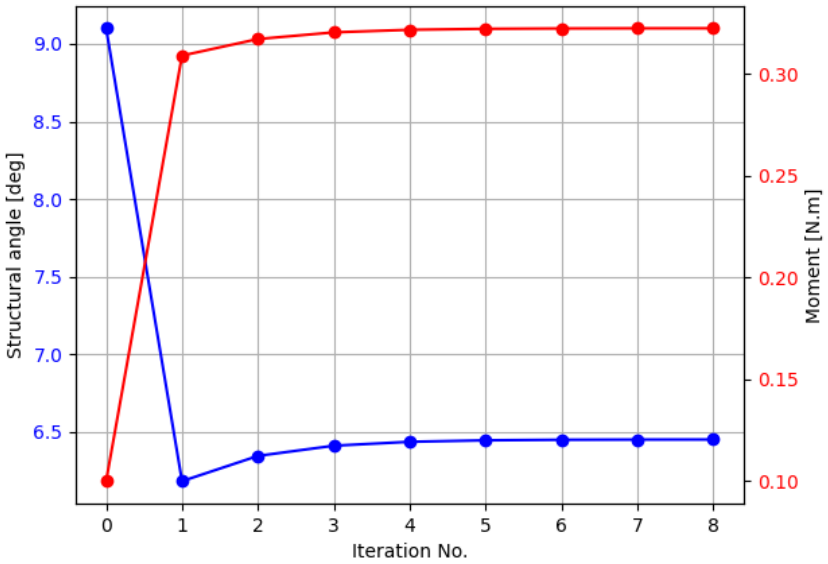


Figure 4.6: Airfoil-spring system: Coupling variables evolution for the Gauss-Seidel MDA.

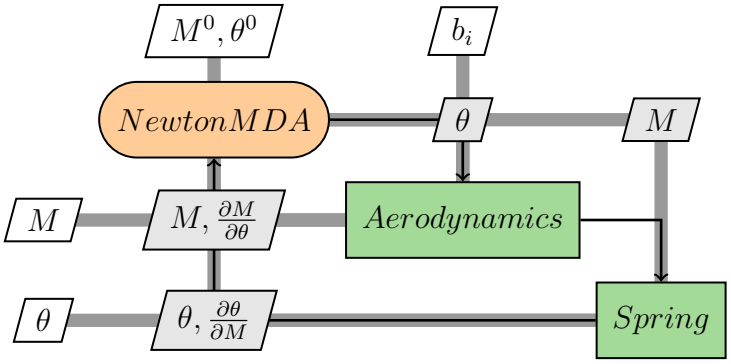


Figure 4.7: Airfoil-spring system: XDSM for the Newton MDA.

Method	Gauss-Seidel	Newton
Iterations	8	5
Aerodynamics evaluations	8	10
Spring evaluations	8	10

Table 4.1: Airfoil-spring system: Performance comparison between Gauss-Seidel MDA and Newton MDA.

differences, while the latter is computed analytically. At the k-th Newton-MDA iteration, the following linear system is solved:

$$\begin{bmatrix} 1 & -\frac{\partial M}{\partial \theta}(\theta^{(k-1)}) \\ -\frac{\partial \theta}{\partial M}(M^{(k-1)}) & 1 \end{bmatrix} \begin{bmatrix} \Delta M^k \\ \Delta \theta^k \end{bmatrix} = \begin{bmatrix} M^{(k-1)} - M(\theta^{(k-1)}) \\ \theta^{(k-1)} - \theta(M^{(k-1)}) \end{bmatrix} \tag{4.4}$$

The XDSM for the solution process using the Newton MDA is shown in fig. 4.7.

The values of the spring stiffness and the inflow velocity, the starting point and the termination tolerance remain the same. The evolution of the residual metric and the coupling variables during the MDA iterations can be seen in figures 4.8 and 4.9. The MDA converges in 5 iterations, requiring 10 calls to each disciplinary solver, or 20 total disciplinary evaluations. The increased cost of the Newton MDA (per iteration) is directly caused by the need to assemble the Jacobian matrix of eq. 4.4, which apart from evaluation, requires differentiation of the disciplines. The MDA converges to the same point as the Gauss-Seidel MDA, namely $\theta = 6.45°$ and $M = 0.322\text{N.m}/°$. A brief performance comparison between the two MDA methods used can be seen in table 4.1.

The results computed by the MDAs are verified by a graphical solution to the system. The curve of the aerodynamic moment M_{aero} as a function of the

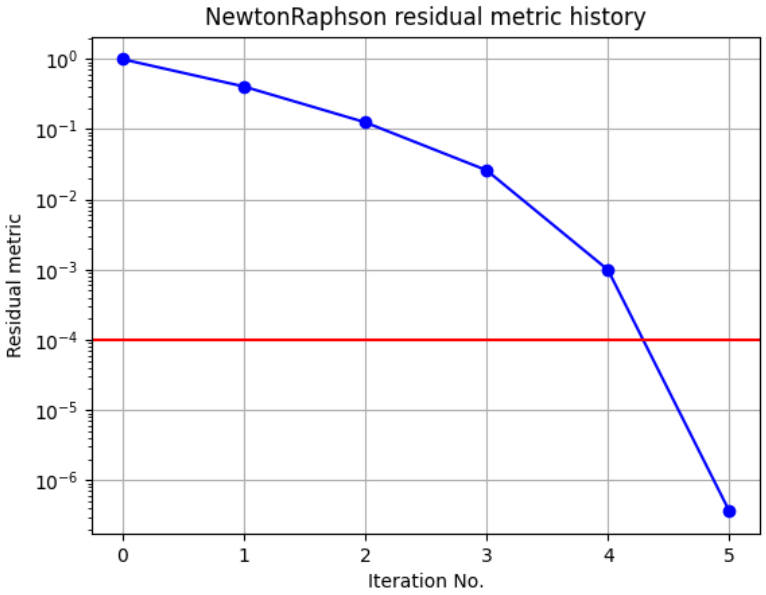


Figure 4.8: Airfoil-spring system: Residual metric evolution for the Newton MDA.

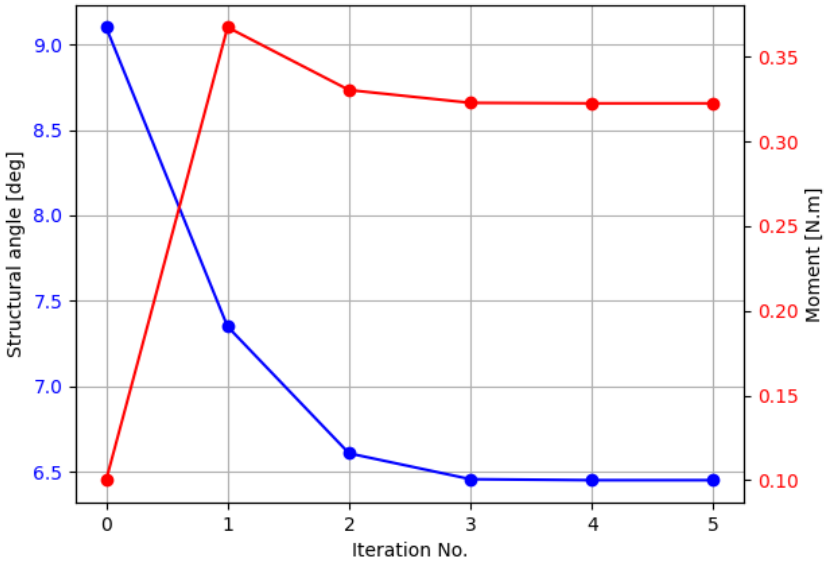


Figure 4.9: Airfoil-spring system: Coupling variables evolution for the Newton MDA.

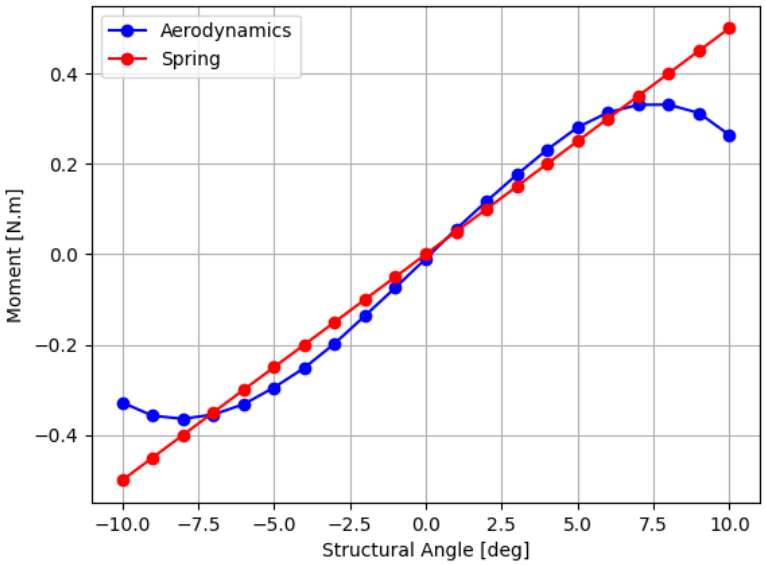


Figure 4.10: Airfoil-spring system: Operating characteristic curves. The points of intersection correspond to equilibrium points, or MDA solutions.

structural angle θ is produced by executing the aerodynamics solver for various values of θ in a range from $-10°$ to $10°$. Similarly, the curve of the spring moment M_{spring} as a function of θ is produced by computing $M_{spring} = J\theta$ for the same θ range. Intersections of the two curves correspond to solutions of eq. 4.2. The curves are shown in fig. 4.10. There exist three solutions, namely $(-6.45, -0.322)$, $(0, 0)$ and $(6.45, 0.322)$. The MDAs converge to the last of the three solutions, to which their starting point is the closest.

4.3 Shape optimization (MDO)

Shape optimization is performed in order to achieve a desired lift value for the airfoil. The objective function is simply:

$$f = 0.5(L - L^*)^2 \tag{4.5}$$

where L and L^* are the actual and desired lift values respectively.

The entire airfoil-spring system is considered, rendering the optimization problem multidisciplinary. The MDF and IDF architectures, implemented in mSense, are used to solve the MDO problem.

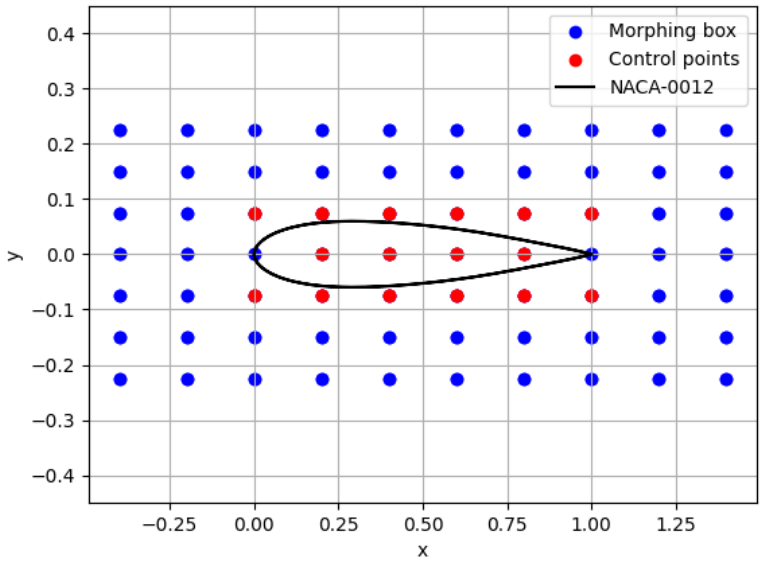


Figure 4.11: Airfoil-spring system: Parameterization of the NACA-0012 airfoil section using volumetric NURBS. The control points, highlighted in red, can move along the y-axis.

Design variable	Size	Lower bound	Baseline Value	Upper bound
b_1	6	-0.1125	-0.075	-0.0375
b_2	4	-0.0375	0.0	0.0375
b_3	6	0.0375	0.075	0.1125

Table 4.2: Airfoil-spring system: Design variables for the airfoil shape optimization problem, along with their dimension, lower and upper bound, and baseline value.

4.3.1 Setup

The airfoil shape is parameterized using volumetric NURBS. The shape is controlled by a 10×7 morphing box. Only 16 out of the 70 points are actually used to manipulate the airfoil's shape. These are the control points of the NURBS, and can move along the y-axis. The parameterization is shown in fig 4.11. The y-coordinates of the 16 control points constitute the design variables of the optimization problem. They are divided into 3 groups, corresponding to the top (b_1), middle (b_2) and bottom (b_3) rows of red points shown in fig 4.11. The points in each group have the same lower and upper bound, and baseline value (fig. 4.2).

For the aerodynamics discipline b_1, b_2 and b_3 are now also inputs. Before each flow analysis (primal solver) the computational mesh has to be adapted to match

the parameterized airfoil. This procedure is performed by PUMA, using Inverse Distance Weighting (IDW). Sensitivities with respect to the design variables b_i are computed by PUMA's continuous adjoint solver. The sensitivity of any output of the aerodynamics discipline with respect to θ is computed using first-order, finite-differences, similar to section 4.2.

The XDSM for optimizing the airfoil-spring system using MDF is shown in fig. 3.2. At each MDF iteration the disciplines are guided to feasibility through the Gauss-Seidel MDA, which is chosen instead of the Newton variant due to the smaller number of disciplinary evaluations required. The termination tolerance is set to 10^{-4} and no relaxation is used. After the MDA is converged, the computation of the coupled derivatives follows.

Aerodynamics has to provide the derivatives of M and f with respect to b_i and θ, namely $\frac{\partial M}{\partial b_i}$, $\frac{\partial f}{\partial b_i}$, $\frac{\partial M}{\partial \theta}$ and $\frac{\partial f}{\partial \theta}$. The spring model only provides the derivative of θ with respect to M, i.e. $\frac{\partial \theta}{\partial M}$. The coupled derivatives of the objective with respect to the design variables, namely $\frac{df}{db_i}$, are computed using the provided disciplinary sensitivity information and the coupled adjoint equations of 2.24 and 2.25. For the sake of clarity, the coupled adjoint method for the airfoil-spring system yields:

$$\frac{df}{db} = \frac{\partial f}{\partial b} - \begin{bmatrix}\phi_1 & \phi_2\end{bmatrix} \begin{bmatrix}\frac{\partial M}{\partial b} \\ \frac{\partial \theta}{\partial b}\end{bmatrix}$$

$$\begin{bmatrix}1 & -\frac{\partial M}{\partial \theta} \\ -\frac{\partial \theta}{\partial M} & 1\end{bmatrix} \begin{bmatrix}\phi_1 \\ \phi_2\end{bmatrix} = \begin{bmatrix}\frac{\partial f}{\partial M} \\ \frac{\partial f}{\partial \theta}\end{bmatrix}$$

where b is the concatenated vector of b_i, i.e. $b = \begin{bmatrix}b_1 & b_2 & b_3\end{bmatrix}^\top$. Therefore, the computation of the coupled derivatives requires the solution of a 2×2 linear system at each optimization cycle.

IDF adds target variables and feasibility constraints for θ and M. The target variables are denoted by θ^t and M^t, while the feasibility constraints are $h^t_\theta = \theta^t - \theta = 0$ and $h^t_M = M^t - M = 0$. Aerodynamics must provide the sensitivities of f and h^t_M with respect to b_i and θ, which are $\frac{df}{b_i}$, $\frac{df}{\theta}$ and $\frac{dh^t_M}{db_i}$, $\frac{dh^t_M}{d\theta}$, while the spring model has to provide $\frac{dh^t_\theta}{dM^t}$. The derivative of any feasibility constraint can be computed easily through the chain rule. For example, $\frac{dh^t_M}{db_i}$ is computed as $\frac{dh^t_M}{db_i} = -\frac{dM}{db_i}$, where $\frac{dM}{db_i}$ is computed by the adjoint solver of PUMA. The XDSM for IDF is shown in fig. 4.13.

4.3.2 Results

For both the MDF and IDF architectures $SLSQP$ is chosen as the optimizer, with the termination tolerance set to 10^{-6}. The spring stiffness J is set to $0.7N.m/^\circ$ and the freestream velocity U_∞ to $90m/s$. The initial values of b_1, b_2 and b_3 are

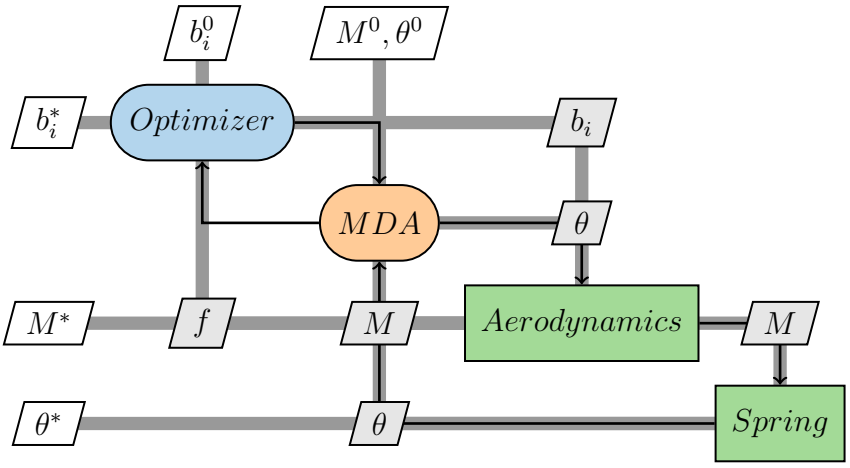


Figure 4.12: Airfoil-spring system: XDSM for the solution of the shape optimization problem using MDF.

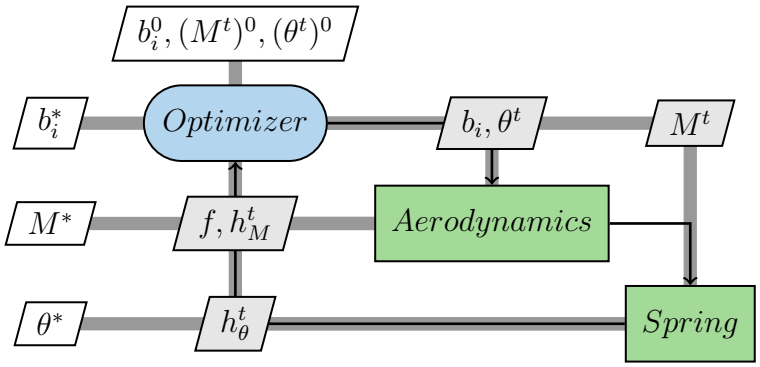


Figure 4.13: Airfoil-spring system: XDSM for the solution of the shape optimization problem using IDF.

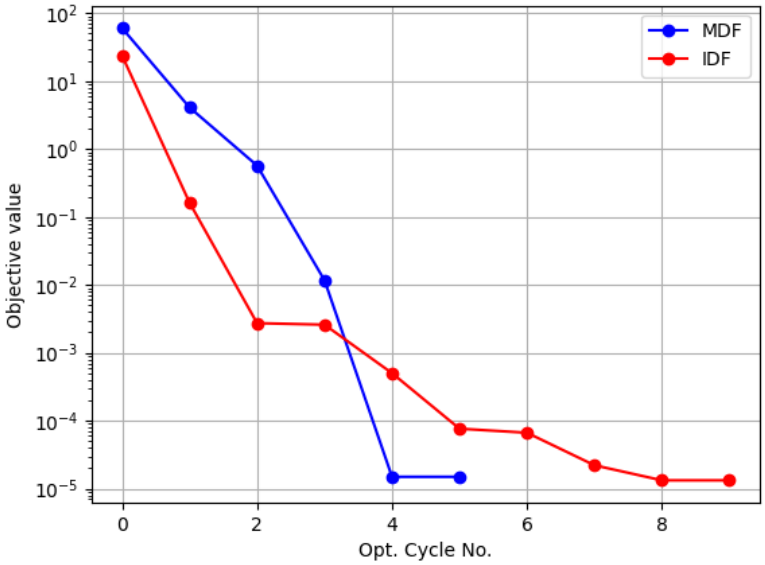


Figure 4.14: Airfoil-spring system: Evolution of the objective function value for the shape optimization problem. Comparison between the MDF and IDF architectures.

found in table 4.2. The starting point of the coupling variables is $\theta = 6.45°$ and $M = 0.322N.m/°$, which is no longer a feasible point, since the stiffness value J has changed. The evolution of the objective function value for both architectures is shown in fig. 4.14, while the evolution of IDF's h^t_θ and h^t_M feasibility constraints is shown in fig. 4.15. Despite the starting point being infeasible, IDF manages to converge the solution to multidisciplinary feasibility. The number of calls to the aerodynamics primal and adjoint solvers, as well as the number of total SLSQP iterations for each architecture are found in table 4.3. Both architectures are able to converge to the solution. In order to reach the same level of convergence, IDF requires 3 less calls to the primal solver than MDF, but 4 more calls to the adjoint solver and 4 more optimizer iterations. Therefore, MDF performs better for this problem, requiring a lower computational cost overall. This contrasts the results obtained from the simpler benchmark problems of chapter 3, which can perhaps be attributed to the more numerically complicated nature of the problem itself, combined with the fact that, in aerodynamics, derivatives w.r.t θ ($\frac{\partial M}{\partial \theta}$ and $\frac{\partial f}{\partial \theta}$) are approximated using finite-differences. The inaccuracies caused by this might be handled better by MDF than IDF, leading to the performance difference.

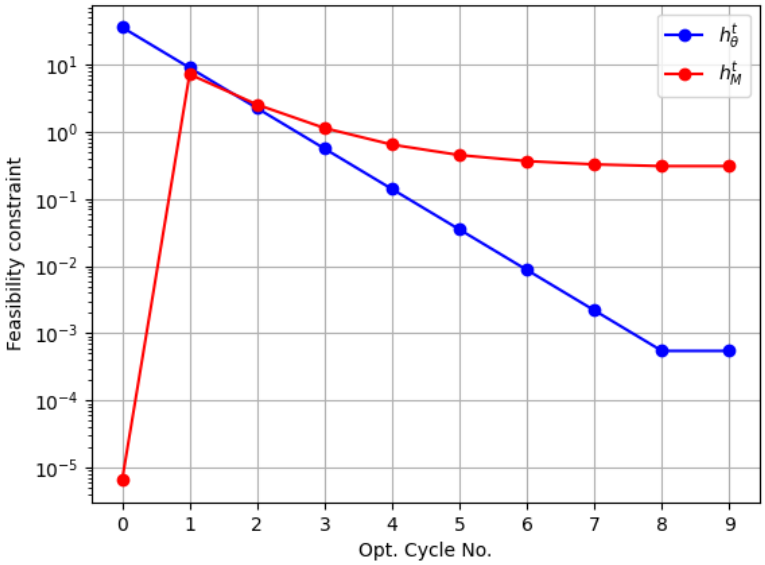


Figure 4.15: Airfoil-spring system: Evolution of IDF's feasibility constraints.

Architecture	MDF	IDF
Primal calls	30	27
Adjoint calls	5	9
Optimization cycles	5	9

Table 4.3: Airfoil-spring system: Performance comparison between the MDF and IDF architectures for the shape optimization problem. MDF requires more primal solver calls than IDF, but less adjoint solver calls and less optimization cycles.

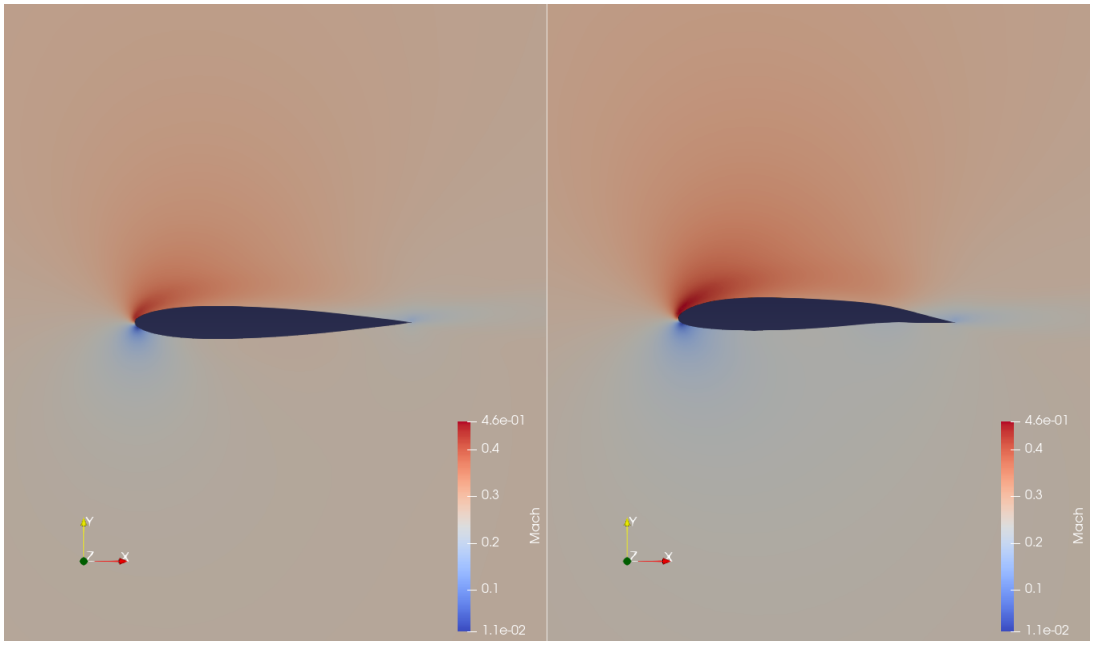


Figure 4.16: Airfoil-spring system: Comparison of the Mach number field around the baseline (left) and optimized (right) airfoils. Instead of rotating the airfoil, the angle of attack α is adjusted, so that the same (aerodynamic) effect is achieved.

Chapter 5

Elastic Tube FSI

5.1 Problem description

Fluid flows through an elastic tube of internal and external radius $R_i = 0.2m$ and $R_o = 0.3m$ respectively, and length $L = 0.6m$. The flow is considered laminar, the fluid density is $\rho_f = 10^3 \frac{kg}{m^3}$ and its kinematic viscosity $\nu_f = 0.07 \frac{m^2}{s}$. The solid outer wall of the tube is modelled as an linear-elastic material with Young's modulus $E = 10^6 Pa$ and a Poisson's ratio of $\nu = 0.2$. The fluid enters with an inlet static pressure of $P_{inlet} = 11500Pa$ and exits with $P_{outlet} = 10000Pa$. The lower and upper ends of the elastic wall are fixed in all directions, while the left end is under the fluid's pressure and the right end is free to deform. The problem is symmetric along the tube's center-line, therefore only the right-half is modelled. The domain is visualized in fig. 5.1.

The flow analysis is performed using PUMA, which was introduced in chapter 4. The incompressible Navier-Stokes equations are solved. At the inlet and outlet static pressure boundary conditions are imposed. At the FSI interface the no-slip condition is enforced, while the left-most boundary is considered symmetric. The linear-elastic, plane-stress problem is solved using SFEM, which is an MPI-enabled finite-element code developed by the author. As previously described, the upper and lower ends of the elastic domain are fixed in all directions, the right end is free to deform and at the interface the solid is subjected to the fluid's pressure. A structured, quadrilateral mesh is used for the fluid domain, with inflation layers used normal to the fluid-solid interface, in order to resolve the laminar boundary layer. The solid domain is meshed using triangular elements. The mesh is matching at the interface, meaning that the fluid and solid nodes match one-to-one. The fluid mesh consists of 7200 nodes and 7433 quadrilaterals, while the solid mesh is comprised of 6690 nodes and 13378 triangles. The mesh is visualized in fig. 5.2.

The FSI model includes two disciplines, these being the flow and structural

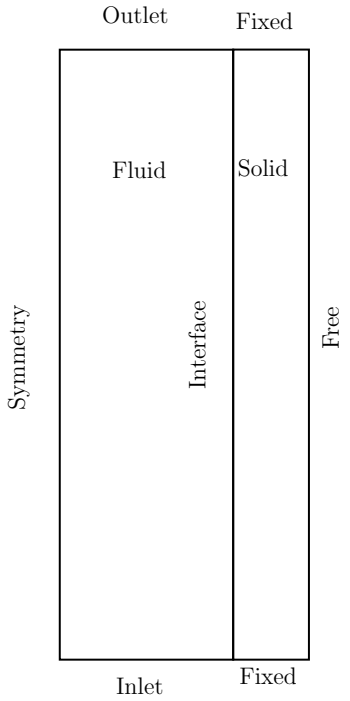


Figure 5.1: Elastic tube FSI: The 2-dimensional FSI domain.

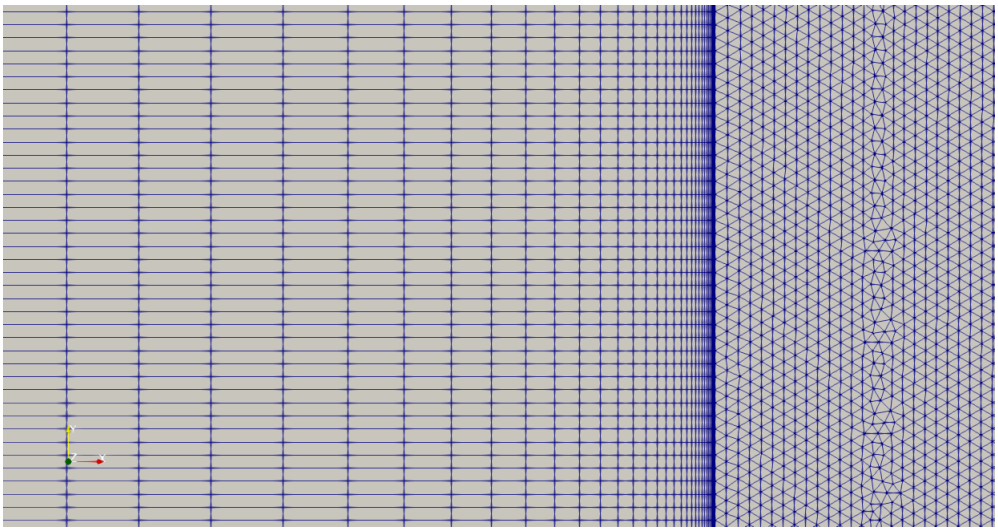


Figure 5.2: Elastic tube FSI: A close view of the mesh at the fluid-solid interface.

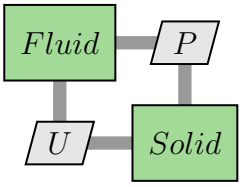


Figure 5.3: Elastic tube FSI: XDSM.

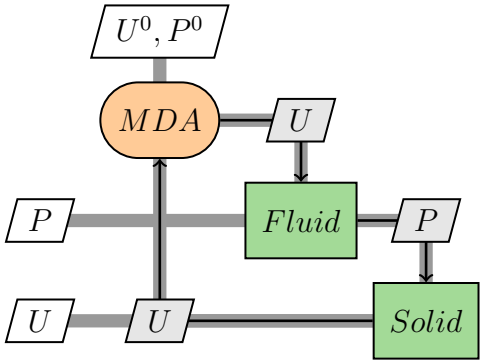


Figure 5.4: Elastic tube FSI: XDSM for the Gauss-Seidel MDA.

solvers. The solvers are coupled through the fluid-solid interface, at which the fluid's pressure causes the solid to deform. The coupling variables are hence the fluid pressure P and the elastic displacements $U = \begin{bmatrix} u_x & u_y \end{bmatrix}$ at the interface. Since the fluid and solid meshes are matching at the interface, no mapping is required to transfer the pressure loads or the displacements from one mesh to the other. For a given U, the fluid solver (PUMA) first deforms its mesh accordingly, then solves the flow equations, and finally outputs the pressure P at the interface. Similarly, the solid solver (SFEM) solves the linear-elastic equations for a given P, and produces U at the interface. The XDSM for the FSI model is shown in fig. 5.3.

5.2 MDA

The FSI model reaches equilibrium by means of the Gauss-Seidel MDA (presented in chapter 2.1.1. The MDA termination tolerance is set to 10^{-4} and no relaxation is used. The corresponding XDSM is shown in fig. 5.4.

The MDA converges in 3 iterations, and the residual history is shown in fig. 5.5. The final, deformed FSI domain is visualized in fig. 5.6. The fluid domain is coloured by the vertical velocity magnitude, while the solid domain is colored by the horizontal displacement. The maximum horizontal displacement is about

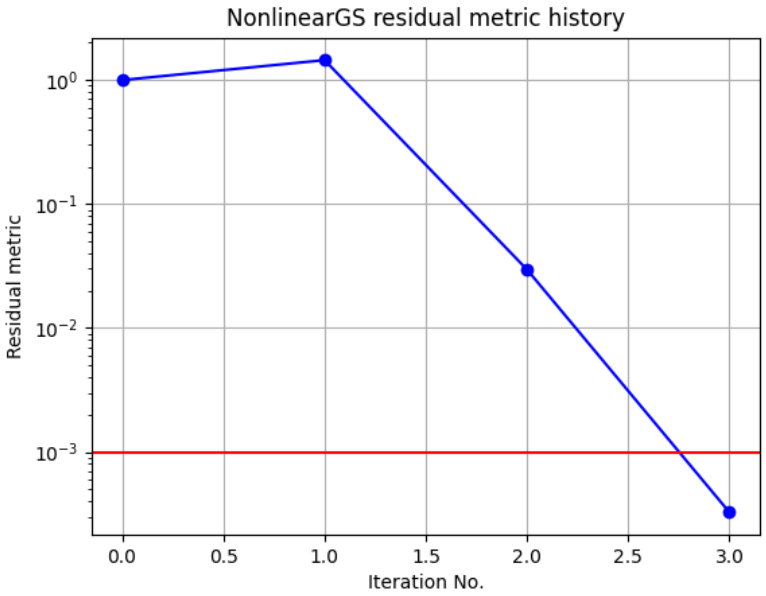


Figure 5.5: Elastic tube FSI: Residual metric evolution for the Gauss-Seidel MDA.

$U_{x,max} = 0.057m$, while the maximum vertical velocity is about $V_{y,max} = 0.98\frac{m}{s}$.

5.3 Material Property optimization (MDO)

It is possible to control the maximum horizontal displacement $U_{x,max}$ of the elastic tube wall by changing the value of the Young's modulus E of the solid. Increasing E increases the stiffness of the solid wall, therefore reducing the maximum displacement. For a desired value of the displacement $U^*_{x,max}$, this process can be formulated as a minimization problem where the objective function is:

$$f = 0.5(U_{x,max} - U^*_{x,max})^2 \tag{5.1}$$

The solid domain is divided into 4 equally-sized regions lengthwise, each one having a separate value of E. The regions are visualized in fig. 5.7.

The resulting optimization problem is solved using the MDF approach. At each iteration feasibility is ensured through the Gauss-Seidel MDA, for which the termination tolerance is set to 10^{-3} and no relaxation is used. The coupled sensitivities of the objective f w.r.t the design variables E_i are obtained using finite-differences on the MDA level, meaning that for each perturbed design variable a separate MDA is needed. This brings the cost of each optimization cycle to $n_X + 1$ MDAs, where n_X is the number of design variables (here $n_X = 4$). Although this approach

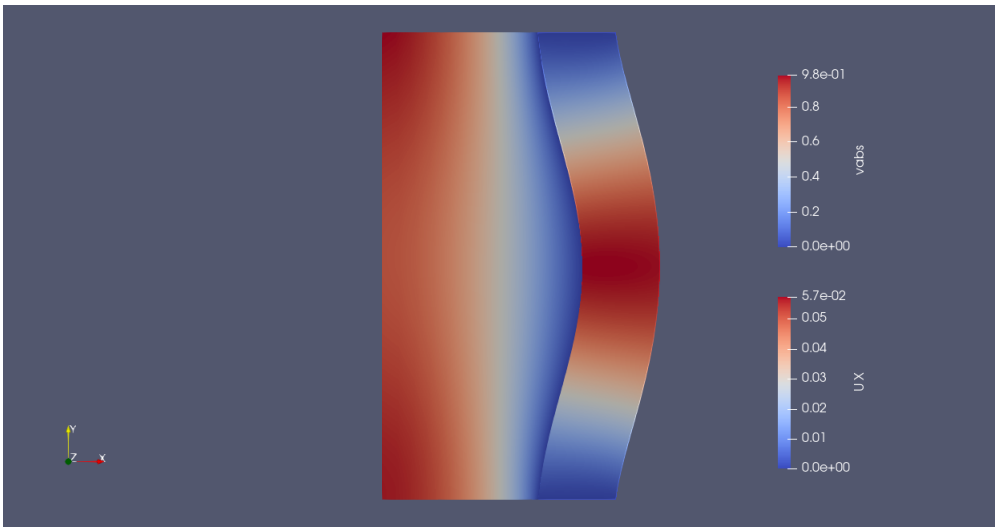


Figure 5.6: Elastic tube FSI: The deformed FSI domain, resulting from the MDA. The fluid domain is colored by the fluid's vertical velocity, while the solid domain is colored by horizontal displacement.

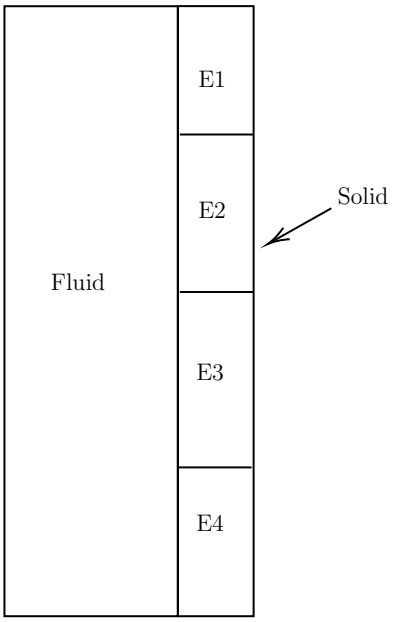


Figure 5.7: Elastic tube FSI: The solid domain is divided into 4 equal-sized regions.

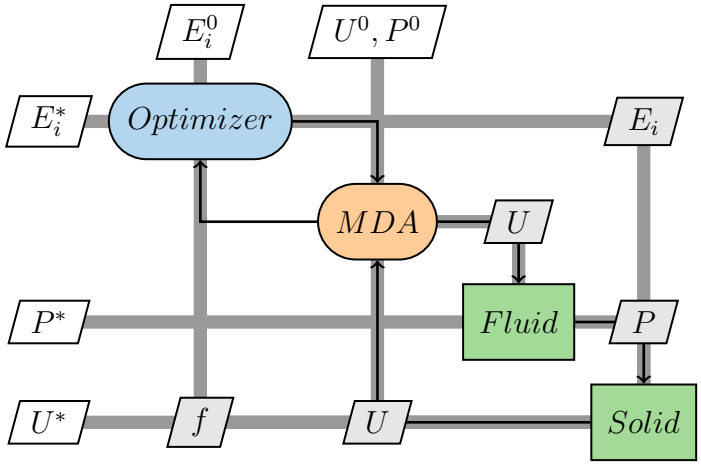


Figure 5.8: Elastic tube FSI: XDSM for the solution of the elastic tube's material property optimization problem using MDF.

Region	Baseline	Optimized
1	10^6	1897544
2	10^6	1898094
3	10^6	1900584
4	10^6	1907181

Table 5.1: Elastic tube FSI: Baseline and optimized values for the Young's modulus (in Pa) for each region.

suffers in both accuracy and computational cost, it is chosen here due to ease of implementation. SLSQP is chosen as the optimizer, and its termination tolerance is set to 10^{-6}. The XDSM for MDF for the FSI system is shown in fig. 5.8.

The baseline value for all E_i is $10^6 Pa$. The desired value of the maximum horizontal displacement is $U^*_{x,max} = 0.03m$. The optimization converges in 4 cycles. The evolution of $U_{x,max}$ during the MDF optimization cycles is shown in fig. 5.9. The final values of E_i are shown in table 5.1. All E_i converge to about $190000 Pa$.

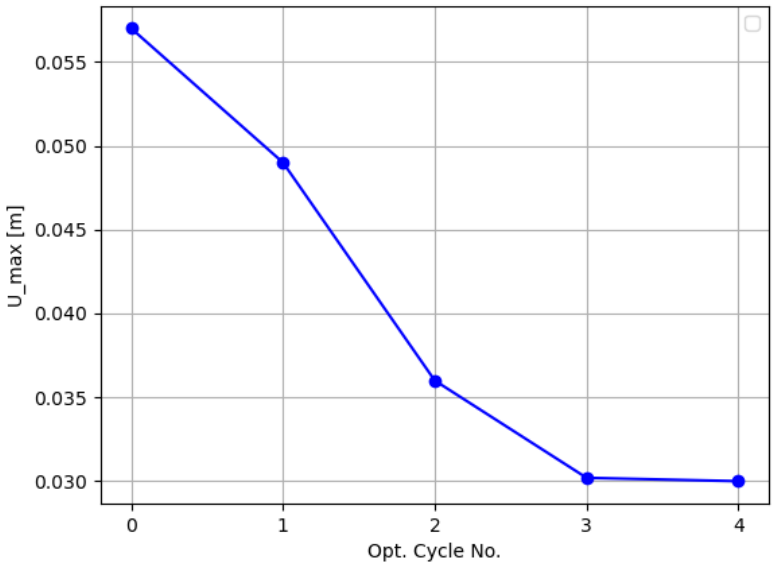


Figure 5.9: Elastic tube FSI: Evolution of the maximum horizontal displacement during the MDF cycles, for the elastic tube's material property optimization problem. The desired value is $U^*_{x,max} = 0.03m$.

Chapter 6

Aerostructural optimization of the ONERA M6 wing

The aerostructural optimization of aircraft wings is one of the most common use cases of MDAO. Here, the ONERA M6 wing is considered, which is a popular benchmark for CFD codes. Because no standard structural model exists, a simple finite element model is constructed, consisting of beam elements along the span of the wing. First, using an MDA the equilibrium of the aerostructural system is computed. Then, using MDO, certain characteristics of the wing, such as the drag it produces, are improved.

6.1 Problem description

The ONERA M6 wing is placed inside of a three-dimensional, inviscid flow field. At flow conditions of Mach $M = 0.84$ and an angle of attack equal to 0 ($\alpha = 0°$), the flow is transonic, producing a shock-wave near the leading-edge of the wing. The wing is not rigid, and deforms due to the forces acting on it. These are the aerodynamic forces, computed by integration of the pressure on the surface of the wing, and the wing's own weight. The wing is cantilevered, meaning that it is fixed at the root but free to move in any direction at the tip. In order to make the MDAO problem meaningful, it is considered that the wing is a part of an aircraft configuration, and therefore has to carry the weight of the aircraft. This is used when setting up the optimization problem.

6.1.1 Aerodynamic model

The ONERA M6 is wing is a swept, semi-span wing with no twist. It uses the symmetric ONERA D airfoil section. The wing has a span of 1.1963 meters, a

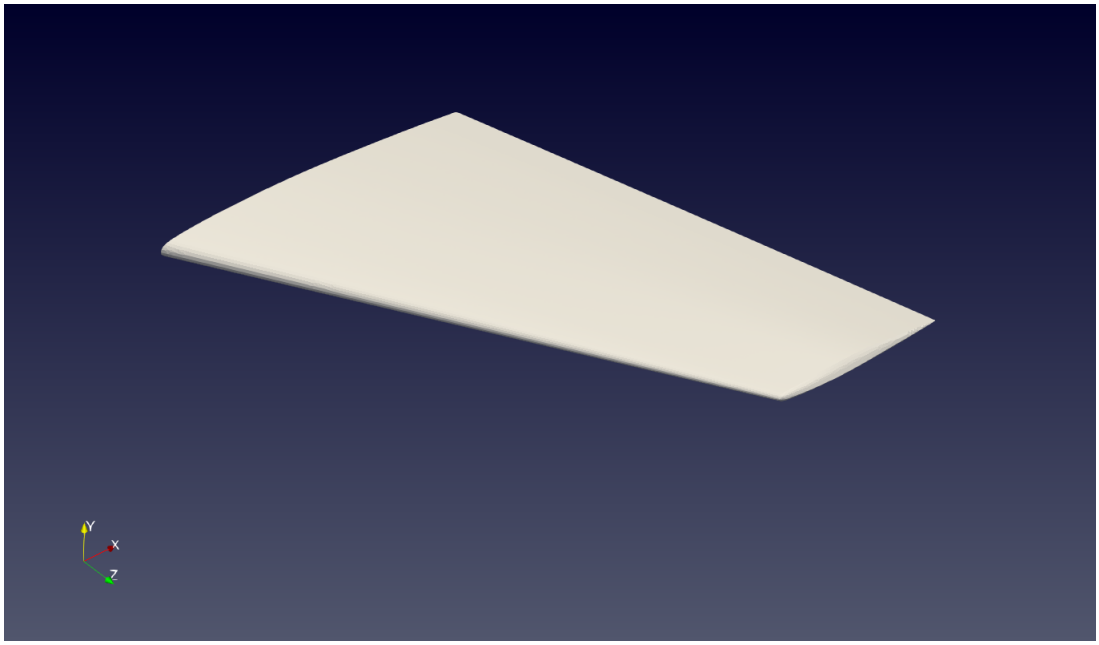


Figure 6.1: ONERA M6: CAD model of the wing.

mean aerodynamic chord of 0.64607 meters, an aspect ratio of 3.8, a taper ratio of 0.562, a leading-edge sweep angle of 30° and a trailing-edge sweep angle of 18°. The geometry of the wing is shown in fig. 6.1. Wind tunnel tests concerned with the flow over the ONERA M6 were conducted by Schmitt and Charpin [31]. The tests were performed for various transonic Mach numbers and angles of attack. The results of [31] are widely used to validate CFD codes.

The flow around the wing is modelled using the Euler equations. The in-house, GPU enabled CFD solver PUMA, introduced in chapter 4, is used on an unstructured mesh of 72791 nodes and 341797 cells. The surface mesh of the wing is shown in fig 6.2.

The wing's surface is parameterized using volumetric NURBS. There exist 280 control points in total, of which only 18 are updated during optimization, while the rest remain unchanged. In fig. 6.3, the points in blue correspond to the fixed control points, while the red points are free to move in the span-wise and flap-wise directions. Since the 18 control points can move in two directions, this results in 36 shape parameters or design variables for optimization.

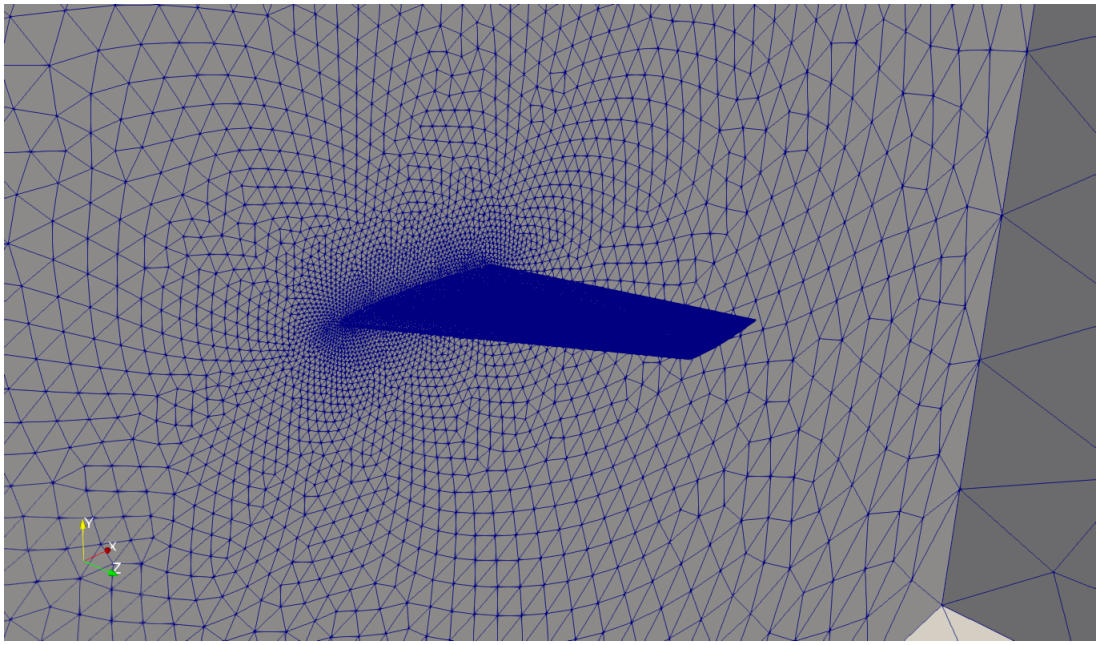


Figure 6.2: ONERA M6: Mesh around the wing.

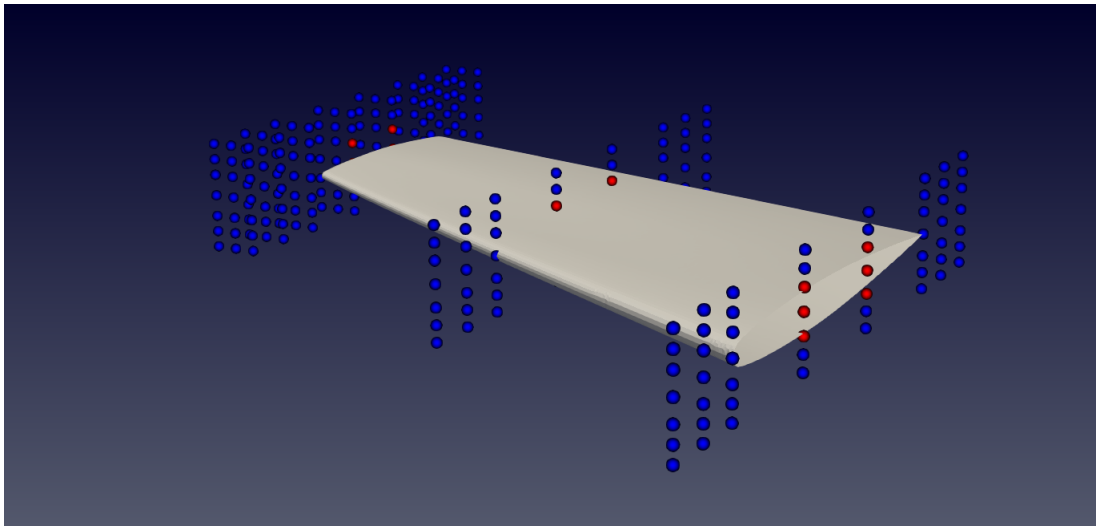


Figure 6.3: ONERA M6: The (baseline) volumetric NURBS parameterization of the wing. The points in blue are the control points for the NURBS. The points in red are the active control points, and they are able to move in the span-wise and flap-wise directions.

6.1.2 Structural model

In order to use the ONERA M6 wing for aerostructural MDAO, a structural model must first be developed. A simple but effective approach of modelling the elastic behaviour of the wing under aerodynamic loading, is to consider a beam model which captures the bending of the wing in the span-wise direction. The model is constructed from one-dimensional beam finite elements. Each element has 2 nodes and 2 DoF per node. For a node i its two DoF correspond to the vertical displacement v_i and rotation ω_i. A single beam element is shown in fig. 6.4. The FEM model is implemented in SFEM, which is finite-element code developed by the author and introduced in chapter 5.

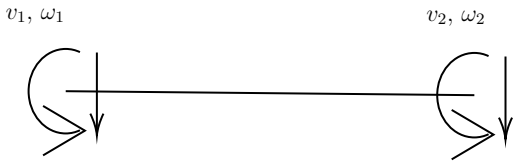


Figure 6.4: ONERA M6 structural model: Schematic of a single beam element.

The total deflection inside each element is interpolated as follows:

$$v(s) = \begin{bmatrix} N_1(s) & N_2(s) & N_3(s) & N_4(s) \end{bmatrix} \begin{bmatrix} v_1 & \omega_1 & v_2 & \omega_2 \end{bmatrix}^T \tag{6.1}$$

In the above equation, N_i are the element shape functions, which are formulated as:

$$\begin{aligned} N_1(s) &= 1 - 3s^2 + 2s^3 \\ N_2(s) &= L(s - 2s^2 + s^3) \\ N_3(s) &= 3s^2 - 2s^3 \\ N_4(s) &= L(-s^2 + s^3) \end{aligned} \tag{6.2}$$

where L is the length of each element. The normalized coordinate s is simply $\frac{x}{L}$, where x is a coordinate along the element's length. The element stiffness matrix is:

$$K_e = \frac{EI}{L^3} \begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ -6L & -4L^2 & 6L & -2L^2 \end{bmatrix} \tag{6.3}$$

where E is Young's modulus and I is the moment of inertia of each element. The bending moment inside each element is computed as:

Symbol	Description	Value	Units
E	Young's modulus	$3.5 * 10^7$	Pa
σ_{yield}	Yield stress	$4 * 10^6$	Pa
N	Safety factor	2	-
σ_d	Design stress	$2 * 10^6$	Pa
ρ	Density	2710	kg/m^3
R_i	Outer radius	0.2	m
t_i^0	Initial thickness	0.03	m

Table 6.1: ONERA M6 structural model: Material and geometrical properties of the beam elements.

$$M_b(s) = EI\frac{d^2\nu}{dx^2} \tag{6.4}$$

From the bending moment, the bending stress is easily calculated as:

$$\sigma_b(s) = \frac{c}{I}M_b(s) \tag{6.5}$$

where c is the distance between the furthest point on the elements cross-section and its bending-axis. Each element i has a hollow, cylindrical cross-section, of outer radius R_i and thickness t_i. Therefore, $c = R_i$. The material properties and geometric properties of the elements are found in table 6.1. Although the radius of each element is fixed, its thickness is allowed to change during optimization.

In total, 5 elements with 6 nodes total are used, resulting in 12 DoF. The DoF corresponding to the node closest to the root of the wing are fixed, i.e. they cannot move in any direction, so the total number of DoF is reduced to 10. The structural model is visualized in fig. 6.5.

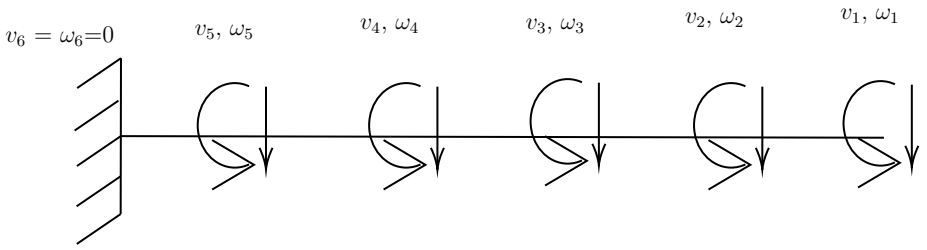


Figure 6.5: ONERA M6 structural model: Schematic of the structural model. The leftmost (closest to the wing's root) node and its corresponding DoF are fixed. The model computes the bending of the wing.

Unlike the displacements ν_i and rotations ω_i which are computed for each node, the stresses $\sigma_{b,i}$ are computed for each element. It is important that, for

each element, the stress $\sigma_{b,i}$ does not exceed the allowed limit, or design stress σ_d. The design stress is equal to the yield stress σ_{yield}, divided with a safety factor N, i.e. $\sigma_d = \frac{\sigma_{yield}}{N}$. For optimization, this corresponds to the set of inequality constraints $\sigma_{b,i} \leq \sigma_d$. In order to reduce the dimensionality of the constraint, the discrete Kreisselmeier-Steinhauser (KS) function [21] is used:

$$\sigma_{DKS} = \frac{1}{\rho} \log(\sum_{i=0}^{n_{Elements}} e^{\rho \sigma_{b,i}}) \tag{6.6}$$

The constraint now becomes $\sigma_{DKS} \leq \sigma_d$. Hence, by using the KS function only a single structural constraint is required, instead of having one constraint for each element.

Each element is subject to an aerodynamic force and its own weight. The aerodynamic force is computed by integrating the surface pressure. In order to assign an aerodynamic force value to each element, the integration is done in patches. The patches are equally spaced along the span of the wing, each one roughly corresponding to each element. This is showcased in fig. 6.6.

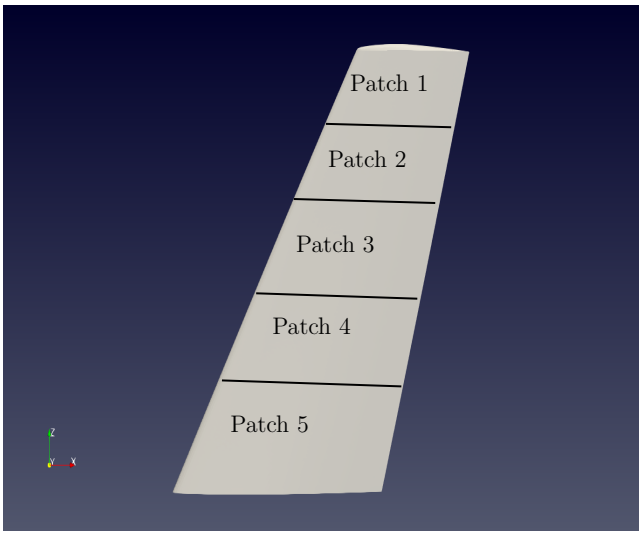


Figure 6.6: ONERA M6 structural model: The patches used for integration of the surface pressure on the wing. The number of patches is equal to the number of beam elements. Each patch computes the aerodynamic force for its corresponding element.

6.1.3 Aircraft configuration

As it was previously mentioned, it is considered that the ONERA M6 wing is part of a theoretical aircraft configuration, and must therefore carry its weight. This is done with the aim of making the MDAO problem more realistic, without further increasing the computational cost by analysing a whole aircraft.

The total weight of the theoretical aircraft W_{total} is the sum of its empty weight W_{empty}, its payload weight W_{paylod} and its fuel weight W_{fuel}. The empty weight of the aircraft is taken to be almost equal to its structural weight, which is made up from the weight of the wing W_{wing}, the fuselage $W_{fuselage}$ and the tail W_{tail}. In short, the weight of the aircraft can be computed from the following expressions:

$$\begin{aligned} W_{total} &= W_{empty} + W_{paylod} + W_{fuel} \\ W_{empty} &= W_{wing} + W_{fuselage} + W_{tail} \end{aligned} \tag{6.7}$$

During optimization only the weight of the wing is allowed to vary. Hence, it is convenient to express the W_{total} as the sum of a fixed weight W_{fixed} and the wing weight:

$$\begin{aligned} W_{total} &= W_{fixed} + W_{wing} \\ W_{fixed} &= W_{fuselage} + W_{tail} + W_{paylod} + W_{fuel} \end{aligned} \tag{6.8}$$

For the baseline configuration, it is considered that the wing weight is around 10% of the total weight. Given the material and geometrical properties of the structural model presented in table 6.1, the initial wing weight is $W^0_{wing} = 2218N$. Therefore, the initial total weight is $W^0_{total} = \frac{W^0_{wing}}{0.1} = 22180N$. The fixed weight, which does not change during optimization, is computed as $W_{fixed} = W^0_{total} - W^0_{wing} = 19962N$.

Similar to the weight, the total lift L_{total} and drag D_{total} can be expressed as sums of components:

$$\begin{aligned} L_{total} &= L_{wing} + L_{fuselage} + L_{tail} \\ D_{total} &= D_{wing} + D_{fuselage} + D_{tail} \end{aligned} \tag{6.9}$$

Since only the lift and drag of the wing is known, the values of the other components have to be estimated. For the lift, it is considered that the wing and the fuselage produce 90% and 14% of the total amount respectively, while the tail reduces it by 4%. The baseline ONERA M6, for the conditions described in the previous section, produces a lift of about $10^4 N$. Therefore, the total baseline lift is estimated as follows:

$$L^0_{total} = \frac{2 * 10^4}{0.9} = 22222N \tag{6.10}$$

Quantity	Baseline Value (N)	Constant
W_{wing}	2218	No
W_{fixed}	19962	Yes
W_{total}	22180	No
L_{wing}	20000	No
$L_{fuselage}$	3111	Yes
L_{tail}	-888	Yes
L_{total}	22222	No
D_{wing}	950	No
$D_{fuselage}$	950	Yes
D_{tail}	211	Yes
D_{total}	2111	No

Table 6.2: ONERA M6 aerostructural model: Baseline values of the aircraft's weight, lift and drag values. The last column states whether the quantity remains constant during optimization.

The fuselage and tail lift contributions can now be calculated as:

$$\begin{aligned} L_{fuselage} &= L^0_{fuselage} = 0.14 * L^0_{total} \\ L_{tail} &= L^0_{tail} = -0.04 * L^0_{total} \end{aligned} \tag{6.11}$$

The values of $L_{fuselage}$ and L_{tail} remain constant during optimization, since the geometry of the fuselage and the tail are not changed. A similar process is followed for the drag. It is considered that both the wing and the fuselage contributed each about 45% of the total aircraft drag, while the tail adds the remaining 10%. The baseline drag value for the ONERA M6 is about $475N$. Since the Euler equations are used, this corresponds only to wave drag. The value of the total baseline drag can be calculated as:

$$D^0_{total} = \frac{2 * 475}{0.45} = 2111N \tag{6.12}$$

The fuselage and tail drag contributions are now easily computed:

$$\begin{aligned} D_{fuselage} &= D^0_{fuselage} = 0.45 * D^0_{total} \\ D_{tail} &= D^0_{tail} = 0.1 * D^0_{total} \end{aligned} \tag{6.13}$$

Similar to $L_{fuselage}$ and L_{tail}, the values of $D_{fuselage}$ and D_{tail} remain constant during optimization. The values of all weight, lift and drag components presented above can be found in table 6.2.

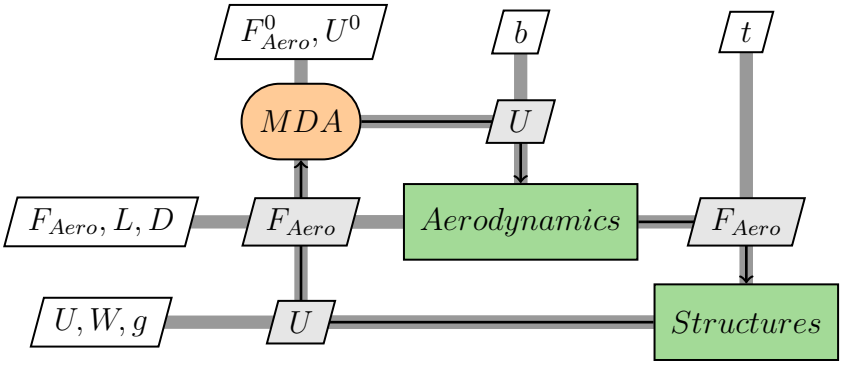


Figure 6.7: ONERA M6 aerostructural MDA: XDSM for the Gauss-Seidel MDA.

6.2 MDA

By performing an MDA for the baseline configuration it is possible to determine the deformation of the wing and the stresses it undergoes while flying. Furthermore, considering that the wing deforms during flight, the drag and lift values also change. The MDA loop begins by executing the aerodynamics solver, PUMA. Before each flow solution, the CFD mesh has to be modified to account for the structural displacements, produced by the FEM solver. After modifying the mesh, the flow equations are solved, and the surface pressure values are obtained. By integrating these, the aerodynamic forces per element, denoted by F_{Aero}, are computed. The FEM solver uses these forces to compute the nodal displacements U. The FEM nodal displacements are then interpolated to the CFD mesh, and the process repeats. The coupling variables are the element aerodynamic forces F_{Aero} (of size $n_{Elements} = 5$) and the nodal displacements U (of size $n_{Nodes} = 6$). F_{Aero} is the output of aerodynamics, while U is the output of the structural solver. This is also shown in the XDSM for the aerostructural MDA (fig. 6.7).

The Gauss-Seidel MDA is used, the procedure for which is described in algorithm 2. The termination tolerance is set to 10^{-4} and no relaxation is used. The evolution of the residual during the MDA iterations is shown in fig. 6.8. The MDA converges in 4 iterations. The total lift L_{total} and drag D_{total}, as well as the aggregated stress σ_{DKS} for the undeformed and deformed baseline wing are listed in table 6.3. Finally, the undeformed and deformed configurations are shown in fig. 6.9.

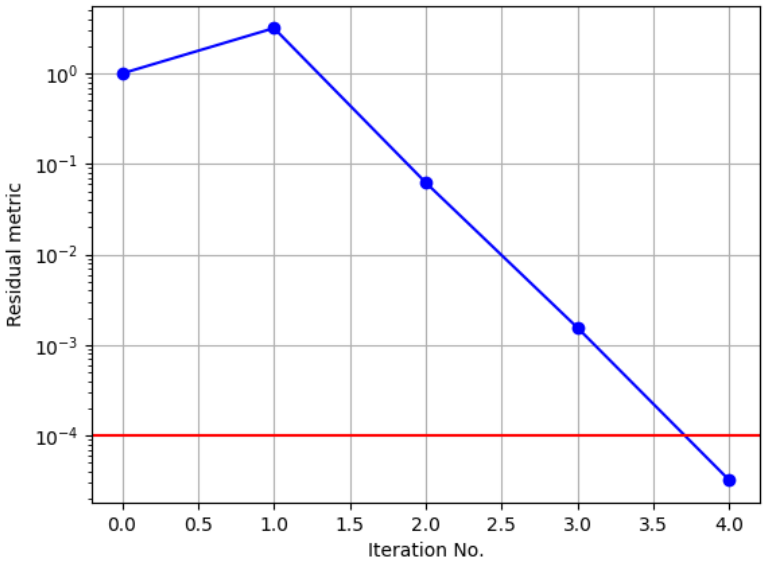


Figure 6.8: ONERA M6 aerostructural MDA: Residual metric evolution for the Gauss-Seidel MDA.

Quantity	Value (undeformed)	Value (deformed)	Unit
L_{total}	22715	23367	N
D_{total}	2112	$2136N$	N
σ_{DKS}	0	1941346	Pa

Table 6.3: ONERA M6 aerostructural MDA: Values of the lift, drag and the KS aggregated stress for the baseline aerostructural model. Comparison between the undeformed and deformed (obtained by the MDA) configurations.

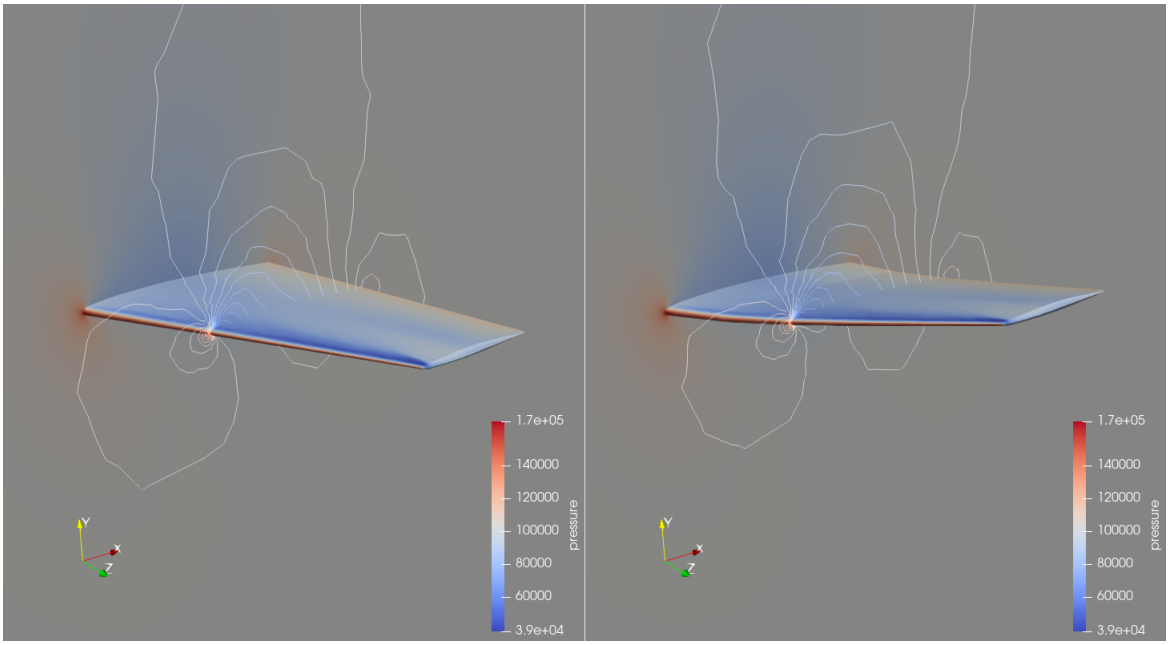


Figure 6.9: ONERA M6 aerostructural MDA: Comparison of the undeformed (left) and deformed (obtained by the MDA, right) configurations for the baseline aerostructural model. The pressure field contours are shown.

6.3 Shape and structural optimization (MDO)

Shape and structural optimization is performed in order to improve certain performance metrics of the wing. The choice of performance metric, namely the objective function, as well as the set of constraints used can significantly affect the result of the optimization. For all cases however, the set of design variables remain the same. These are the shape parameters of the wing, namely the active control points of the volumetric NURBS, denoted by b, and the thickness of each structural element t. In total, there exist $n_X = 42$ design variables, 36 of which correspond to b and 6 of which correspond to t.

A constraint imposed regardless of the choice of objective function is that the stress does not exceed the maximum allowed amount or design stress σ_d. This constraint, denoted by g, is expressed as:

$$g = \frac{\sigma_{DKS}}{\sigma_d} \leq 1 \tag{6.14}$$

Again, the discrete KS stress aggregate is used, resulting in a single structural constraint. A second constraint imposed in all cases is that the total produced lift L_{total} is equal or greater than the total weight W_{total} of the aircraft. This constraint

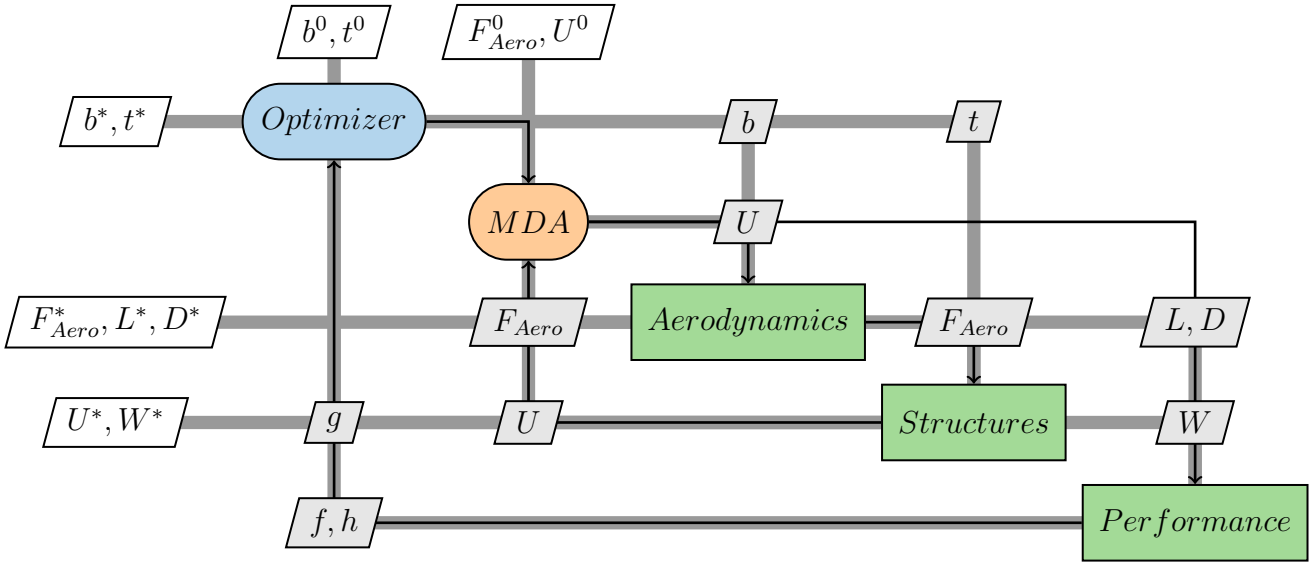


Figure 6.10: ONERA M6 aerostructural MDO: XDSM for the aerostructural MDO, using the MDF architecture.

is denoted by h, and is expressed as:

$$h = L_{total} - W_{total} \geq 0 \tag{6.15}$$

Regardless of the objective function f used, the optimization process is the same. The Multidisciplinary Feasible (MDF) architecture is used. MDF requires an MDA at each optimization cycle. The Gauss-Seidel MDA is used, with a termination tolerance of 10^{-4} and no relaxation. Each optimization cycle begins by adapting the CFD mesh to the current shape parameters b. Then, the MDA process converges the system to equilibrium, as described in the previous section. Each MDA iteration begins by modifying the CFD mesh to account for the structural displacements, then the aerodynamics solver is executed, solving the flow equations and computing F_{Aero}, the (single) wing lift L and drag D. Afterwards, the structural solver computes U and evaluates the structural weight of the (single) wing W and the constraint g. After the MDA converges successfully, a new discipline, called performance, uses L, D and W to evaluate the constraint h and the objective f. The process is showcased in the relevant XDSM (fig. 6.10).

For the computation of the coupled derivatives, the coupled adjoint method is used. There exist $n_X = 42$ design variables in total, while the size of the objective f and the constraints g and h is $n_F = 3$. The disciplinary partial

derivatives of aerodynamics are computed using the adjoint solver of PUMA. For each differentiation a total of $2 + n_{Elements}$ calls to the adjoint solver are required. This is because aerodynamics has to provide the derivatives of its outputs, which are L, D and F_{Aero}, w.r.t its inputs U and b. The structural discipline has to provide the derivatives of U, g and W w.r.t F_{Aero} and t. The newly added third discipline, performance, has to provide the derivatives of the objective f and the constraint h w.r.t the lift L, the drag D and the weight W. The disciplinary derivatives of both the structural and performance disciplines are computed using finite-differences with a step-size of 10^{-6}. For all cases, the SLSQP optimizer is used. The termination tolerance is set to 10^{-6} and a maximum of 10 cycles are allowed.

6.3.1 Results

Two objective functions are used. The first is the weighted sum of the aircraft's drag and structural weight. Reducing the drag has the obvious benefit of improving aerodynamic efficiency. Reducing the structural weight means that either less stress is placed on the wing (due to requiring less lift), or that the aircraft can carry more payload or fuel (for the same lift value). The objective function is mathematically expressed as:

$$f = \frac{D}{D_0} + \frac{W}{W_0} \tag{6.16}$$

The values D_0 and W_0 are used to scale D and W, so that both $\frac{D}{D_0}$ $\frac{W}{W_0}$ have a value near 1. Here, D and W correspond to the drag and weight of the wing, while D_0 and W_0 are the initial wing drag and weight values, found in table 6.2.

The optimization is performed as described previously and summarized in fig. 6.10. The evolution of the wing drag, lift, weight and lift-to-drag ratio is shown in fig. 6.11. Similarly, the evolution of the total aircraft lift and weight, and the stress constraint g is shown in fig. 6.12. The optimized stress and thickness distributions for the wing are shown in fig. 6.13. The optimizer is able to successfully reduce both the drag and weight, by almost 10% and 50% respectively (compared to the initial value). Although the lift is decreased, the lift-to-drag ratio increases, since the drag is reduced by a higher percentage. The reason the lift decreases is due to the weight reduction. In order to reduce the weight, the optimizer had to reduce the thickness of the structural elements, meaning that the stress increased. As a result, the lift also decreases to prevent the stress from exceeding the design limit. However, the difference of total lift and weight, or $L_{total} - W_{total}$ also decreases (fig. 6.12), meaning that at the end of optimization the aircraft is able to carry less extra weight. Obviously, this is a drawback of this objective function.

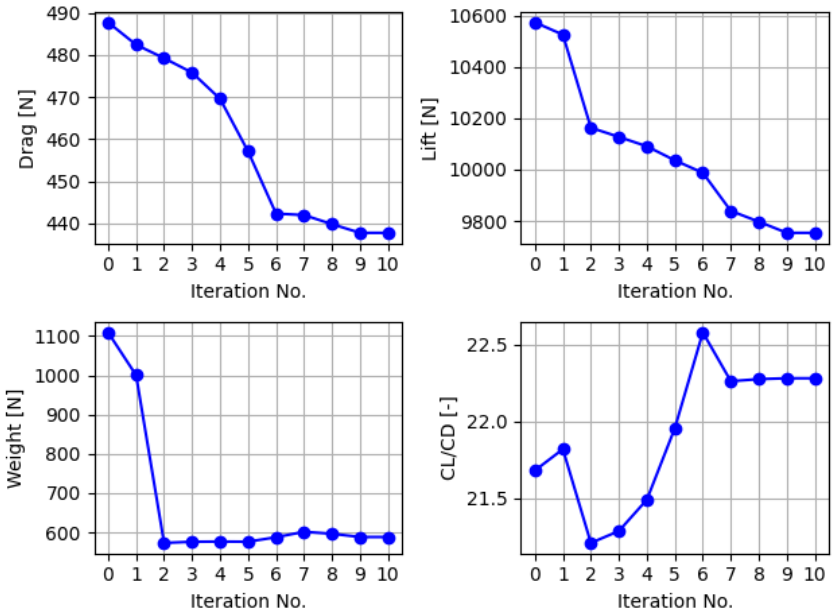


Figure 6.11: ONERA M6 aerostructural MDO: Evolution of the wing drag (upper left), lift (upper right), weight (lower left) and lift-to-drag ratio (lower right). The weighted sum of the drag and weight is used as the objective.

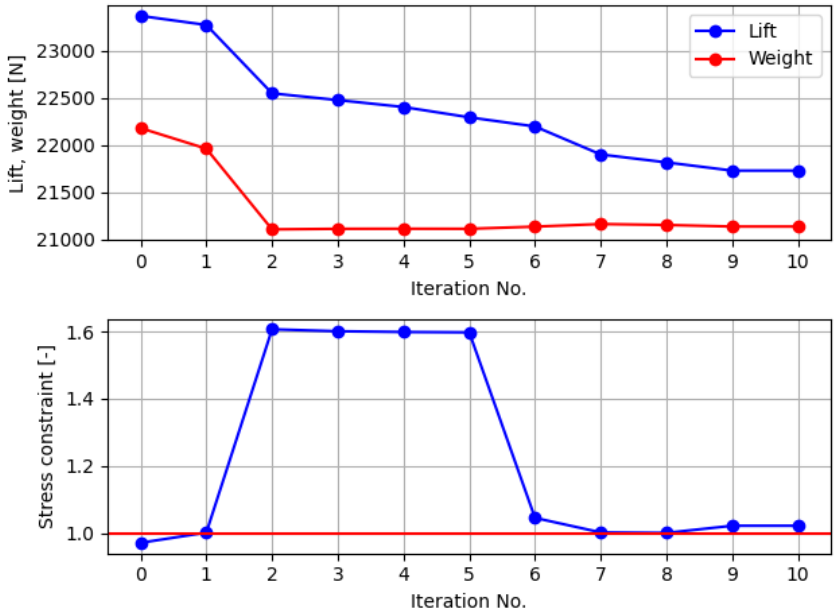


Figure 6.12: ONERA M6 aerostructural MDO: Evolution of the total aircraft lift and weight, and the stress constraint. The weighted sum of the drag and weight is used as the objective.

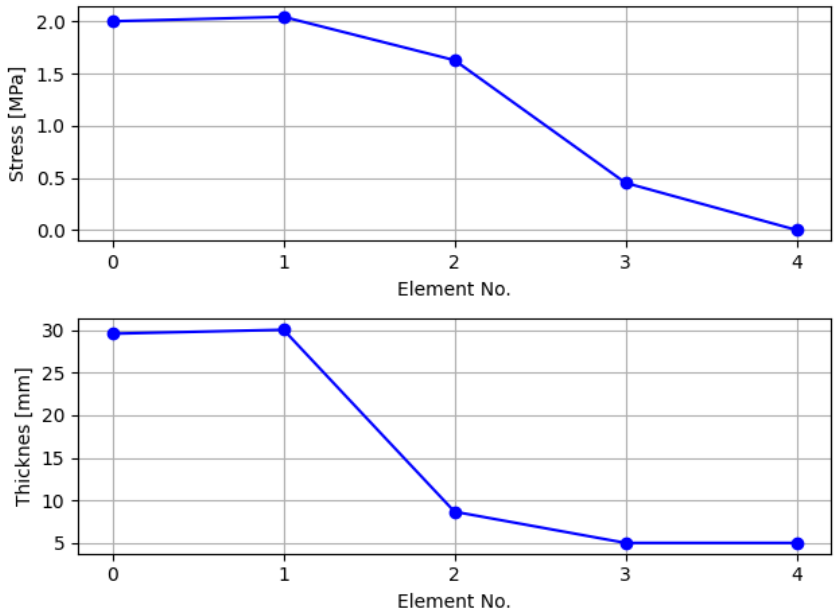


Figure 6.13: ONERA M6 aerostructural MDO: Stress (upper) and thickness (lower) distributions along the wing's span. Element number 0 corresponds to the element closest to the wing's root and 4 to the tip. The weighted sum of the drag and weight is used as the objective.

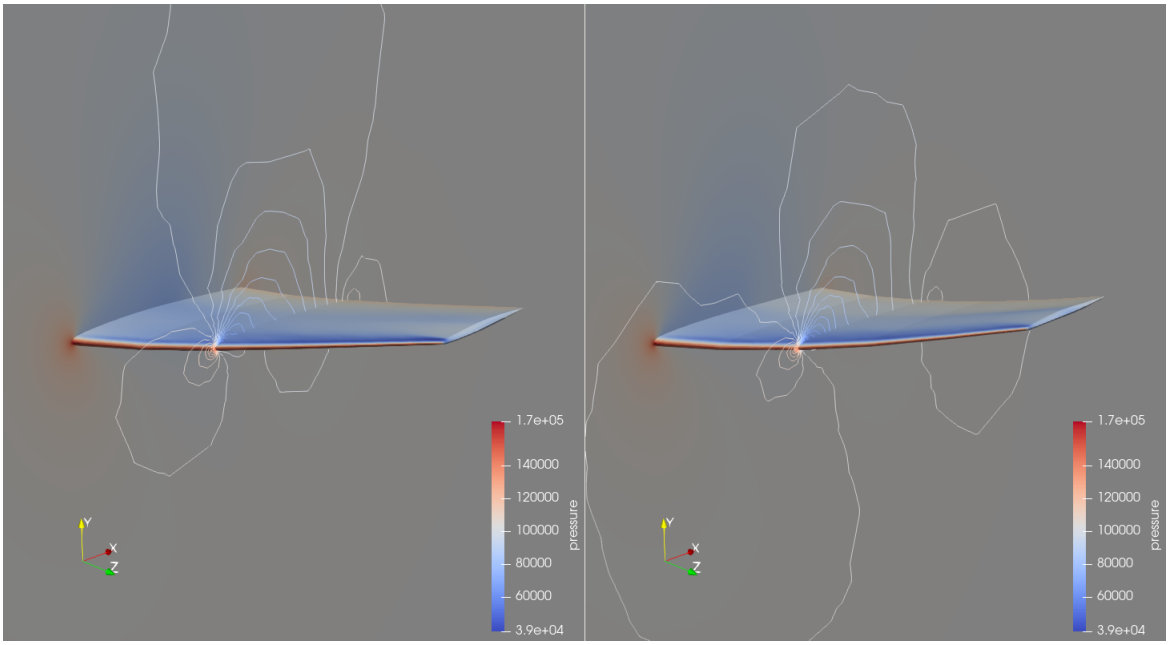


Figure 6.14: ONERA M6 aerostructural MDO: Comparison of the deformed configurations for the baseline (left) and optimized (right) wings. The pressure field contours are shown. The optimized wing is less stiff than the baseline, and bends more. The weighted sum of the drag and weight is used as the objective.

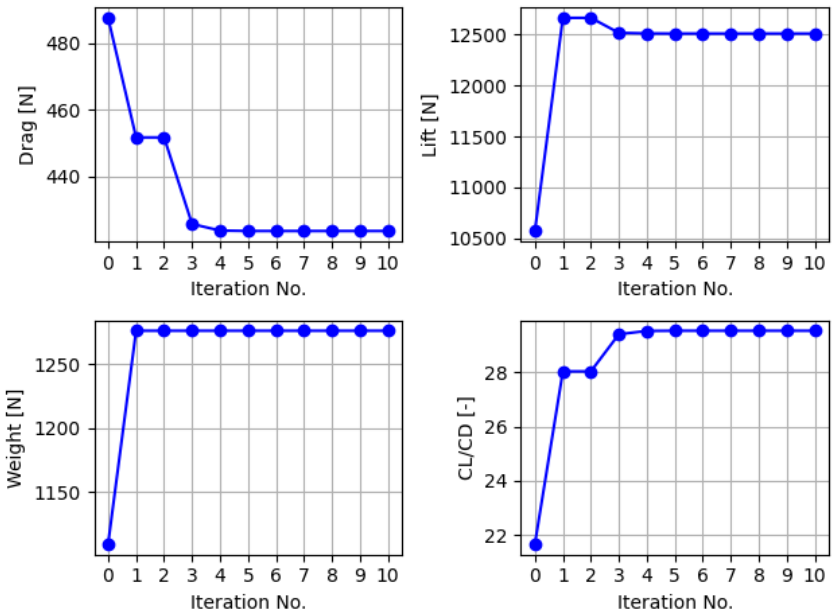


Figure 6.15: ONERA M6 aerostructural MDO: Evolution of the wing drag (upper left), lift (upper right), weight (lower left) and lift-to-drag ratio (lower right). The lift-to-drag ratio is used as the objective.

The second objective function used is the lift-to-drag ratio or $\frac{L}{D}$. Here L and D are the lift and drag values of the wing. Maximizing this function is mainly done with the aim of improving aerodynamic performance, namely reducing drag and increasing lift. Again, the optimization is performed using MDF. The evolution of the wing drag, lift, weight and lift-to-drag ratio is shown in fig. 6.15. Similarly, the evolution of the total aircraft lift and weight, and the stress constraint g is shown in fig. 6.16. The optimized stress and thickness distributions for the wing are shown in fig. 6.17.

The optimizer is able to both increase the lift and reduce the drag of the wing. Interestingly, the optimizer is able to reduce the drag by a larger amount than with the previous objective ($\frac{D}{D_0} + \frac{W}{W_0}$). The lift value is also significantly increased. Figure 6.17 reveals that the optimized thickness distribution is constant, with all structural elements having the same thickness, equal to the allowed upper bound. By observing the evolution of the stress constraint g, it is revealed that for all iterations the value of g is near, or marginally exceeding, the allowed limit. Therefore, the optimizer moves the thickness values close to the upper bound, to satisfy the constraint, and ,unlike before, there is no reason to decrease any of them. Hence, the distribution is constant with a value equal to the upper bound.

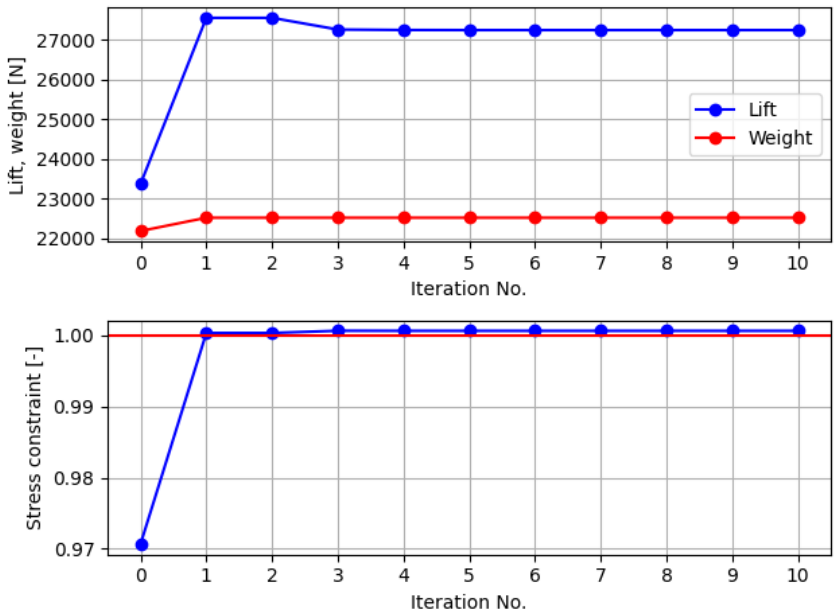


Figure 6.16: ONERA M6 aerostructural MDO: Evolution of the total aircraft lift and weight, and the stress constraint. The lift-to-drag ratio is used as the objective.

This means that the structural weight of the wing increases, as shown in fig. 6.15. However, the increase in lift more than makes up for it, as the optimized wing is able to carry more extra weight than the baseline (fig. 6.16). Thus, this objective function ($\frac{L}{D}$) performs better for this problem and setup than the weighed sum of the wing's drag and weight ($\frac{D}{D_0} + \frac{W}{W_0}$). Regardless of the objective function used, the MDF architecture is able to improve the design and converge towards the optimum.

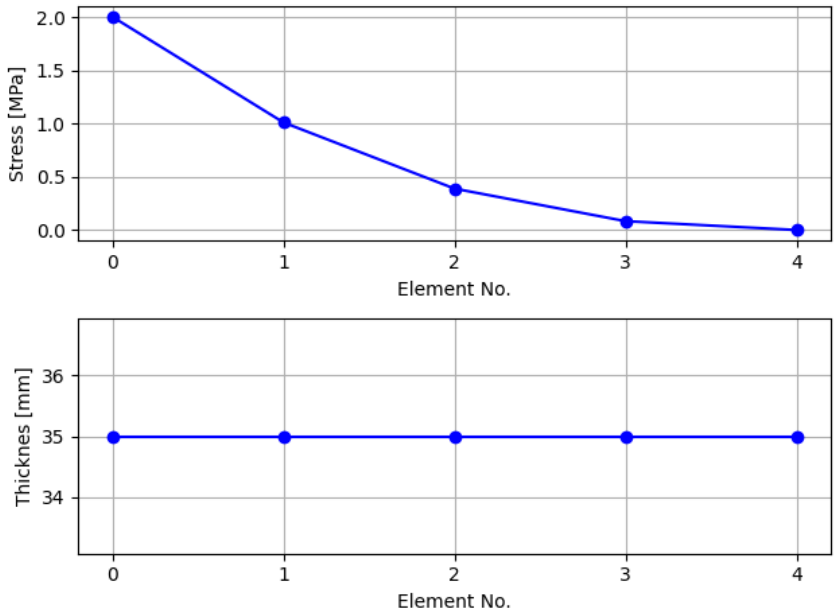


Figure 6.17: ONERA M6 aerostructural MDO: Stress (upper) and thickness (lower) distributions along the wing's span. Element number 0 corresponds to the element closest to the wing's root and 4 to the tip. The lift-to-drag ratio is used as the objective.

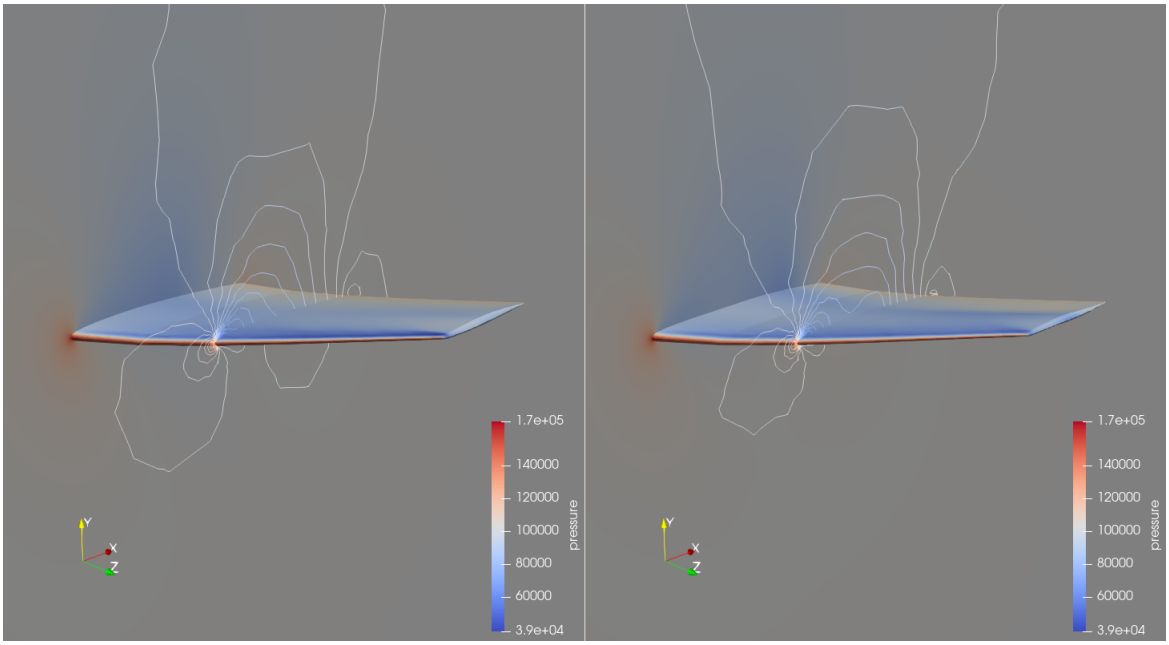


Figure 6.18: ONERA M6 aerostructural MDO: Comparison of the deformed configurations for the baseline (left) and optimized (right) wings. The pressure field contours are shown. The optimized wing is stiffer than the baseline, therefore it bends less. The lift-to-drag ratio is used as the objective.

Chapter 7

Conclusions and recommendations for future work

7.1 Summary and conclusions

The aim of this thesis was to showcase and implement MDAO, and then apply it to different problems. The theory behind MDAO was detailed, explaining the difference between single-discipline and multidisciplinary numerical models, and presenting different MDA methods, MDO architectures and ways to compute the coupled derivatives. A Python framework, named mSense, was developed to facilitate easy setup and solution of multidisciplinary problems. The theory was first applied to standard MDO benchmark problems, which were used compare the performance of three MDO architectures. Then, an FSI problem was solved, which was concerned with the analysis and optimization of an airfoil-spring system. The resulting multidisciplinary shape optimization problem was solved using two monolithic architectures, MDF and IDF. The solution of a more computationally expensive FSI problem, simulating flow inside of an elastic, deformable tube followed. Finally, the methodology is applied to the aerostructural analysis and optimization of an aircraft wing.

MDAO is a very valuable tool. It provides a robust mathematical framework for the analysis and design of complex, coupled systems. However, not all methods are applicable or suitable to all problems. For example, for the problems tested, it was concluded that the monolithic MDF and IDF architectures tend to perform overall better than the distributed CO architecture. Furthermore, although IDF performed better than MDF for the simpler benchmark problems, in the airfoil-spring FSI problem the opposite held true. Overall, drawing from the results obtained in this thesis, MDF seems to be the most generally applicable and performant architecture.

7.2 Recommendations for future work

Some recommendations for future work are:

- Formulate MDAO problems using the residual form. The main benefit of the residual form is that the computation of coupled derivatives (via the coupled adjoint method) can become significantly more efficient, especially in problems with many coupling variables and computationally demanding disciplinary solvers. However, due to the invasive nature of the residual form, this requires not only changes to existing MDAO software, but also to the physics solvers, as they must provide extensive access to their internals.

- Investigation of other MDO architectures. The architectures presented in this thesis are well-established and have been studied quite extensively. Newer architectures may provide better performance, especially for specific problems. For example, Asymmetric Subspace Optimization (ASO), developed for aerostructural optimization problems, has been shown to outperform MDF under certain conditions [8].

- Use of high-fidelity solvers for aerostructural optimization. In the aerostructural wing analysis and optimization application presented in this thesis, the fluid analysis was performed using the Euler equations and the structural behaviour was simulated using simple beam elements. The use of a viscous fluid analysis and a more detailed structural model, such as a shell finite element model, provides the opportunity for better and more realistic designs.

Bibliography

[1] N. Alexandrov and R. Lewis. Analytical and computational aspects of collaborative optimization for multidisciplinary design. *AIAA*, 40:301–309, 2002.

[2] E. Arian. *Convergence Estimates for Multidisciplinary Analysis and Optimization*. ICASE Report No. 97-57. NASA Langley Research Center. Institute for Computer Applications in Science and Engineering [ICASE], 1997.

[3] V. G. Asouti, X. S. Trompoukis, I. C. Kampolis, and K. C. Giannakoglou. Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on graphics processing units. *International Journal for Numerical Methods in Fluids*, 67(2):232–246, 2011.

[4] V. Balabanov, C. Charpentier, D. Ghosh, G. Quinn, G. Vanderplaats, and G. Venter. Visualdoc: A software system for general purpose integration and design optimization. *AIAA 2002-5513*, 2002.

[5] R. Braun. *Collaborative optimization: an architecture for large-scale distributed design*. PhD thesis, Standford University, 1996.

[6] R. Braun, I. Kroo, and P. Gage. Post-optimality analysis in aerospace vehicle design. *AIAA 93-3932*, 1993.

[7] I. Budianto and J. Olds. Design and deployment of a satellite constellation using collaborative optimization. *Journal of Spacecraft and Rockets*, 41(6):p.956–963, 2004.

[8] I. Chittick and J. Martins. An asymmetric suboptimization approach to aerostructural optimization. *Optimization and Engineering*, 10:133–152, 03 2009.

[9] F. Gallard, C. Vanaret, D. Guénot, V. Gachelin, R. Lafage, B. Pauwels, P.-J. Barjhoux, and A. Gazaix. Gems: A python library for automation of multidisciplinary design optimization process generation. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018.

[10] K. Giannakoglou and D. Papadimitriou. *Adjoint Methods for Shape Optimization*, pages 79–108. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[11] J. Gray, J. Hwang, J. Martins, K. Moore, and B. Naylor. Openmdao: An open-source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 2019.

[12] Justin Gray, Kenneth Moore, Tristan Hearn, and Bret Naylor. Standard platform for benchmarking multidisciplinary design analysis and optimization architectures. *AIAA Journal*, 51:2380–2394, 10 2013.

[13] R. Haftka. Automated procedure for design of wing structures to satisfy strength and flutter requirements. Technical report, NASA Langley research center, 1973.

[14] R. Haftka. Simultaneous analysis and design. *AIAA Journal 1099-1103*, 1985.

[15] J. Hwang, D. Lee, J. Cutler, and J. Martins. Large-scale multidisciplinary optimization of a small satellite's design and operation. *Journal of Spacecraft and Rockets*, 51, 09 2014.

[16] G. Kennedy and J. Martins. Parallel solution methods for aerostructural analysis and design optimization. *13th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference 2010*, 09 2010.

[17] G. Kenway, G. Kennedy, and J. Martins. Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and adjoint derivative computations. *AIAA Journal*, 52:935–951, 2014.

[18] G. Kenway and J. Martins. Buffet-onset constraint formulation for aerodynamic shape optimization. *AIAA Journal*, 55:1–18, 2017.

[19] P. Koch, B. Wujek, O. Golovidov, and T. Simpson. Facilitating probabilistic multidisciplinary design optimization using kriging approximation models. *AIAA 2002-5415*, 2002.

[20] D. Kraft. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988.

[21] A. Lambe, G. Kennedy, and J. Martins. An evaluation of constraint aggregation strategies for wing box mass minimization. *Structural and Multidisciplinary Optimization*, 55, 2017.

[22] A. Lambe and J. 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.

[23] L. Ledsinger. Solutions to decomposed branching trajectories with powered flyback using multidisciplinary design optimization. 2000.

[24] R. Lewis, G. Shubin, E. Cramer, J. Dennis, P. Frank, R. Michael, L. Gregory, and R. Shubin. Problem formulation for multidisciplinary optimization. *SIAM Journal on Optimization*, 4, 02 1997.

[25] J. Martins, J. Alonso, and J. Reuther. A coupled-adjoint sensitivity analysis method for high-fidelity aero-structural design. *Optimization and Engineering*, 6(1):33–62, 2005.

[26] J. Martins and A. Ning. *Engineering Design Optimization*. Cambridge University Press, Cambridge, UK, 2022.

[27] J. Martins, P. Sturdza, and J. Alonso. The complex-step derivative approximation. *ACM Trans. Math. Softw.*, 29:245–262, 2003.

[28] S. Padula, N. Alex, L. Green, and N. Alexandrov. MDO test suite at nasa langley research center. 10 1996.

[29] B. Potsaid, Y. Bellouard, J. Ting, and J.T. Wen. A multidisciplinary design and optimization methodology for the adaptive scanning optical microscope (ASOM) - art. no. 62890l. *Proceedings of SPIE - The International Society for Optical Engineering*, 6289, 09 2006.

[30] M. J. D. Powell. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pages 51–67. Springer Netherlands, Dordrecht, 1994.

[31] V. Schmitt and C. Françoise. Pressure distributions on the onera m6 wing at transonic mach numbers. Technical report, AGARD, 1979.

[32] R. Sellar, S.M. Batill, and J. Renaud. Response surface based, concurrent subspace optimization for multidisciplinary system design. 1996.

[33] J. Sobieski. *Recent Experiences in Multidisciplinary Analysis and Optimization, Part 1*. 1984.

[34] J. Sobieski. Sensitivity of complex, internally coupled systems. *AIAA Journal*, 28(1):153–160, 1990.

[35] J. Sobieski, T. Altus, M. Phillips, and R. Sandusky. Bilevel integrated system synthesis for concurrent and distributed processing. *AIAA Journal*, 41(10):1996–2003, 2003.

[36] N. Tedford and J. Martins. Benchmarking multidisciplinary design optimization algorithms. *Optimization and Engineering*, 11:159–183, 02 2010.

[37] K. Tsiakas, X. Trompoukis, V. Asouti, K. Giannakoglou, G. Rogé, S. Julisson, L. Martin, and S. Kleinveld. Discrete and continuous adjoint-based aerostructural wing shape optimization of a business jet. *Fluids*, 9(4), 2024.

[38] P. Virtanen, R. Gommers, T. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. van der Walt, M. Brett, J. Wilson, K. Millman, N. Mayorov, A. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, I. Polat, Y. Feng, E. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. Harris, A. Archibald, A. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[39] A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, Mar 2006.

Appendix A

MSense basic user guide

MSense is a Python package developed for MDAO. It allows the user to quickly setup and solve MDAO problems, and makes it easy to switch between various MDA methods and MDO architectures. MSense is written with an object-oriented approach and can be used through its Python Application Programming Interface (API). In order to demonstrate the use of mSense, the code for setup and solution of Sellar's problem (from chapter 3) is presented.

Sellar's problem is an analytic MDO problem which consists of two disciplines and an objective. The variables present in the problem are the shared design variables z_1 and z_2, the local design variable x_1, the coupling variables y_1 and y_2, the local constraints g_1 and g_2 and the objective f. For Sellar's problem, the variables are defined in code as follows:

```
1  # Import everything from the API
2  from msense.api import *
3
4  # Design variables
5  z1 = Variable(name="z1", lb=-10, ub=10)
6  z2 = Variable(name="z2", lb=0, ub=10)
7  x1 = Variable(name="x1", lb=0, ub=10)
8
9  # Couplings
10 y1 = Variable(name="y1)
11 y2 = Variable(name="y2")
12
13 # Constraints and objective
14 g1 = Variable("g1", lb=-np.inf, ub=0, keep_feasible=False)
15 g2 = Variable("g2", lb=-np.inf, ub=0, keep_feasible=False)
16 f = Variable("f")
```

The first step to using mSense is always to import the functions and classes defined in the API module (line 2). All variables present in a problem are defined in mSense using the `Variable` class. In order to define a `Variable`, the user

must specify its unique name and optionally its size, which defaults to 1. If the `Variable` corresponds to a design variable, the user can also specify its lower `lb` and upper `ub` bounds, which if not specified default to $-\infty$ and ∞ respectively. `Variable`s which correspond to constraints should have at least one of `lb` and `lb` set to a finite value. This is because in mSense constraints are specified through variable bounds as follows:

$$lb \leq g \leq ub \tag{A.1}$$

If `lb=ub`, the constraint is treated as an equality constraint. The user can also specify whether the constraint should remain inside the feasible region during optimization, by setting the option `keep_feasible` to `True` (lines 14 and 15).

The next step after defining all variables is to define the disciplines. In mSense, the evaluation and differentiation of a discipline is wrapped inside the `Discipline` class, which is a Python Abstract Base Class (ABC). When defining a `Discipline`, the user must first specify its unique name along with its input and output `Variable`s. Two of `Discipline`'s methods are not implemented, and have to be overridden by the user. These are the `_eval()` and `_differentiate()` methods, which specify how the discipline evaluates its outputs and its disciplinary derivatives. In the code snippet below, the first discipline in Sellar's problem is defined:

```
1  from msense.api import *
2  import numpy as np
3
4  class SellarDiscipline1(Discipline):
5      def __init__(self, z1: Variable, z2: Variable,
6                   x1: Variable, y2: Variable,
7                   y1: Variable, g1: Variable):
8
9          # Initialize the base object
10         super().__init__(name="SellarDiscipline1",
11                          input_vars=[z1, z2, x1, y2],
12                          output_vars=[y1, g1])
13
14     def _eval(self) -> None:
15         # Get the input variable values
16         _z1 = self._values["z1"]
17         _z2 = self._values["z2"]
18         _x1 = self._values["x1"]
19         _y2 = self._values["y2"]
20
21         # Compute y1 and g1
22         self._values["y1"] = np.sqrt(_z1**2 + _z2 + _x1 - 0.2 *
       _y2)
23         self._values["g1"] = 3.16 - self._values["y1"]**2
```

```
24
25      def _differentiate(self) -> None:
26          # Get the input variable values
27          _z1 = self._values["z1"]
28          _z2 = self._values["z2"]
29          _x1 = self._values["x1"]
30          _y2 = self._values["y2"]
31          _y1 = self._values["y1"]
32
33          # Compute the derivatives of y1
34          self._jac["y1"] = {"z1": _z1/_y1,
35                             "z2": 1 / (2*_y1),
36                             "x1": 1/(2*_y1),
37                             "y2": -0.2/(2*_y1)}
38
39          # Compute the derivatives of y1
40          self._jac["g1"] = {"z1": -2*_y1*self._jac["y1"]["z1"],
41                             "z2": -2*_y1*self._jac["y1"]["z2"],
42                             "x1": -2*_y1*self._jac["y1"]["x1"],
43                             "y2": -2*_y1*self._jac["y1"]["y2"]}
```

Apart from importing the mSense API, `numpy` is also imported. The class `SellarDiscipline1` is derived from the `Discipline` base class, as seen on line 4. Inside the `__init__()` function, the instance of the base class must also be initialized. This is done on line 10, using the `super().__init__()` method. The name of the discipline, and its input and output variables are passed as arguments to this method. The only necessary step left to fully define the discipline is to override the `_eval()` method, which specifies how the discipline evaluates its outputs. As seen on lines 15-19, the user can access a discipline's current input variable values through the `_values` dictionary. Then, using these values, the outputs are computed and their values must be placed inside the `_values` dictionary (lines 22-23). Although not required, the user can also override the `_differentiate()` method to specify how the discipline computes its derivatives. Similar to `_eval()`, the user can access the discipline's current input and output values using the `_values` dictionary. Using these, the user should update the `_jac` dictionary, which stores the disciplinary derivatives (lines 34-43). `_jac` consists of one dictionary per output variable, which in turn stores the value of the derivatives of the output variable w.r.t each input variable. This is demonstrated in lines 32-41. If the `_differentiate()` method is not overridden, the disciplinary derivatives can also be approximated. This can be achieved by calling the `set_jacobian_approximation()` method on the instance of the derived class, for example:

```
1       sellar_disc_1 = SellarDiscipline1(z1, z2, x1, y2, y1, g1)
2       sellar_disc_1.set_jacobian_approximation()
```

When calling the `set_jacobian_approximation()` method the user can optionally specify the method use for approximation (either `"finite_difference"` or `"complex_step"`) and the step-size. By default the finite-differences are used, with a step-size of 1^{-4}. If the user wishes to evaluate or differentiate the discipline, the `eval()` and `differentiate()` methods should be called. For example:

```
sellar_disc_1 = SellarDiscipline1(z1, z2, x1, y2, y1, g1)
input_values = {"x1": 1.0, "z1": 4, "z2": 3, "y2": 0.9}
output_values = sellar_disc_1.eval(input_values)
jacobian = sellar_disc_1.differentiate(input_values)
```

Here, `input_values` is dictionary holding the input variable values with which the user wishes to evaluate the disciplinary outputs. By calling the `eval()` method, mSense first checks that no input variables values are missing and that the sizes of all passed variable values are correct, then calls `_eval()`, and returns the output variable values as a dictionary named `output_values`. The validity of the output values is also checked. It should be mentioned here that mSense has the capability to cache discipline evaluations, which avoids re-evaluating or re-differentiating a discipline with the same input values. By default, only the values of the latest successful evaluation/differentiation are cached. The user can change this by setting the value of `cache_policy` inside the base class initializer (`super().__init()`). Accepted values are `"latest"` (cache only the latest evaluation, default), `"all"` (cache all evaluations) and `None` (do not cache). `differentiate()` works similar to `eval()`. By default, a `Discipline` has to be first evaluated for a set of input values before it is differentiated. This can be changed by setting the field `_diff_policy` from `True` to `False`.

The second discipline and the objective in Sellar's problem are similar are defined similar to the first discipline:

```
class SellarDiscipline2(Discipline):
    def __init__(self, z1: Variable, z2: Variable,
                 y1: Variable, y2: Variable,
                 g2: Variable):

        # Initialize the base object
        super().__init__(name="Disc2",
                         input_vars=[z1, z2, y1],
                         output_vars=[y2, g2])

    def _eval(self) -> None:
        # Get the input variable values
        _z1 = self._values["z1"]
        _z2 = self._values["z2"]
        _y1 = self._values["y1"]

        # Compute y2 and g2
```

```
        self._values["y2"] = np.abs(_y1) + _z1 + _z2
        self._values["g2"] = self._values["y2"] - 24

    def _differentiate(self) -> None:
        # Get the input variable values
        _z1 = self._values["z1"]
        _z2 = self._values["z2"]
        _y1 = self._values["y1"]

        # Compute the derivatives of y2 and g2
        self._jac["y2"] = {"y1": np.sign(_y1), "z1": 1.0, "z2": 1.0}
        self._jac["g2"] = {"y1": np.sign(_y1), "z1": 1.0, "z2": 1.0}


class SellarObjective(Discipline):
    def __init__(self, x1: Variable, z2: Variable,
                 y1: Variable, y2: Variable,
                 f: Variable):

        # Initialize the base object
        super().__init__("Objective", [x1, z2, y1, y2], [f])

    def _eval(self) -> None:
        # Get the input variable values
        _x1 = self._values["x1"]
        _z2 = self._values["z2"]
        _y1 = self._values["y1"]
        _y2 = self._values["y2"]

        # Compute f
        self._values["f"] = _x1**2 + _z2 + _y1**2 + np.exp(-_y2)

    def _differentiate(self) -> None:
        # Get the input variable values
        _x1 = self._values["x1"]
        _z2 = self._values["z2"]
        _y1 = self._values["y1"]
        _y2 = self._values["y2"]

        # Compute the derivatives of f
        self._jac["f"]["x1"] = 2*_x1
        self._jac["f"]["z2"] = 1.0
        self._jac["f"]["y1"] = 2 * _y1
        self._jac["f"]["y2"] = -np.exp(-_y2)
```

The user can now perform an MDA as follows:

```
# Instantiate discipline objects
```

```
2 sellar_disc_1 = SellarDiscipline1(z1, z2, x1, y2, y1, g1)
3 sellar_disc_2 = SellarDiscipline2(z1, z2, y1, y2, g2)
4 sellar_objective = SellarObjective(x1, z2, y1, y2, f)
5
6 # Create the MDA solver
7 gs_solver = create_solver(type=SolverType.NONLINEAR_GS,
8                          n_iter_max=10,
9                          relax_fact=1,
10                         tol=1e-4,
11                         disciplines=[sellar_disc_1,
12                                      sellar_disc_2,
13                                      sellar_objective])
14
15 # Set starting values
16 starting_values = {"x1": 1.0, "z1": 4, "z2": 3, "y1": 0.8, "y2":
       0.9}
17
18 # Solve the MDA problem
19 coupling_values = gs_solver.solve(starting_values)
```

After instantiating the three `Discipline` derived objects (lines 2-4), the user creates a `Solver` object, through the `create_solver()` function. This function returns a `Solver` object after specifying which disciplines are involved in the MDA. The user can also optionally specify the type of solver, the maximum iterations allowed, the tolerance and the relaxation factor. The default solver type is `"nonlinear_gs"`, which corresponds to the nonlinear Gauss-Seidel MDA. The nonlinear Jacobi and Newton methods are also available by setting the type to `"nonlinear_jacobi"` and `"newton_raphson` respectively. By calling the `solve()` method with the specified starting values, the solver solves the MDA and returns the converged values of (only) the coupling variables.

In order to solve an MDO problem, the user must first create it using the `create_opt_problem()` function, which returns an `OptProblem` object. The design variables, objective, constraints and involved disciplines have to be defined when creating the problem. The user can also choose if the underlying optimizer (called a `Driver` in mSense) should use normalization, by setting the `use_norm` argument to either `True` or `False`. Normalization can only be used if all design variables have finite bounds. By setting `maximize_objective` to `True`, the problem is solved as a maximization problem. Bu default, `use_norm` is set to `True` and `maximize_objective` to `False`. The user can also set the `type` of `OptProblem`. This corresponds to which MDO architecture should be used to formulate and solve the problem. The default option is the MDF architecture (`type="mdf"`), but the IDF (`type="idf"`) and CO (`type="co"`) architectures are also available. If the MDF architecture is used, the user is also required to specify the accompanying MDA solver, as seen in the snippet below:

```
# Create the MDO problem
prob = create_opt_problem(type=OptProblemType.MDF,
                      design_vars=[x1, z1, z2],
                      objective=f,
                      constraints=[g1, g2],
                      maximize_objective=False,
                      use_norm=True
                      disciplines=[sellar_disc_1,
                                   sellar_disc_2,
                                   sellar_objective],
                      solver=gs_solver)
```

After creating the MDO problem, the user has to set the initial values of the design vector, and call the `solve()` method of the `OptProblem` object. This method returns the optimized values of the design variables. The user can see the evolution of the objective function by calling the `plot_objective_history()` method of the `OptProblem` object.

```
# Set initial design vector values
initial_values = {"x1": 1.0, "z1": 4, "z2": 3}

# Solve the problem
optimized_values = prob.solve(initial_values)

# Plot the objective's evolution
prob.plot_objective_history()
```

The complete code for Sellar's problem, as well as other examples can be found in the GitHub repository (https://github.com/dlmpal/mSense), under the `examples` folder. In the examples therein more advanced features of mSense are shown, such as how to set the `Driver` of an `OptProblem`.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής
& Βελτιστοποίησης

Πολυτομεακή Ανάλυση και Βελτιστοποίηση: Θεωρία, Υλοποίηση και Εφαρμογές

Διπλωματική Εργασία

Δημήτριος Πάλλας

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Εισαγωγή

Τα περισσότερα σύγχρονα προβλήματα στη μηχανολογία είναι πολυτομεακά-πολυεπιστημονικά, καθώς ασχολούνται με συστήματα τα οποία αποτελούνται από διαφορετικές συνιστώσες-πεδία, και συχνά εξετάζεται η αλληλεπίδρασή τους με πολλαπλά φυσικά φαινόμενα. Η ανάγκη σχεδιασμού τέτοιων συστημάτων ώθησε την ανάπτυξη της θεωρίας της Πολυτομεακής Ανάλυσης και Βελτιστοποίησης (Multidisciplinary Analysis and Optimization, MDAO), η οποία παρέχει το μαθηματικό υπόβαθρο για την ανάλυση ή/και τη βελτιστοποίηση των συστημάτων αυτών. Το κύριο όφελος της μεθοδολογίας MDAO είναι ότι μπορεί να εκμεταλλευτεί τις αλληλεπιδράσεις μεταξύ των διαφορετικών συνιστωσών-πεδίων του συστήματος, με αποτέλεσμα να οδηγεί στο βέλτιστο σχεδιασμό. Αυτό είναι αδύνατο αν δεν ληφθούν υπόψη οι αλληλεπιδράσεις, ή εάν γίνεται η υπόθεση ότι η σύζευξη είναι μονόδρομη.

Η θεωρία MDAO έχει τις ρίζες της στην αεροναυπηγική [2]. Κλασικά πολυτομεακά πρόβλημα είναι τα προβλήματα αλληλεπίδρασης ρευστού-στερεού (Fluid-Structure Interaction, FSI), όπως η αεροδομική ανάλυση και βελτιστοποίηση πτέρυγας αεροσκάφους. Η κλασική προσέγγιση επίλυσης του προβλήματος αυτού, όσον αφορά την ανάλυση, είναι ο υπολογισμός των αεροδυναμικών φορτίων, η μεταφορά τους στη δομική ανάλυση και, τέλος, ο υπολογισμός των τάσεων και παραμορφώσεων της πτέρυγας. Προφανώς όμως, η μετατόπιση της πτέρυγας οδηγεί σε αλλαγή της ροής γύρω από αυτήν, άρα και σε αλλαγή των φορτίων. Προβλήματα όπως αυτό, όπου η αμφίδρομη σύζευξη μεταξύ πεδίων δεν μπορεί να αγνοηθεί, αποτελούν το κύριο αντικείμενο μελέτης της θεωρίας MDAO.

Ορολογία και μαθηματική περιγραφή

Τα συστήματα στη μηχανολογία και στη μηχανική γενικότερα μοντελοποιούνται αριθμητικά μέσω ενός συστήματος εξισώσεων, με τη λύση των οποίων προσεγγίζεται η κατάσταση του συστήματος. Για ένα μοντέλο με n μεταβλητές κατάστασης, οι εξισώσεις αυτές γράφονται ως:

$$r_i(y_1, y_2, \ldots, y_n) = 0,\ i \in (1, n) \tag{1}$$

Στην παραπάνω έκφραση, r_i είναι το υπόλοιπο της i-οστής εξίσωσης και y_i η i-οστή μεταβλητή κατάστασης. Γράφοντας τις μεταβλητές κατάστασης και τα υπόλοιπα σε διανυσματική μορφή, δηλαδή $Y = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}^\top$ και $R = \begin{bmatrix} r_1 & r_2 & \dots & r_n \end{bmatrix}^\top$, η παραπάνω έκφραση γράφεται σύντομα ως:

$$R(Y) = 0 \tag{2}$$

Η λογική αυτή επεκτείνεται σε πολυτομεακά συστήματα, αν θεωρηθεί ότι κάθε υποσύστημα-πεδίο διαθέτει το δικό του διάνυσμα μεταβλητών κατάστασης Y_i και σετ εξισώσεων R_i. Για ένα σύστημα m πεδίων, και λαμβάνοντας υπόψη τις αλληλεπιδράσεις μεταξύ των πεδίων, ισχύει ότι:

$$R_i(Y_1, Y_2, \dots, Y_m) = 0, i \in (1, m) \tag{3}$$

Η παραπάνω εκδοχή του μοντέλου του πολυτομεακού συστήματος ονομάζεται υπολειμματική. Η εναλλακτική εκδοχή αυτής είναι η συναρτησιακή, όπου αντί των μεταβλητών κατάστασης Y_i κάθε πεδίου, θεωρούνται μόνο οι μεταβλητές εξόδου $\hat{Y}_i$. Οι $\hat{Y}_i$ είναι είτε υποσύνολο των Y_i ή υπολογίζονται άμεσα από αυτές. Οι είσοδοι ενός πεδίου i συμβολίζονται με $Y_{j \neq i}$ και είναι υποσύνολο των εξόδων όλων των άλλων πεδίων, άρα στη γενική περίπτωση ισχύει ότι $\hat{Y}_{j \neq i} = \begin{bmatrix} \hat{Y}_1 & \dots & \hat{Y}_{i-1} & \hat{Y}_{i+1} & \dots & \hat{Y}_m \end{bmatrix}^\top$. Έτσι, για την συναρτησιακή μορφή του πολυτομεακού μοντέλου, το σύνολο εξισώσεων που το περιγράφει είναι:

$$\hat{Y}_i = \hat{Y}_i(\hat{Y}_{j \neq i}), i \in (1, m) \tag{4}$$

Κάθε εκδοχή έχει διαφορετικά πλεονεκτήματα και μειονεκτήματα. Εδώ αναφέρεται μόνο η προφανής διαφορά ότι η υπολειμματική μορφή διαχειρίζεται πολύ μεγαλύτερο αριθμό μεταβλητών από την συναρτησιακή, καθώς το μέγεθος των εξόδων $\hat{Y}_i$ είναι συχνά πολύ μικρότερο του μεγέθους των μεταβλητών κατάστασης Y_i.

Η θεωρία MDAO ασχολείται με την ανάλυση και βελτιστοποίηση πολυτομεακών μοντέλων. Η πολυτομεακή ανάλυση (Multidisciplinary Analysis, MDA) είναι η διαδικασία ταυτόχρονης επίλυσης των εξισώσεων όλων των υποσυστημάτων/πεδίων. Η πολυτομεακή βελτιστοποίηση (Multidisciplinary Optimization, MDO) είναι η διαδικασία βελτιστοποίησης ενός πολυτομεακού συστήματος, η οποία παράλληλα σέβεται την συμβατότητα μεταξύ των πεδίων, δηλαδή λαμβάνει υπόψη τις αλληλεπιδράσεις τους. Ένα πρόβλημα MDO μπορεί να επιλυθεί με περισσότερους από έναν τρόπους, ή αρχιτεκτονικές. Οι αρχιτεκτονικές MDO χωρίζονται σε δύο ευρείες κατηγορίες, μονολιθικές και κατανεμημένες. Οι πρώτες επιλύουν ένα πρόβλημα βελτιστοποίησης, ενώ οι δεύτερες πολλαπλά. Παραδείγματα της πρώτης κατηγορίας είναι οι αρχιτεκτονικές Multidisciplinary Feasible (MDF) [3] και Individual Discipline Feasible (IDF) [3], ενώ παραδείγματα της δεύτερης είναι οι Bi-Level Integrated System Synthesis (BLISS) [6] και Collaborative Optimization (CO) [1].

Τα περισσότερα προβλήματα MDO, ασχέτως αρχιτεκτονικής, επιλύονται μέσω αιτιοκρατικών αλγορίθμων βελτιστοποίησης, λόγω της γενικά καλύτερης επίδοσής τους. Οι αλγόριθμοι αυτοί απαιτούν τις παραγώγους της συνάρτησης-στόχου και των περιορισμών ως προς τις μεταβλητές σχεδιασμού. Στην πολυτομεακή βελτιστοποίηση, ο υπολογισμός των παραγώγων αυτών πρέπει να λαμβάνει υπόψη την αλληλεπίδραση των πεδίων, και για αυτό ονομάζονται και πεπλεγμένες παράγωγοι. Για τον αποδοτικό υπολογισμό των πεπλεγμένων παραγώγων έχουν αναπτυχθεί πεπλεγμένα ή πολυτομεακά ανάλογα των μεθόδων της ευθείας διαφόρισης και της συζυγούς μεθόδου [5].

Το λογισμικό mSense

Για τη διαχείριση της πολυπλοκότητας που σχετίζεται με το MDAO αναπτύσσεται ένα πακέτο λογισμικού στη γλώσσα Python. Το πάκετο ονομάζεται mSense (Multidisciplinary + Sensitivity), και χρησιμοποιείται για της εφαρμογές της διπλωματικής εργασίας. Σκοπός του πακέτου είναι η διευκόλυνση της ανάπτυξης πολυτομεακών αριθμητικών μοντέλων, μέσω σύζευξης εξειδικευμένων λογισμικών ανάλυσης για κάθε πεδίο. Το mSense διαθέτει εργαλεία τόσο για την ανάλυση , όσο και για την βελτιστοποίηση τέτοιων μοντέλων. Όσον αφορά τη βελτιστοποίηση, στο πακέτο είναι προγραμματισμένες τρεις αρχιτεκτονικές MDO, οι MDF, IDF και CO, και παρέχεται η δυνατότητα υπολογισμού των πεπλεγμένων παραγώγων τόσο μέσω των πεπλεγμένων εκδοχών της μεθόδου ευθείας διαφόρισης και της συζυγούς μεθόδου, αλλά και μέσω προσέγγισης (πεπερασμένες διαφορές). Ο πηγαίος κώδικας, καθώς και παραδείγματα χρήσης, είναι διαθέσιμα εδώ: https://github.com/dlmpal/mSense.

Θεωρία MDAO

Πολυτομεακή ανάλυση

Σκοπός της πολυτομεακής ανάλυσης (Multidisciplinary Analysis, MDA) είναι η επίλυση του συστήματος εξισώσεων του πολυτομεακού μοντέλου. Πρακτικά αυτό ισοδυναμεί με ταυτόχρονη ικανοποίηση των εξισώσεων κατάστασης κάθε πεδίου. Οι μέθοδοι MDA χωρίζονται πρακτικά σε δύο κατηγορίες. Η πρώτη κατηγορία αποτελείται από γενικευμένες μεθόδους σταθερού σημείο, όπως η μη-γραμμική, block

μέθοδος Gauss-Seidel και η μη-γραμμική, block μέθοδος Jacobi. Η δεύτερη κατηγορία αποτελείται από μεθόδους βασισμένες στη μέθοδο Newton για μη-γραμμικά συστήματα.

Υπολογισμός πεπλεγμένων παραγώγων

Για την εφαρμογή αιτιοκρατικής βελτιστοποίησης απαραίτητος είναι ο υπολογισμός των παραγώγων της συνάρτησης-στόχου και των περιορισμών ως προς τις μεταβλητές σχεδιασμού. Σε μονοτομεακά συστήματα ή συστήματα ενός πεδίου για τον σκοπό αυτό χρησιμοποιούνται ή μέθοδος ευθείας διαφόρισης και η συζυγής μέθοδος. Αν με F συμβολίζεται το διάνυσμα που περιέχει την συνάρτηση-στόχο και όλους του περιορισμούς, με X το διάνυσμα μεταβλητών σχεδιασμού και με Y και R τα διανύσματα μεταβλητών κατάστασης και υπολοίπων αντίστοιχα, τότε η μέθοδος ευθείας διαφόρισης γράφεται:

$$\frac{dF}{dX} = \frac{\partial F}{\partial X} - \frac{\partial F}{\partial Y}\left(\frac{\partial R}{\partial Y}^{-1}\frac{\partial R}{\partial X}\right) \tag{5}$$

Ομοίως, η συζυγής μέθοδος γράφεται:

$$\begin{aligned} \frac{dF}{dX} &= \frac{\partial F}{\partial X} - \psi^\top \frac{\partial R}{\partial X} \\ \frac{\partial R}{\partial Y}^\top \psi &= \frac{\partial F}{\partial Y}^\top \end{aligned} \tag{6}$$

όπου ψ είναι το διάνυσμα των συζυγών μεταβλητών. Για πολυτομεακά συστήματα έχουν αναπτυχθεί αντίστοιχες μέθοδοι, που ονομάζονται πεπλεγμένη μέθοδος ευθείας διαφόρισης και πεπλεγμένη συζυγής μέθοδος. Για ένα σύστημα m πεδίων, εκφρασμένο στη συναρτησιακή μορφή, η πεπλεγμένη μέθοδος ευθείας διαφόρισης γράφεται:

$$\begin{aligned} &\begin{bmatrix} I & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_2} & \dots & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_m} \\ -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_1} & I & \dots & -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_m} \\ \vdots & \vdots & \dots & \vdots \\ -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_1} & -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_2} & \dots & I \end{bmatrix} \begin{bmatrix} \frac{d\hat{Y}_1}{dX} \\ \frac{d\hat{Y}_2}{dX} \\ \vdots \\ \frac{d\hat{Y}_m}{dX} \end{bmatrix} = \begin{bmatrix} \frac{\partial \hat{Y}_1}{\partial X} \\ \frac{\partial \hat{Y}_2}{\partial X} \\ \vdots \\ \frac{\partial \hat{Y}_m}{\partial X} \end{bmatrix} \\ &\frac{dF}{dX} = \frac{\partial F}{\partial X} + \begin{bmatrix} \frac{\partial F}{\partial \hat{Y}_1} & \frac{\partial F}{\partial \hat{Y}_2} & \dots & \frac{\partial F}{\partial \hat{Y}_m} \end{bmatrix} \begin{bmatrix} \frac{d\hat{Y}_1}{dX} \\ \frac{d\hat{Y}_2}{dX} \\ \vdots \\ \frac{d\hat{Y}_m}{dX} \end{bmatrix} \end{aligned} \tag{7}$$

Για το ίδιο σύστημα η πεπλεγμένη συζυγής μέθοδος γράφεται:

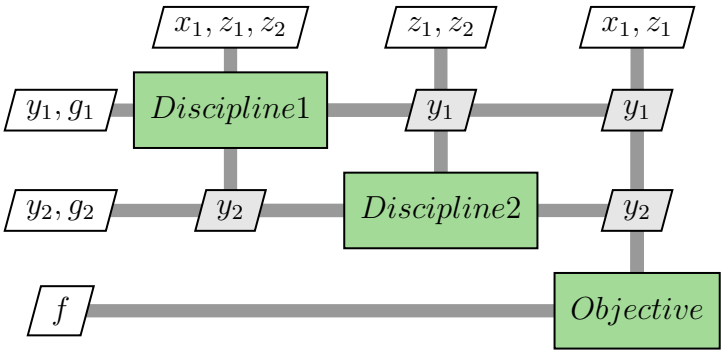
$$\frac{dF}{dX} = \frac{\partial F}{\partial X} - \begin{bmatrix}\Phi_1^\top & \Phi_2^\top & \dots & \Phi_m^\top\end{bmatrix} \begin{bmatrix}\frac{\partial \hat{Y}_1}{\partial X} \\ \frac{\partial \hat{Y}_2}{\partial X} \\ \vdots \\ \frac{\partial \hat{Y}_m}{\partial X}\end{bmatrix}$$

$$\begin{bmatrix} I & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_2}^\top & \dots & -\frac{\partial \hat{Y}_1}{\partial \hat{Y}_m}^\top \\ -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_1}^\top & I & \dots & -\frac{\partial \hat{Y}_2}{\partial \hat{Y}_m}^\top \\ \vdots & \vdots & \dots & \vdots \\ -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_1}^\top & -\frac{\partial \hat{Y}_m}{\partial \hat{Y}_2}^\top & \dots & I \end{bmatrix} \begin{bmatrix}\Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_m\end{bmatrix} = \begin{bmatrix}\frac{\partial F}{\partial \hat{Y}_1}^\top \\ \frac{\partial F}{\partial \hat{Y}_2}^\top \\ \vdots \\ \frac{\partial F}{\partial \hat{Y}_m}^\top\end{bmatrix} \tag{8}$$

όπου $\begin{bmatrix}\phi_1 & \phi_2 & \dots & \phi_m\end{bmatrix}$ είναι το διάνυσμα των συζυγών μεταβλητών.

Πολυτομεακή βελτιστοποίηση

Η κύρια διαφορά της Πολυτομεακής Βελτιστοποίησης (Multidisciplinary Optimization, MDO) σε σχέση με την Μονοτομεακή Βελτιστοποίηση είναι ότι η πρώτη πρέπει να λάβει υπόψιν τις αλληλεπιδράσεις των διαφόρων πεδίων του συστήματος. Ένα πρόβλημα MDO μπορεί να διατυπωθεί και να επιλυθεί με διάφορους τρόπους, καθένας από τους οποίος συνιστά μία διαφορετική αρχιτεκτονική MDO. Οι αρχιτεκτονικές χωρίζονται βάσει δύο κυρίως κριτηρίων. Το πρώτο είναι το πως επιτυγχάνουν τη συμβατότητα των μεταβλητών κατάστασης των πεδίων. Μερικές χρησιμοποιούν MDA, ενώ άλλες περιορισμούς συμβατότητας. Το δεύτερο κριτήριο είναι εάν επιλύουν ένα ή περισσότερα προβλήματα βελτιστοποίησης. Οι λεγόμενες μονολιθικές αρχιτεκτονικές επιλύουν ένα μεγάλο ενιαίο πρόβλημα, ενώ οι κατανεμημένες επιλύουν περισσότερα μικρά προβλήματα. Μία συχνά χρησιμοποιούμενη μονολιθική μέθοδος είναι η MDF, η οποία χρησιμοποιεί MDA για την εξασφάλιση της συμβατότητας των πεδίων. Η μέθοδος IDF είναι επίσης μονολιθική, αλλά χρησιμοποιεί περιορισμούς συμβατότητας για να επιτύχει τη συμβατότητα των πεδίων. Οι περισσότερες κατανεμημένες αρχιτεκτονικές μπορούν να κατηγοριοποιηθούν ως κατανεμημένες παραλλαγές της MDF



Σχήμα 1: Πρόβλημα του Sellar: Διάγραμμα XDSM.

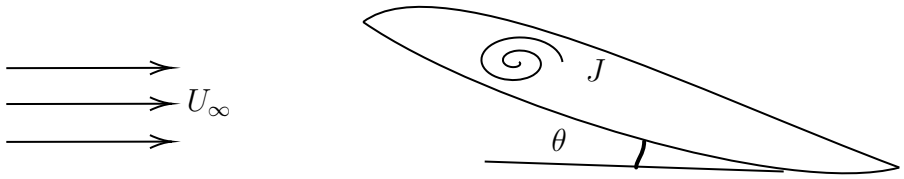
και της IDF, ανάλογα με το πως εξασφαλίζουν τη συμβατότητα.

Σύγκριση επίδοσης διαφόρων αρχιτεκτονικών MDO

Η ύπαρξη διαφορετικών αρχιτεκτονικών MDO δημιουργεί το ερώτημα για το ποια μέθοδος είναι περισσότερο αποδοτική και αξιόπιστη. Ο σκοπός κάθε αρχιτεκτονικής είναι να καταλήξει στο βέλτιστο, ικανοποιώντας τόσο τους περιορισμούς του προβλήματος όσο και τη συμβατότητα μεταξύ των πεδίων. Σημαντική είναι επίσης η ικανότητα της αρχιτεκτονικής να λύσει το πρόβλημα σε αποδεκτό υπολογιστικό χρόνο. Για τη σύγκριση των διαφόρων αρχιτεκτονικών έχουν προταθεί ορισμένα προβλήματα, μεταξύ των οποίων είναι το πρόβλημα του Sellar [4]. Ενδεικτικά, το XDSM του προβλήματος φαίνεται στο σχήμα 1. Συγκρίνεται η επίδοση των αρχιτεκτονικών Multidisciplinary Feasible (MDF), Individual Discipline Feasible (IDF) και Collaborative Optimization (CO). Οι αρχιτεκτονικές συγκρίνονται ως προς τον αριθμό υπολογισμού κάθε πεδίου που απαιτούν. Τα αποτελέσματα φαίνονται στον πίνακα 1.

Αρχιτεκτονική	MDF	IDF	CO
Υπολογισμοί πεδίου 1	61	30	1210
Υπολογισμοί πεδίου 2	53	24	601
Υπολογισμοί συνάρτησης-στόχου	61	30	168
Επαναλήψεις βελτιστοποιητή	9	7	30
Βελτιστοποιημένη τιμή συνάρτησης-στόχου	3.18339	3.18339	3.1828

Πίνακας 1: Πρόβλημα του Sellar: Σύγκριση διαφόρων αρχιτεκτονικών.



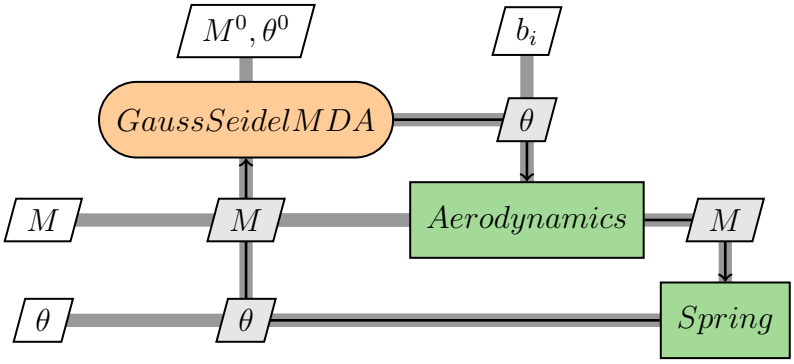
Σχήμα 2: Σύστημα αεροτομής-ελατηρίου: Σχηματική αναπαράσταση.

Σύστημα αεροτομής-ελατηρίου

Μία αεροτομή NACA-0012 τοποθετείται εντός ενός ατριβούς πεδίου ροής, ενώ είναι τοποθετημένο σε στρεπτικό ελατήριο στο $\frac{1}{4}$ της χορδής του. Λόγω της ροής αναπτύσσεται ροπή που τείνει να στρέψει την αεροτομή. Το σύστημα αεροτομή-ελατήριο απεικονίζεται στο σχήμα 2.

Υπάρχουν δύο πεδία, αυτό της αεροδυναμικής και αυτό της δομικής. Η αεροδυναμική περιγράφεται από τις εξισώσεις Euler που επιλύονται από τον οικείο επιλύτη PUMA. Η δομική περιγράφεται από την αναλυτική εξίσωση του ελατηρίου. Το σημείο ισορροπίας του συστήματος, δηλαδή η γωνία θ και η ροπή M στην οποία ισορροπεί, μπορεί να βρεθεί μέσω MDA. Ενδεικτικά, το διάγραμμα XDSM για MDA μέσω της γενικευμένης μεθόδου Gauss-Seidel φαίνεται στην εικόνα 3.

Προκειμένου η αεροτομή να παράγει μία επιθυμητή τιμή άνωσης L^*, χρησιμοποιείται βελτιστοποίηση μορφής. Η συνάρτηση-στόχος που χρησιμοποιείται είναι η $f = 0.5(L - L^*)^2$. Η αεροτομή παραμετροποιείται μέσω κουτιού μορφοποίησης



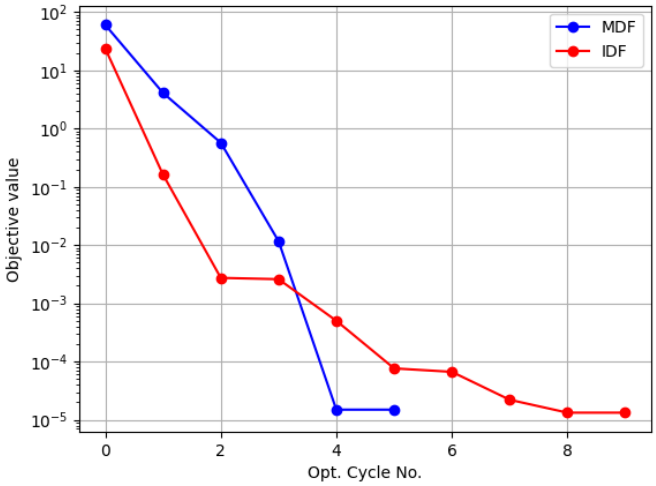
Σχήμα 3: Σύστημα αεροτομής-ελατηρίου: Διάγραμμα XDSM για την πολυτομεακή ανάλυση του συστήματος μέσω της γενικευμένης μεθόδου Gauss-Seidel.

NURBS. Το πρόβλημα βελτιστοποίησης είναι πολυτομεακό, εφόσον πρέπει να συμπεριληφθεί και η επίδραση του ελατηρίου. Χρησιμοποιούνται οι αρχιτεκτονικές MDF και IDF, και συγκρίνονται τα αποτελέσματά τους. Στο σχήμα 4 φαίνεται η πορεία σύγκλισης της τιμής της συνάρτησης-στόχου για τις δύο αρχιτεκτονικές.

Αλληλεπίδραση ρευστού-στερεού σε ελαστικό αγωγό

Ρευστό ρέει εντός διδιάστατου ελαστικού αγωγού, παραμορφώνοντάς τον. Η ροή θεωρείται στρωτή, και η πίεση εισόδου και εξόδου θεωρούνται γνωστές. Ο αγωγός είναι πακτωμένος στα δύο άκρα. Το πρόβλημα είναι συμμετρικό ως προς τον κεντρικό άξονα του αγωγού, και άρα μόνο το μισό μοντελοποιείται, όπως φαίνεται στο σχήμα 5. Για το ρευστό χρησιμοποιούνται οι εξισώσεις Navier-Stokes, οι οποίες επιλύονται μέσω του οικείου επιλύτη PUMA, χωρίς μοντέλο τύρβης (αφού η ροή είναι στρωτή). Για το στερεό τμήμα χρησιμοποιούνται οι εξισώσεις επίπεδης ελαστικότητας, οι οποίες επιλύονται μέσω κώδικα πεπερασμένων στοιχείων, ονόματι SFEM, που αναπτύχθηκε για τις ανάγκες της διπλωματικής εργασίας.

Καθώς το ρευστό ρέει, παραμορφώνει τον αγωγό, με αποτέλεσμα να αλλάζει η ροή. Προκειμένου να βρεθεί η κατάσταση ισορροπίας, χρησιμοποιείται πολυτομεακή



Σχήμα 4: Σύστημα αεροτομής-ελατηρίου: Πορεία σύγκλισης της τιμής της συνάρτησης-στόχου για τις αρχιτεκτονικές MDF και IDF, για το πρόβλημα βελτιστοποίησης μορφής της αεροτομής.

ανάλυση. Ο παραμορφωμένος αγωγός φαίνεται στο σχήμα 6.

Προκειμένου να ελεγχθεί η μέγιστη οριζόντια μετατόπιση του αγωγού εφαρμόζεται βελτιστοποίηση των φυσικών ιδιοτήτων του υλικού του. Το πρόβλημα επιλύεται μέσω της αρχιτεκτονικής MDF. Η σύγκλιση της μεθόδου φαίνεται στο σχήμα 7.

Αεροδομική βελτιστοποίηση της πτέρυγας ONERA M6

Η αεροδομική βελτιστοποίηση πτερύγων αεροσκαφών αποτελεί μία από τις συνηθέστερες εφαρμογές MDAO. Επιλέγεται η πτέρυγα ONERA M6, καθώς χρησιμοποιείται συχνά για την επικύρωση κωδίκων υπολογιστικής ρευστοδυναμικής. Η πτέρυγα τοποθετείται εντός τριδιάστατου, ατριβούς, διηχητικού ροϊκού πεδίου. Χρησιμοποιούνται οι εξισώσεις Euler, που επιλύονται από τον οικείο επιλύτη PUMA. Λόγω των αεροδυναμικών δυνάμεων που δέχεται, η πτέρυγα κάμπτεται κατά το μήκος

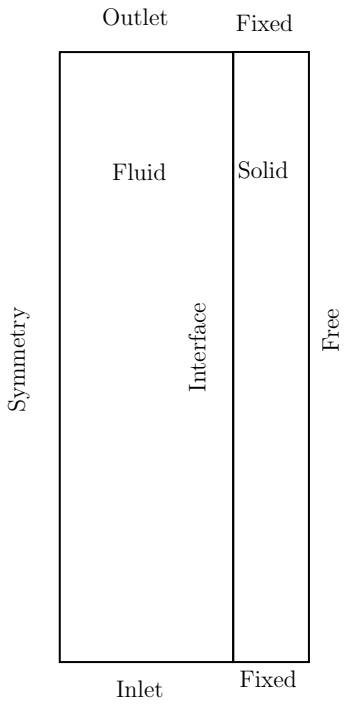
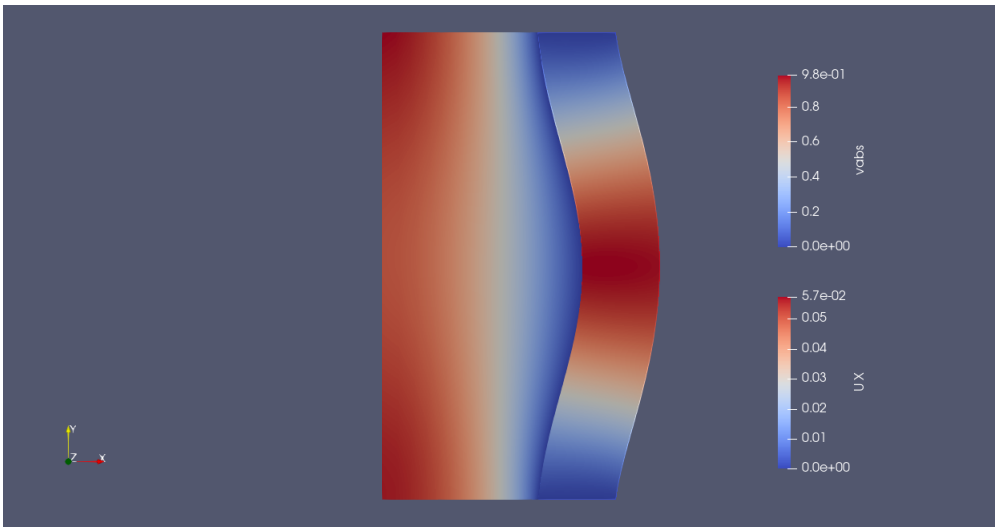
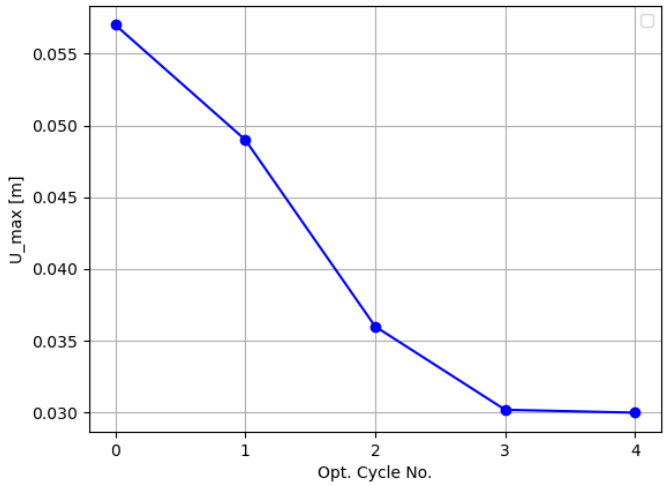


Figure 5: FSI ελαστικού αγωγού: Αναπαράσταση του υπολογιστικού χωρίου.



Σχήμα 6: FSI ελαστικού αγωγού: Ο παραμορφωμένος αγωγός, όπως προκύπτει από την πολυτομεακή ανάλυση. Το ρευστό χρωματίζεται από την κάθετή του ταχύτητα, ενώ το στερεό από την οριζόντια παραμόρφωση.



Σχήμα 7: FSI ελαστικού αγωγού: Πορεία σύγκλισης της τιμής της μέγιστης οριζόντιας μετατόπισης $U_{x,max}$ για την αρχιτεκτονική MDF, για το πρόβλημα βελτιστοποίησης των φυσικών ιδιοτήτων του αγωγού.

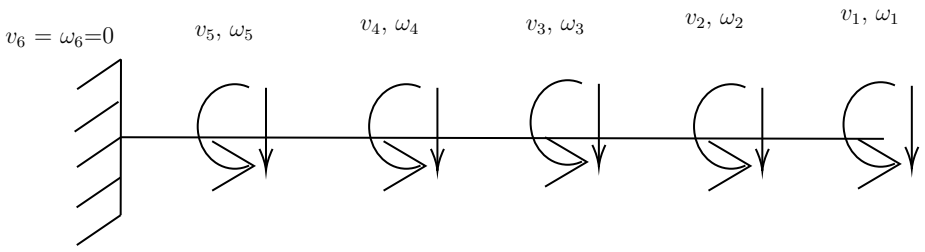
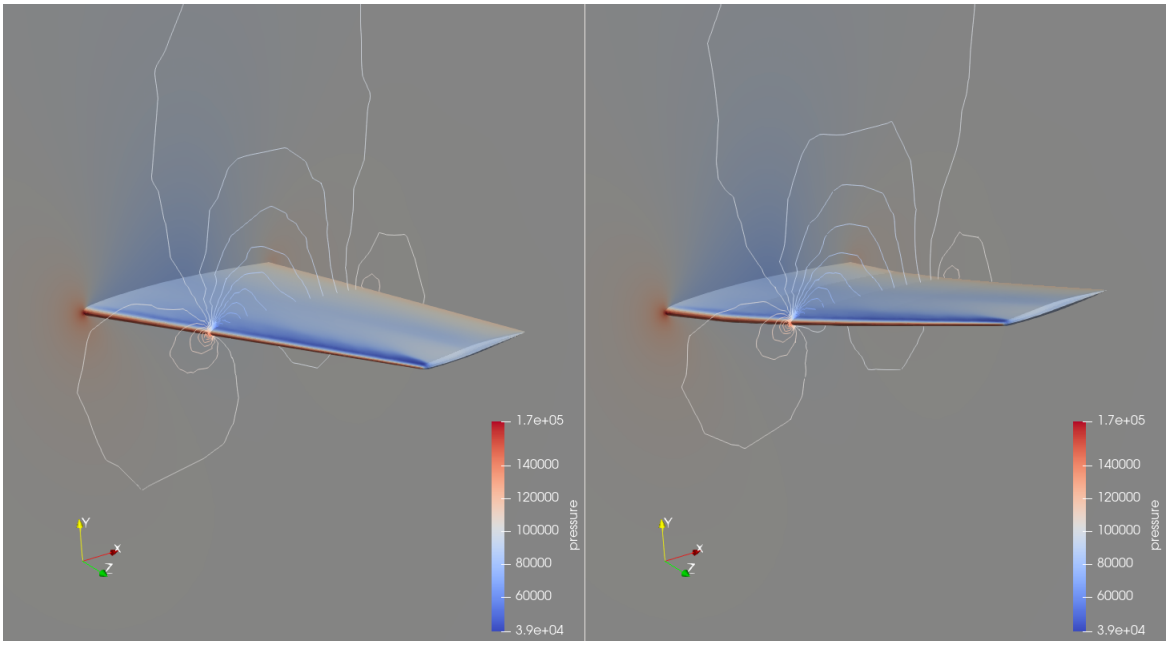


Figure 8: ONERA M6: Δομικό μοντέλο της πτέρυγας

της. Για να μοντελοποιηθεί λοιπόν η δομική της συμπεριφορά, αναπτύσσεται μοντέλο πεπερασμένων στοιχείων με δοκούς. Το μοντέλο οπτικοποιείται στο σχήμα 8. Υπάρχουν δύο βαθμοί ελευθερίας ανά κόμβο, ένας για τη μετατόπιση και ένας για την στροφή του κόμβου.

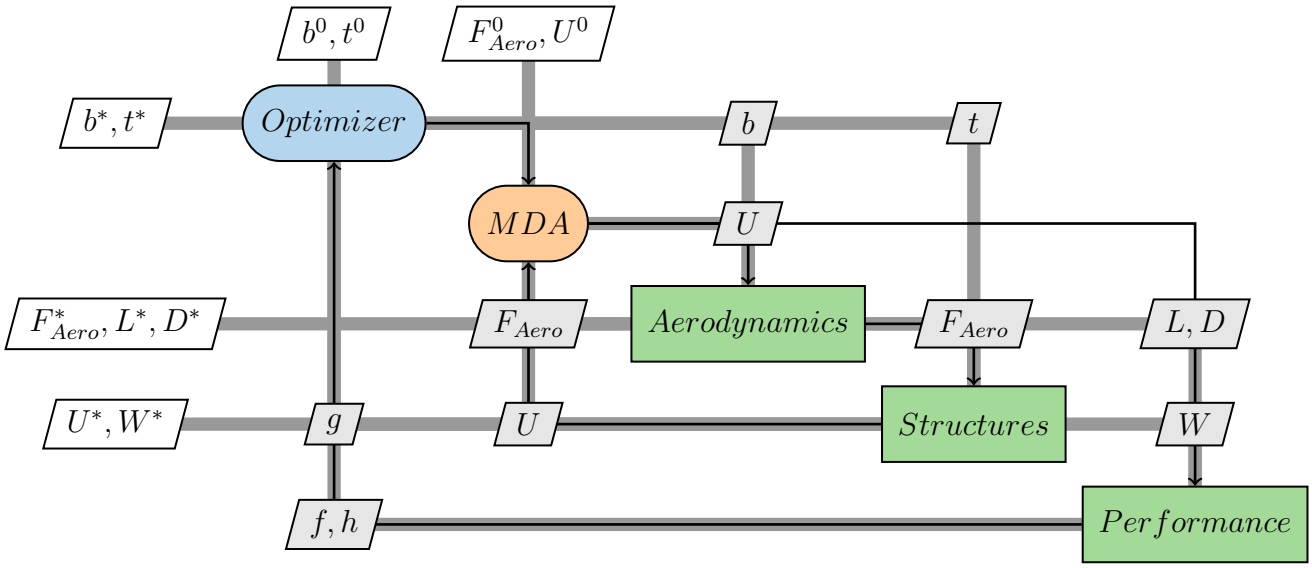
Θεωρείται πως η πτέρυγα δεν είναι μεμονωμένη, αλλά ανήκει σε αεροσκάφος του οποίου το βάρος πρέπει να σηκώσει. Αυτό γίνεται ώστε το πρόβλημα να είναι περισσότερο ρεαλιστικό, χωρίς να αυξηθεί το υπολογιστικό κόστος λόγω αεροδομικής μελέτης ολόκληρου αεροσκάφους. Μέσω πολυτομεακής ανάλυσης υπολογίζονται η παραμόρφωση της πτέρυγας κατά την πτήση, καθώς και τα φορτία που δέχεται. Τα δύο πεδία είναι η αεροδυναμική, και η δομική. Σύγκριση μεταξύ της απαραμόρφωτης και παραμορφωμένης πτέρυγας φαίνεται στο σχήμα 9.

Προκειμένου να βελτιωθεί η επίδοση της πτέρυγας εφαρμόζεται βελτιστοποίηση.

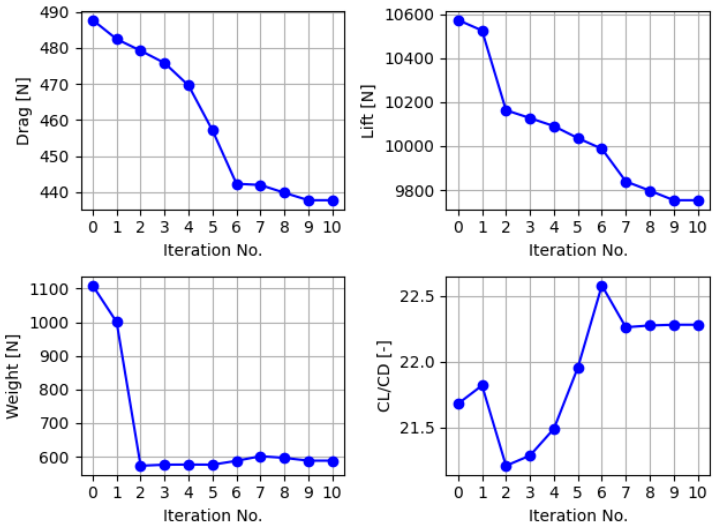


Σχήμα 9: ONERA M6: Σύγκριση μεταξύ της απαραμόρφωτης (αριστερά) και παραμορφωμένης (δεξιά) πτέρυγας.

Μεταβάλλεται τόσο το σχήμα της πτέρυγας, όσο και η δομή της, μέσω αλλαγής του πάχους των δομικών στοιχείων. Εξετάζονται δύο συναρτήσεις-στόχου. Η πρώτη είναι ένα ζυγισμένο άθροισμά του βάρους και της οπισθέλκουσας, ενώ η δεύτερη είναι ο λόγος άνωσης προς οπισθέλκουσα. Για την επίλυση του προβλήματος γίνεται χρήση της αρχιτεκτονικής MDF, για την οποία το διάγραμμα XDSM φαίνεται στο σχήμα 10. Ενδεικτικά, στο σχήμα 11 παρουσιάζεται η πορεία σύγκλισης της οπισθέλκουσας, της άνωσης, του δομικού βάρους και του λόγου άνωσης προς οπισθέλκουσα της πτέρυγας, για την πρώτη συνάρτηση-στόχο. Η αρχιτεκτονική MDF καταφέρνει να βελτιώσει την επίδοση της πτέρυγας και με τις δύο συναρτήσεις-στόχου, ωστόσο η δεύτερη συνάρτηση-στόχος (δηλαδή ο λόγος άνωσης προς οπισθέλκουσας) οδηγεί σε καλύτερα αποτελέσματα.



Σχήμα 10: ONERA M6: Διάγραμμα XDSM για το πρόβλημα της αεροδομικής βελτιστοποίησης της πτέρυγας μέσω της αρχιτεκτονικής MDF.



Σχήμα 11: ONERA M6: Πορεία σύγκλισης της οπισθέλκουσας (πάνω αριστερά), της άνωσης (πάνω δεξιά), του δομικού βάρους (κάτω αριστερά) και του λόγου άνωσης προς οπισθέλκουσα (κάτω δεξιά) της πτέρυγας. Η συνάρτηση-στόχος είναι το ζυγισμένο άθροισμα βάρους και οπισθέλκουσας.

Ανακεφαλαίωση και προτάσεις για μελλοντική μελέτη

Ο σκοπός αυτής της διπλωματικής εργασίας ήταν να υλοποιήσει τη μεθοδολογία MDAO και να την εφαρμόσει σε διάφορα προβλήματα. Μετά την παρουσίαση της θεωρίας, έγινε σύγκριση της επίδοσης τριών αρχιτεκτονικών MDO σε δύο αναλυτικά προβλήματα. Έπειτα, επιλύθηκαν δύο προβλήματα αλληλεπίδρασης ρευστού-στερεού και το πρόβλημα αεροδομικής βελτιστοποίησης πτέρυγας αεροσκάφους. Συνολικά, η μεθοδολογία MDAO παρέχει έναν αξιόπιστο και αποτελεσματικό τρόπο διαχείρισης πεπλεγμένων συστημάτων. Ωστόσο, το πεδίο είναι σχετικά νέο και υπάρχουν περιθώρια για βελτίωση και μελέτη, όπως η υλοποίηση και εφαρμογή της υπολειμματικής μορφής για πολυτομεακά μοντέλα.

Βιβλιογραφία

[1] R. Braun. *Collaborative optimization: an architecture for large-scale distributed design*. PhD thesis, Standford University, 1996.

[2] R. Haftka. Automated procedure for design of wing structures to satisfy strength and flutter requirements. Technical report, NASA Langley research center, 1973.

[3] R. Lewis, G. Shubin, E. Cramer, J. Dennis, P. Frank, R. Michael, L. Gregory, and R. Shubin. Problem formulation for multidisciplinary optimization. *SIAM Journal on Optimization*, 4, 02 1997.

[4] R. Sellar, S.M. Batill, and J. Renaud. Response surface based, concurrent subspace optimization for multidisciplinary system design. 1996.

[5] J. Sobieski. Sensitivity of complex, internally coupled systems. *AIAA Journal*, 28(1):153–160, 1990.

[6] J. Sobieski, T. Altus, M. Phillips, and R. Sandusky. Bilevel integrated system synthesis for concurrent and distributed processing. *AIAA Journal*, 41(10):1996–2003, 2003.