

Component-Based Aerodynamic Shape Optimization using Overset Meshes

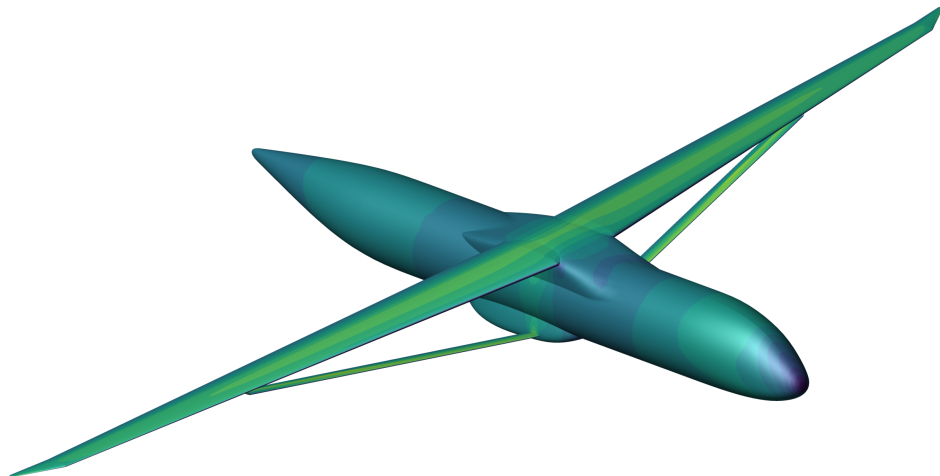
by

Ney R. Secco

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Aerospace Engineering)
in the University of Michigan
2018

Doctoral Committee:

Professor Joaquim R. R. A. Martins, Chair
Assistant Professor Jesse S. Capecelatro
Professor Carlos E. S. Cesnik
Associate Professor Karthik Duraisamy



Ney R. Secco

neysecco@umich.edu

ORCID iD: 0000-0001-6799-2452

© Ney R. Secco 2018

To my dear wife, Claudia.

ACKNOWLEDGEMENTS

This work would not be possible without the help of many people that assisted and inspired me along the way.

First I thank my advisor, Prof. Martins, for giving me opportunity to join the MDO Lab. His mentorship made me a better researcher and a better person. I learned valuable lessons that I will take for my entire life.

I also acknowledge Prof. Jesse Capecelatro, Prof. Carlos Cesnik, and Prof. Karthik Duraisamy for serving as committee members. I appreciate your time considering this work and providing helpful suggestions.

MDO Lab members were also essential for the accomplishment of this work. Dr. Gaetan Kenway's work was the foundation for what I developed in this thesis. Dr. John Hwang showed me the first steps on using the MDO Lab codes and also provided helpful discussions. John Jasa helped with the implementation of the pySurf module. I am also grateful for meeting great friends in the MDO Lab: Anil Yildirim, Charles Mader, David Burdette, Eirikur Johnson, Josh Anibal, Justin Gray, Nicholas Bons, Song Chen, Shamsheer Chauhan, Sicheng He, and Timothy Brooks.

My wife, Claudia, encouraged me to face the challenge of studying abroad. She also kept me focused during moments of pressure. I am forever thankful for your patience and for being always there whenever I needed someone to count on. I thank my family for being always supportive in every step of my life and for giving me confidence and strength to overcome difficulties.

I express my gratitude towards Prof. Bento Mattos, my advisor during the undergraduate and Masters course at the Instituto Tecnológico de Aeronáutica (ITA), since he played a vital role in introducing me to Aircraft Design and MDO. I also thank the Brazilian Air Force for providing the necessary funding for this work.

TABLE OF CONTENTS

Dedication	ii
ACKNOWLEDGMENTS	iii
List of Figures	vii
List of Tables	xii
List of Abbreviations	xiii
Abstract	xv
Chapter	
1 Introduction	1
1.1 High-fidelity aircraft design optimization	3
1.2 Aerodynamic shape optimization with overset meshes	5
1.3 Geometry and mesh manipulation methods	9
1.4 Aerodynamic shape optimization of junctions	12
1.5 Thesis objectives	13
1.6 Thesis outline	14
2 Component-based parametrization	16
2.1 Collar mesh generation overview	16
2.2 Intersection computation	17
2.3 Hyperbolic surface mesh generation	19
2.4 Automatic differentiation	20
2.4.1 Projection subroutine	21
2.4.2 Hyperbolic surface marching	22
2.4.3 Intersection computation	23
2.4.4 Tests for derivative validation	25
2.4.5 Gradient verification: CRM case	26
3 Optimization Framework	28
3.1 Geometry modeler—pyGeo	30
3.2 Collar mesh generator—pySurf	31
3.3 Volume mesh generator—pyHyp	31

3.4	Volume mesh deformation—pyWarp	33
3.5	CFD solver—ADflow	34
3.6	Optimizer—SNOPT	36
3.7	Derivative computation throughout the framework	36
3.8	Noise issues in the functions of interest	40
	3.8.1 Effects of numerical noise on a univariate optimization problem	41
	3.8.2 Identification of the noise source	42
4	Wing-body junction optimization	45
	4.1 Geometric design variables and constraints	46
	4.2 Problem setup	48
	4.3 Baseline configuration studies	50
	4.4 Optimization results	54
	4.5 Summary	64
5	Strut-braced wing optimization	65
	5.1 Optimization problem definition	67
	5.2 Wing and strut optimization	71
	5.3 Junction optimization according to PADRI 2017 guidelines	76
	5.4 Summary	80
6	Concluding remarks	81
	6.1 Conclusions	81
	6.2 Contributions	83
	6.3 Recommendations for future work	85
	Appendices	87
	Bibliography	100

LIST OF FIGURES

1.1	Comparison of shock waves (top) and trailing edge separation (bottom) between the baseline and optimized configurations of the TBW optimization using multiblock structured meshes. The previous optimization could not completely remove shocks and separation near junctions.	2
1.2	Skewed cells on the surface mesh (black) and volume mesh (red) of a truss-braced wing configuration using a patched multiblock mesh.	6
1.3	Hole cutting of a cylinder mesh overlapping a Cartesian background mesh. . .	7
1.4	Collar meshes are necessary to ensure valid cells along intersections.	8
2.1	Steps for generating collar mesh for a wing-fuselage junction. pySurf is responsible for steps a–c. The mesh extrusion module used in step d (pyHyp) is covered in Sec. 3.3.	17
2.2	Additional features implemented in the collar marching scheme to improve mesh quality for CFD analyses.	20
2.3	Representation of a generic subroutine and its corresponding AD versions. . .	20
2.4	Intersection between triangles of two components.	24
2.5	Low-wing configuration shifted to high-wing configuration with automatically generated collar meshes for the CRM geometry.	27
2.6	Aft view of the CRM wing-body junction. The left figure shows the design variable definition and the tracking of the trailing edge intersection point. The figure on the right shows how surface mesh points move as the wing is translated. The red arrows indicate the AD-computed derivatives for the trailing edge intersection position (left) and surface mesh points (right) as we vertically translate the wing. The computed gradients for triangulated surface intersections are smooth and accurate enough for gradient-based optimization.	27
3.1	XDSM of the overall optimization framework. The initialization step generates the reference volume nodes for the mesh deformation.	30
3.2	Hyperbolic extrusion of the surface meshes into volume meshes for a wing-body configuration.	32
3.3	Symmetry plane showing the fuselage near field mesh (red) the wing near field mesh (blue) and the background mesh (black) made by a Cartesian block and an O-mesh.	33
3.4	Application of the mesh deformation algorithm for a wing deflection case. . . .	33

3.5	Zipper meshes are used to fill the gaps among overlapped meshes for force integration. They are not employed during the flow solution.	35
3.6	Subset of the framework subroutines required for the computation of CFD residuals (\mathbf{R}) and functions of interest (f).	39
3.7	Derivative backpropagation chain. The modules represented here are the reverse counterparts of the modules shown in Fig. 3.6. The plus signs indicate that the derivative seed should be accumulated from multiple sources. Subscripts ‘w’, ‘f’, ‘c’, and ‘aero’ indicate that quantities refer to wing, fuselage, collar, and flow conditions, respectively.	39
3.8	Transonic configuration used for wing translation study. The y_{wing} design variable controls the vertical displacement of the wing.	40
3.9	Drag variations due to the vertical translation of the wing of the transonic airplane configuration of Fig. 3.12. The red arrows represent gradients. We observe noise at small steps (on the right), but the gradients are still consistent with the overall trend.	41
3.10	Effects of the CFD mesh refinement on the noise levels for the same wing translation problem shown in Fig. 3.12. The noise levels are smaller when we use finer versions of the CFD structured meshes.	41
3.11	Optimization path for the wing translation problem. The plot on the right is a zoomed-in view showing the last iterations of the optimizer. The optimizer is trapped in a valley caused by the noise, but it manages to improve the baseline design.	42
3.12	Transonic configuration used to study wing translation. The design variable Δx_{wing} controls the horizontal displacement of the wing.	44
3.13	Variations in drag for different horizontal wing positions normalized by the fuselage diameter (D_{fuse}). The right plot shows drag variations for small displacements. The noise caused by changes in overset connectivity decreases as the mesh is refined.	44
4.1	Wing FFD showing the control points (red dots), which are manipulated by the twist and vertical displacement variables.	46
4.2	Pairs of points used to define thickness constraints. The distance between the pair of points cannot decrease during the optimization.	47
4.3	Fuselage FFD showing the free control points as blue dots. The other control points remain fixed to guarantee C_1 continuity within the undeformed region.	47
4.4	Structured surface meshes used for the volume mesh extrusion. The primary component meshes (wing and fuselage) are shown in black, and the automatically generated collar mesh is shown in red.	50
4.5	Wing-body junction flow patterns predicted by ADflow for the DLR-F6 configuration. The baseline geometry shows separation on the junction trailing edge (left), while the addition of the FX2B fairing removes it (right). This is the same trend observed in DPW3.	50
4.6	Effect of grid refinement and turbulence model variant in the recirculation bubble size.	52

4.7	Drag convergence study for the DLR-F6 configurations. N represents the number of cell in the CFD mesh. Blue colors refer to the baseline DLR-F6 configurations and red colors refer to the DLR-F6-FX2B configuration.	52
4.8	Effect of refinement of triangulated surface over CFD results. The coarse triangularization is used for optimization.	53
4.9	Fairing-only optimization history. Some normal displacement design variables went to the lower boundary of 0.04 m.	54
4.10	Optimized fairing for the fairing-only optimization (Problem F).	55
4.11	Wing-body junction flow before and after the fairing optimization (Problem F). Red regions indicate reversed flow. The redesigned fairing reduces the recirculation bubble.	55
4.12	Comparison of lift and drag distributions between the baseline (B) and the fairing-optimized configuration (F).	56
4.13	Relative computational time of the tasks performed during and optimization iteration involving an adjoint solution. The average time of the iteration is of 85 seconds.	57
4.14	Progression of the optimized drag value due to the additional active design variables. Drag decreases as we add more degrees of freedom to the optimization.	57
4.15	Rear views comparing the fairing sizes obtained for different optimizations. The fairing gets smaller as the optimizer gets more control over the wing properties.	58
4.16	Contour plot showing the distance that the fuselage surface moved during optimization. The fuselage surface moves a relatively large amount when the optimizer can only control the fairing.	58
4.17	Optimization histories for the F+T and F+T+S problems. The fairing design variables do not reach the upper bound in either problem.	59
4.18	Comparison between lift and drag distributions for all fairing optimizations. The inclusion of twist variables (T) allows the optimizer to achieve more efficient lift distributions, which are closer to an elliptical one.	60
4.19	Pressure coefficient slices of all optimized configurations. The optimizer designs wings with smoother pressure distributions as we activate wing shape variables (S). The shock on the upper wing surface is also removed by the shape variables. Even though Problems T+S and F+T+S have different design spaces, they show similar airfoils and pressure distributions since they converged to practically the same wing design.	61
4.20	Wing-body junction trailing edge after every optimization problem. Red regions indicate reversed flow. The redesigned fairings eliminate the recirculation bubble in all problems. The wing shape variables achieve smoother chordwise pressure distributions.	62
4.21	Comparison between the optimized configuration with and without fairing design variables. Optimization without the fairing design variables still shows trailing edge separation region (red).	62

4.22	Comparison between lift and drag distributions of the twist and shape optimized configurations with (F+T+S) and without (T+S) fairing variables. The resulting twist and airfoil distributions get progressively more similar as we move towards the tip. The F+T+S configuration has a smaller drag in this region as well, which leads to the improvement seen in Fig. 4.14.	63
4.23	Mesh refinement study for baseline (B) and optimized configurations (F and F+T+S). Dashed lines represent continuum estimates. The optimized design improvements are still present at finer mesh levels.	63
5.1	Baseline SBW configuration of the PADRI 2017 workshop. Views are not in the same scale.	67
5.2	Structured surfaces meshes of the primary components of the aircraft. The O-grids near intersections increase the cell density to facilitate the overset hole cutting process.	69
5.3	Triangulated surfaces used for intersection detection and collar mesh generation.	69
5.4	FFD boxes of the primary components whose shape will be optimized. The dots represent the FFD box control points.	70
5.5	Points of the wing (blue) and strut (red) where thickness constraints are enforced. The distances between the pair of points cannot go below their initial value.	71
5.6	Optimization history of the SBW optimization. The drag decreases by 33 counts for the same lift coefficient.	72
5.7	Comparison of shock and pressure distributions between the baseline (left) and optimized (right) designs. The strut is shown at a different scale.	73
5.8	Separation on the trailing edge of the wing-strut junction before (top) and after (bottom) the optimization.	73
5.9	Spanwise distribution of lift, drag and twist for the baseline (dashed lines) and optimized configuration (solid lines). The strut generates downward lift, and the inboard region of the wing increases its lift distribution to compensate for that, yielding an overall elliptical lift distribution.	74
5.10	Component-wise lift distribution. The lift is normalized by the C_L constraint.	74
5.11	Cross-sectional slices of the wing and strut and corresponding pressure distributions for the baseline (dashed lines) and optimized (solid lines) configurations.	75
5.12	Cross-sectional slices of the vertical segment of the strut for the baseline (dashed lines) and optimized (solid lines) configurations.	76
5.13	Optimization history of the SBW optimization for the PADRI 2017 guidelines. The drag decreases by 14 counts while maintaining the same lift coefficient of the baseline configuration (red line).	77
5.14	Discrepancies between the CFD surface nodes of the baseline (black) and optimized (red) configurations only occur on the lower surface of the wing and on the strut surface within the specified spanwise range.	78
5.15	Comparison of shock waves between the baseline (left) and optimized (right) designs for the PADRI optimization case. Shocks still remain outside of the optimized region.	78

5.16	Rear view of the wing-strut intersection showing separation regions (blue). The optimizer manages to remove the trailing edge separation only within the spanwise range where the design variables are active.	78
5.17	Comparison among the cross-sectional slices of the wing and strut and corresponding pressure distributions for the baseline configuration (red dashed lines), the fully optimized configuration (red solid lines), and the optimized configuration for the PADRI guidelines (black). The optimized shape for the PADRI 2017 case has more twist to compensate the fixed twist angles of the wing and the remainder of the strut.	79
6.1	Execution order of the MACH framework modules during each optimization iteration (black lines) and summary of the modifications done to these modules as part of this thesis (in red). The dashed lines indicate processes that are only executed in the initialization step of the optimization.	84
6.2	Possible applications for the component-based parametrization technique developed in this thesis.	86
A.1	Meshes marched with different splay factors. The red line is the baseline curve for the marching process.	96
A.2	The use of blending factor ($\nu = 0.5$) avoids highly skewed cells where guide curves are oblique to the intersection line.	98

LIST OF TABLES

4.1	DLR-F6 aerodynamic shape optimization problem.	46
4.2	Identification tags for each optimization problem.	48
4.3	Inputs required by pySurf; these are generated by ICEM CFD using the original IGES representation of the DLR-F6 model.	48
4.4	Mesh levels used for drag convergence study. The maximum y^+ values are computed based on the converged CFD results.	51
4.5	Aerodynamic coefficients obtained for each optimization.	57
5.1	Geometric characteristics and flight conditions used for the baseline SBW configuration analysis.	68
5.2	SBW aerodynamic shape optimization problem.	70
5.3	SBW aerodynamic shape optimization problem after PADRI 2017 guidelines. .	77

LIST OF ABBREVIATIONS

AD algorithmic differentiation

ADT alternating digital tree

API application programming interface

ASO aerodynamic shape optimization

BLI boundary layer ingestion

CAD computer aided design

CDGT conceptual design geometry tools

CFD computational fluid dynamics

CRM common research model

DPW3 Third Drag Prediction Workshop

DPW6 Sixth Drag Prediction Workshop

FFD free form deformation

IDW inverse distance weighting

IHC implicit hole cutting

LAPACK linear algebra package

MACH multidisciplinary design optimization of aircraft configurations with high fidelity

NURBS non uniform rational basis splines

OML outer mold line

PADRI Platform for Aircraft Drag Reduction Innovation

PDE partial differential equations

QCR quadratic constitutive relation

RANS Reynolds-averaged Navier–Stokes

SA Spalart–Allmaras

SNOPT Sparse Nonlinear Optimizer

SBW strut-braced wing

SUGAR Subsonic Ultra-Green Aircraft Research

TBW truss-braced wing

VLM vortex-lattice method

XDSM extended design structure matrix

ABSTRACT

Advances in computational power allow the increase in the fidelity level of analysis tools used in conceptual aircraft design and optimization. These tools not only give more accurate assessments of aircraft efficiency, but also provide insights to improve the performance of next-generation aircraft. Aerodynamic shape optimization involves the inclusion of aerodynamic analysis tools in optimization frameworks to maximize the aerodynamic efficiency of an aircraft configuration via modifications of its outer mold line.

When using CFD-based aerodynamic shape optimization, generating high-quality structured meshes for complex aircraft configurations becomes challenging, especially near junctions. Furthermore, mesh deformation procedures frequently generate negative volume cells when applied to these structured meshes during optimization. Complex geometries can be accurately modeled using overset meshes, whereby multiple high-quality structured meshes corresponding to different aircraft components overlap to model the complete aircraft configuration. However, from the standpoint of geometry manipulation, most methods operate on the entire geometry rather than on separate components, which diminishes the advantages of overset meshes.

Tracking intersections among multiple components is a key challenge in the implementation of component-based geometry manipulation methods. The mesh nodes should also be updated in accordance to the intersection curves.

This thesis addresses this issue by introducing of a geometry module that operates on individual components and uses triangulated surfaces to automatically compute intersections during optimization. A modified hyperbolic mesh marching algorithm is used to

regenerate the overset meshes near intersections. The reverse-mode automatic differentiation is used to compute partial derivatives across this geometry module, so that it fits into an optimization framework that uses a hybrid adjoint method (ADjoint) to efficiently compute gradients for a large number of design variables. Particularities of the automatic differentiation of the geometry module are detailed in this thesis.

By using these automatically updated meshes and the corresponding derivatives, the aerodynamic shape of the DLR-F6 geometry is optimized while allowing changes in the wing-fuselage intersection. Sixteen design variables control the fuselage shape and 128 design variables determine the wing surface. Under transonic flight conditions, the optimization reduces drag by 16 counts (5%) compared with the baseline design.

This approach is also used to minimize drag of the PADRI 2017 strut-braced wing benchmark for a fixed lift constraint at transonic flight conditions. The drag of the optimized configuration is 15% lower than the baseline due to reduction of shocks and separation in the wing-strut junction region. This result is an example where high-fidelity modeling is required to quantify the benefits of a new aircraft configuration and address potential issues during the conceptual design.

The methodologies developed in this work give additional flexibility for geometry and mesh manipulation tools used in aerodynamic shape optimization frameworks. This extends the applicability of design optimization tools to provide insights to more complex cases involving multiple components, including unconventional aircraft configurations.

CHAPTER 1

Introduction

Fuel-burn reduction is one of the main drivers in aircraft design due to environmental and economical reasons. Improvements in computational power allow designers to increase the fidelity level of aerodynamic and structural analysis used during aircraft conceptual design. These tools can be assembled in optimization frameworks to reduce fuel burn and to increase the profitability of future aircraft designs.

The use of high-fidelity analysis tools is also paramount for the design of unconventional aircraft configurations. An important feature of these tools is that they simulate additional phenomena compared to low-fidelity tools. For instance, the vortex-lattice method (VLM) cannot capture shock waves nor flow separation [1], while these features can be quantified in Reynolds-averaged Navier–Stokes (RANS) simulations. The results of the former can be corrected via regressions based on experiments and historical data for conceptual design applications. However, the lack of information for unconventional aircraft prevents the use of this approach for this type of aircraft. In this situation, the use of high-fidelity analysis poses as a cost-effective solution to gather data with enough accuracy for conceptual design purposes.

High-fidelity analysis also considers the entire outer-mold line of the airplane instead of the simplified representations used in low-fidelity tools. For instance, wings are condensed to a single surface following its mean camber line for the VLM, while the actual wing shape (upper surface, lower surface, and trailing edge) can be modeled in RANS analysis. As shapes become more detailed, we need more sophisticated methods to parametrize and manipulate these shapes.

Multiple examples of Euler- and RANS-based design optimizations are reported in the literature. These techniques are usually used to optimize relatively simple geometries, such as airfoils [2–4] or isolated wings [5–8].

For complex configurations, the design variables usually operate on a limited portion of the geometry [8–10]. For example, Merle et al. [10] optimized a conventional aircraft

configuration subject to trim constraints. Even though they simulated the aerodynamics of the entire aircraft configuration (wing, fuselage, tail, nacelle, and pylon), the shape design variables only modified the wing. In addition, they had to damp wing shape deformations near the wing-fuselage intersection to avoid discontinuities in the surface mesh. These simplifying assumptions limited the space of possible geometries and design variables, such as the wing mounting angle [9].

Ivaldi et al. [11] optimized a truss-braced wing (TBW) configuration with RANS analysis. This is a relatively complex configuration due to the presence of interconnected components. The limitations of the geometry manipulation tool used in this work prevented the definition of design variables near junctions. Therefore, the optimized configuration still showed shock waves and flow separation in these regions (Fig. 1.1).

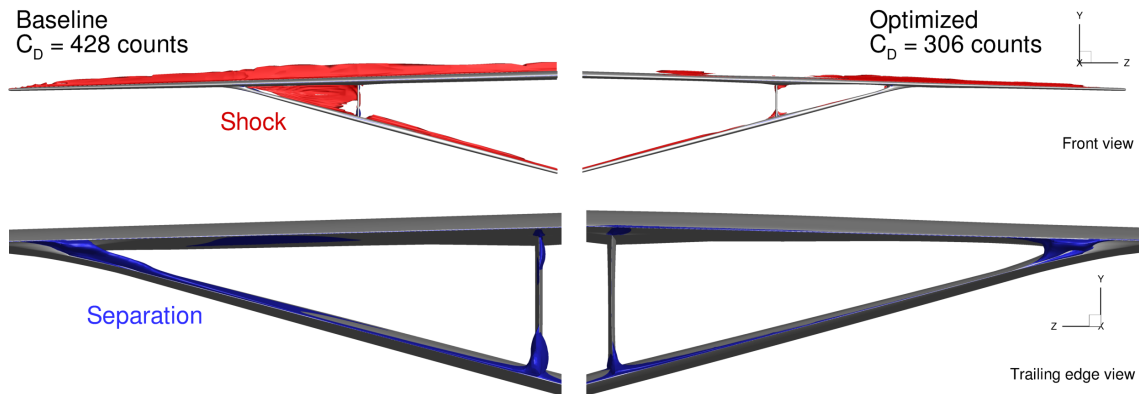


Figure 1.1: Comparison of shock waves (top) and trailing edge separation (bottom) between the baseline and optimized configurations of the TBW optimization using multi-block structured meshes. The previous optimization could not completely remove shocks and separation near junctions.

The challenges that hindered the optimizations discussed above are twofold: first, the geometry manipulation tools operated on the entire configuration at once instead of using component subdivision information to simplify the definition of the design variables. Second, the optimization frameworks could not track changes in the intersections among components to appropriately update the meshes used for aerodynamic analysis. Therefore, the potential of optimization methodologies to gain design insights was not fully explored, especially regarding junction design and simultaneous optimization of multiple components.

This impacts the design optimization of unconventional aircraft designs that rely on efficient interactions among multiple components. For instance, the aerodynamic efficiency of the high-aspect ratio wing of a TBW configuration may be hampered by the interference drag caused by the additional components. The tail-cone thruster configuration also

requires synergy among its components [12–14], since the position of the vertical tail and the rear fuselage shape interfere on the inlet properties of the rear fan.

This thesis focuses on expanding current geometry and mesh manipulation tools to facilitate the use of aerodynamic shape optimization (ASO) methodologies for complex configurations. This includes tracking intersections among components within the optimization process and updating the meshes for the new shape. This thesis also demonstrates that these methods can be differentiated and integrated in gradient-based optimization frameworks.

1.1 High-fidelity aircraft design optimization

Aircraft design is a multidisciplinary problem with a large number of degrees of freedom. Covering this design space solely based on experimental and flight data would be extremely expensive. Therefore, aircraft designers resort to computational tools to prospect most of the design space and narrow down a handful of configurations for posterior design phases, while experiments are used to complement and verify results from the computational framework [15]. Once the execution of these computational tools are streamlined for automatic evaluation of multiple aircraft designs, optimization becomes possible.

Optimization algorithms can be classified in two groups: gradient-free and gradient-based algorithms. Zingg et al. [16] and Yu et al. [17] compare these two approaches for aerodynamic shape optimization applications. Gradient-free optimization algorithms need access only to the inputs and outputs of the design functions (objectives and constraints), so the analysis framework can be used as is, without any additional implementation. Examples of gradient-free algorithms are genetic algorithms [18], particle swarm optimization [19], and the Nelder-Mead simplex [20]. However, the number of function evaluations necessary to converge an optimization problem to a given tolerance dramatically increases with the number of design variables [21], making this approach unfeasible for cases with expensive function evaluations and more than few dozen design variables.

Gradient-based optimization algorithms use gradients of the functions of interest to direct the search for the optimum design point [22]. The convergence rate of these algorithms has a weaker dependence on the number of design variables, making them suitable for optimization problems with hundreds of design variables, which is usually the case for ASO problems. On the other hand, the efficient computation of gradients is not a straightforward task, and it usually requires additional implementation efforts. Peter and Dwight [23] provide a detailed review of gradient computation methods applied to ASO.

Finite difference is the simplest method to compute sensitivities since it requires no

modification to the analysis code used to compute the functions of interest. However, the step size for best accuracy is problem-dependent and also bounded by truncation and subtractive cancellation errors [24]. The complex-step method [25] overcomes the subtractive cancellation issues, thus giving machine-precision-accurate derivatives, but it suffers from the fact that the number of function evaluations required to compute gradients scales linearly with the number of design variables, just like finite difference.

Algorithmic differentiation (AD) is an alternative method to compute gradients in which the chain rule of derivatives is applied line by line of the analysis code [24], either via operator overloading or source code transformation. The analysis code can be differentiated in two modes: the forward AD mode, in which the chain rule products are performed from inputs to outputs, and the reverse AD mode, in which the chain rule products are constructed from outputs to inputs.

Both methods allow the computation of the Jacobian matrix that correlates inputs (design variables) and outputs (functions of interest) of the analysis code. However, each call to the forward AD code provides a column of the Jacobian matrix, while a call to the reverse AD code provides a row of the Jacobian matrix. Thus, the number of total calls of the forward AD code to assemble the full Jacobian matrix is equal to the number of design variables (similarly to finite difference and complex-step), while the total number of calls for the reverse AD code is equal to the number of functions of interest [26]. Since the number of design variables in ASO problems is higher than the number of functions of interest, the reverse AD mode is suitable for efficient gradient computation [23].

The reverse mode AD requires the storage of intermediate values of variables used throughout the code. For instance, in the case of iterative solvers used in computational fluid dynamics (CFD), this amounts to storing the flow state variables of every iteration, leading to prohibitive memory requirements. The discrete adjoint method circumvents this issue since it allows derivative computations across the CFD module based solely on the converged flow state variables.

The discrete adjoint equation is a linear system whose elements are partial derivatives of the CFD residuals and function of interest with respect to the flow state variables. These partial derivatives can be computed using reverse AD, and only the residual evaluation function of the CFD solver needs to be differentiated, leaving the iterative methods outside of the reverse AD chain. This combination of reverse AD and the discrete adjoint method is called the hybrid adjoint method (or ADjoint) [27]. The implementation of the hybrid adjoint method will be further discussed in Chapter 3.

Multiple applications of the hybrid adjoint method are present in the literature for different fidelity levels of aerodynamic and structural analysis. Elham and van Tooren [28]

use the hybrid adjoint method for a VLM code. Mader et al. [27] describe the hybrid adjoint method applied to the Euler equations. Brezillon and Dwight [29] use the adjoint formulation for a RANS solver, but turbulence model variables were assumed as constants in the differentiation process since some partial derivatives were computed by hand or with finite-differences. Lyu et al. [30] include the turbulence model in the AD process to use the hybrid adjoint method with RANS equations while including turbulence model sensitivity. Dumont and Méheut [8] show another example of discrete adjoint application to RANS equations in which they use a time marching scheme to converge the adjoint system.

Burdette [31] demonstrated that an aircraft optimized based on Euler equations and semi-empirical regressions for viscous drag estimation performs poorly when analyzed with RANS equations, justifying the efforts in increasing the fidelity level of the analysis code used in aircraft design optimization. This is specially important to obtain more accurate performance estimates of unconventional aircraft designs, provided there are appropriate tools to represent and manipulate their geometries and meshes while accounting for CFD requirements.

1.2 Aerodynamic shape optimization with overset meshes

CFD requires the discretization of the flow domain into meshes for aerodynamic analysis. Meshes can be classified into two main groups: structured meshes and unstructured meshes.

Structured multiblock meshes are more efficient from the computational point of view since the memory layout already determines the cell connectivity, facilitating the implementation of logical loops [32]. Multiblock meshes can be easily generated for simple geometries but are cumbersome for models with multiple complex features. Mesh topology restrictions may lead to highly skewed cells, as shown in Fig. 1.2 for the case of a wing-strut junction of the TBW configuration. Furthermore, mesh deformation procedures used during the optimization generally reduce mesh quality, severely limiting the range of possible deformations if the optimization starts with an already low quality mesh. Low quality meshes are also prone to the generation of negative volume cells during the mesh deformation process.

The generation of unstructured meshes, on the other hand, is easier to automate even for complex geometries [33]. Nevertheless, the drawback in computational performance becomes significant for ASO applications, as they require a large number of CFD evaluations. Drag prediction benchmarks for CFD simulations also show that unstructured meshes need higher resolution (from 2 to 4 times more elements) than an equivalent structured mesh to achieve the same discretization error [34, 35].

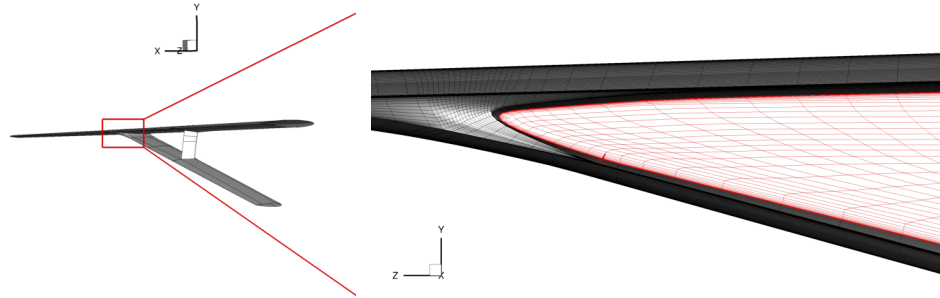


Figure 1.2: Skewed cells on the surface mesh (black) and volume mesh (red) of a truss-braced wing configuration using a patched multiblock mesh.

CFD using overset meshes [36] has the potential to overcome this problem because it gives more flexibility to the generation of structured meshes. Overset CFD solvers allow multiple meshes to overlap in space, and on every CFD iteration they interpolate flow state variables among overlapping meshes.

Complex configurations can be subdivided into components whose meshes can be independently generated as there are no boundary-matching restrictions. These dedicated meshes can also be automatically generated by using, for instance, hyperbolic mesh generation algorithms [37, 38]. In addition to facilitating mesh generation, another advantage of the overset meshes is that, because they are locally composed of multiple structured blocks, it retains the advantages of this mesh type, such as high-quality cells and computational efficiency, while having a globally unstructured arrangement.

The cells of an overset CFD mesh assume different roles based on how their flow state variables are enforced:

Compute Cells: Active cells that are relevant to the solution as they represent the volume.

The PDEs are enforced on these cells.

Blanked Cells: Non-representative cells that may be inside bodies or overlapped by better quality cells. The flow state variables within these cells are not relevant for the residual computation.

Interpolated (Receiver) Cells: Cells that will interpolate flow state variables from compute cells of other overlapping meshes on every iteration.

The process of classifying cells within these categories is called *hole cutting*. This can be done manually, but it is essential to have an automated hole cutting procedure for optimizations with overset meshes since the overlapped regions may change along the design

iterations. The implicit hole cutting (IHC) technique [39] is one of such automated ways of overset connectivity generation. This methodology preserves smaller cells as compute cells, which are usually close to viscous walls, while larger cells become either interpolate cells or blanked cells. Figure 1.3 shows an example of IHC for a 2D cylinder mesh.

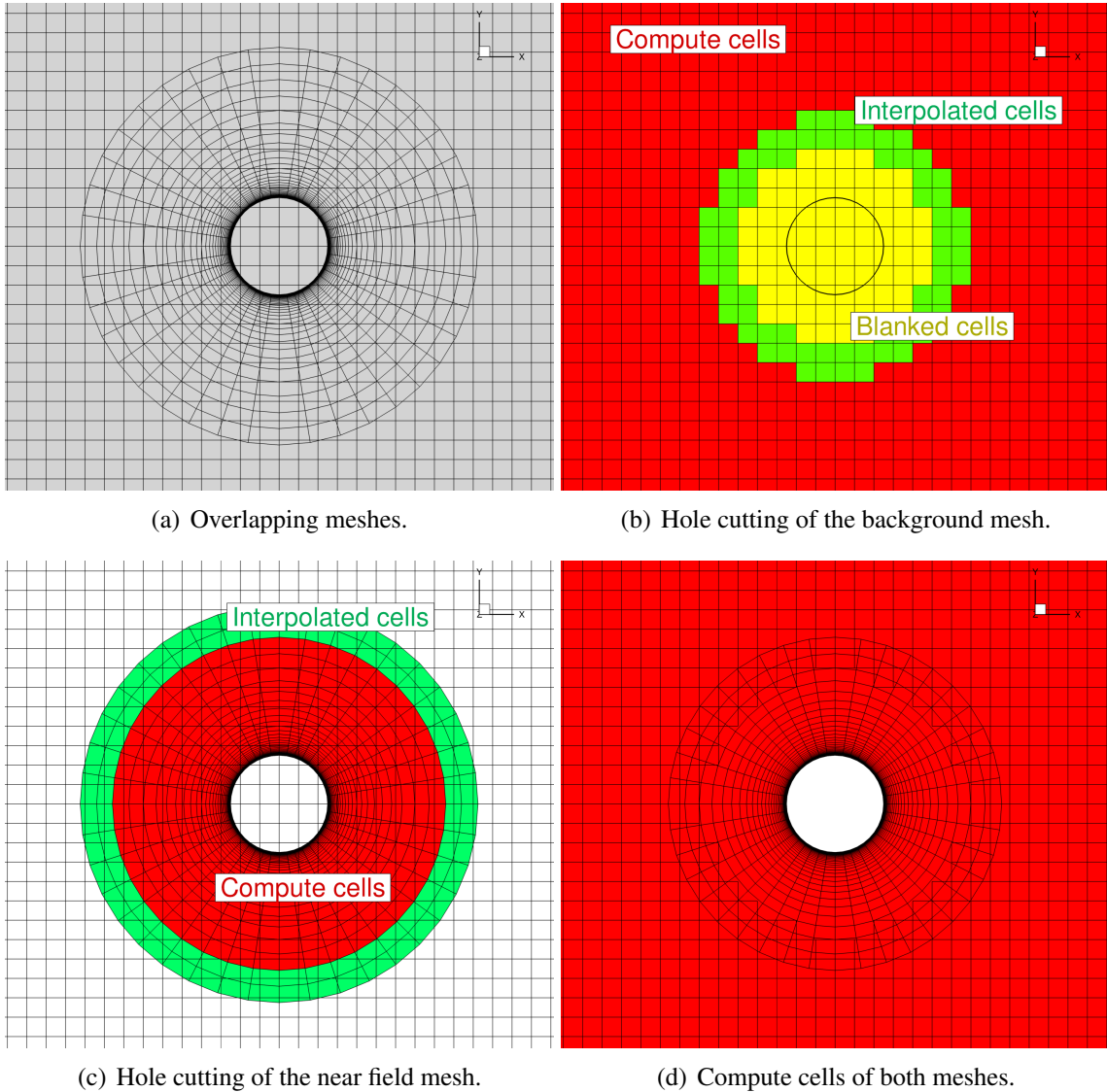


Figure 1.3: Hole cutting of a cylinder mesh overlapping a Cartesian background mesh.

Special attention is necessary when modeling intersections with overset meshes. A valid cell for CFD purposes should be completely inside the flow domain. In other words, if a cell is partially inside a wall, it cannot be classified as a compute cell.

If two intersecting meshes are arbitrarily overlapped, cells from either mesh may be invalid at the intersection, since they may be partially inside the walls of the opposing component (Fig. 1.4). Therefore, an additional overlapping mesh should be introduced

specifically at the junction, called *collar mesh*, to guarantee that the mesh edges represent the intersection curve after the hole cutting process [40].

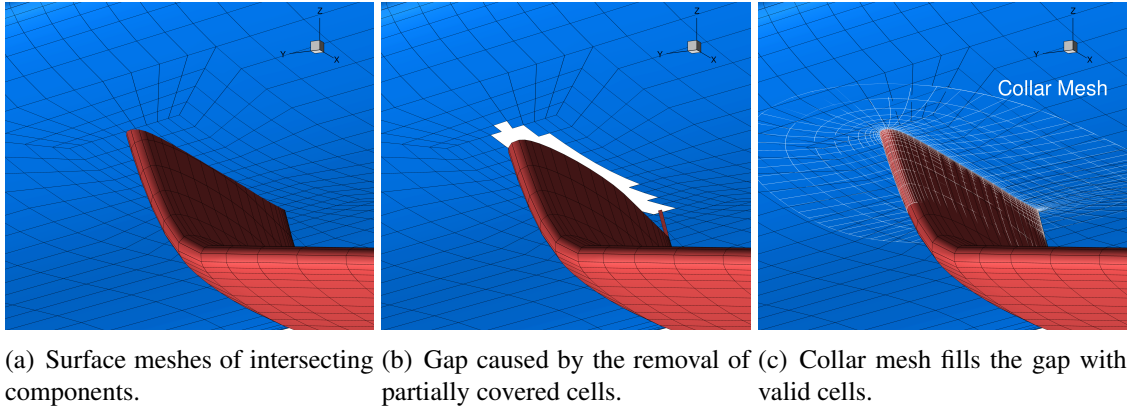


Figure 1.4: Collar meshes are necessary to ensure valid cells along intersections.

Overset meshes have already been used in ASO applications. Liao and Tsai [41] use overset meshes for Euler-based optimization of airfoils using the continuous adjoint approach. Lee and Kim [42] implement the discrete adjoint method for an Euler-based CFD solver and optimized the wing of a wing-body aircraft configuration using overset meshes. Lee et al. [43] later extend their methodology to the RANS equations and the $k-\omega$ turbulence model to minimize distortion in a boundary layer ingestion (BLI) engine inlet. Kenway et al. [44] use overset meshes and the hybrid adjoint method to optimize the wing of the common research model (CRM) aircraft configuration while using RANS analysis with the Spalart–Allmaras (SA) turbulence model.

The impact of changing overset connectivities during the ASO process is not completely described in the literature. Dynamic overset meshes have been used for unsteady aerodynamic and hydrodynamic simulations with no associated optimizations [45–47]. Conversely, the mesh deformation approach used in the optimizations by Lee et al. [43] and Kenway et al. [44] simultaneously deform all overset meshes, what avoids relative changes in interpolation stencils. Changes in overset connectivities are inevitable if components are independently manipulated. In this thesis, the outcomes of variable overset connectivities on the smoothness of functions of interest and their associated gradients is investigated.

In addition, the cited ASO applications with overset meshes did not explore the optimization of intersection regions nor the simultaneous optimization of multiple components of a complex configuration. The main reason is that the geometry and mesh manipulation methods used in these optimization frameworks could not track changes in the intersection line to update collar meshes. This thesis proposes solutions to these issues to unlock the potential of overset meshes for optimizations where geometry components may significantly

shift relative to each other, such as when optimizing the location of the wing-fuselage intersection for a conventional airplane, or when simultaneously modifying components during an optimization, as discussed in Chapter 2.

1.3 Geometry and mesh manipulation methods

The optimization examples mentioned in Sec. 1.1 and Sec. 1.2 need geometry manipulation and mesh manipulation modules in their framework. The geometry manipulation module should generate or modify the aircraft surface representation based on geometric design variables, while the mesh manipulation module receives the updated surface from the geometry module and then modifies the mesh used for CFD analysis.

There are multiple geometry manipulation methods used in ASO. In this section, techniques based on computer aided design (CAD), conceptual design geometry tools (CDGT), and free form deformation (FFD) are discussed. The survey by Samareh [48] has a more thorough review of other geometry manipulation methods.

CAD tools are commonly used during the product development framework in the aerospace industry. These packages include not only the surface definition, but also the procedural steps to construct the geometry such as scaling, intersecting, trimming, among others. Therefore, it would make sense to use the same tool to manipulate geometries during the optimization process. However, since most CAD tools are commercial software, their integration into optimization frameworks is not entirely flexible, since it depends on the application programming interface (API) exposed by the CAD developer. Furthermore, most CAD packages do not compute the derivatives needed by gradient-based optimizers, and users do not have access to the source code to generate AD versions [48].

A recently proposed solution is to base the reference geometry on a parametrized CAD model and use geometry surrogate models to replace the CAD engine in the optimization framework [49]. Provided the surrogate model is smooth and differentiable, this approach provides analytical derivatives across the geometry manipulation module. Conversely, a continuous surrogate model may smooth sharp features of the baseline geometry, such as trailing edges, leading to an inaccurate representation. In addition, a large number of training points may be necessary to obtain a valid surrogate for cases with many design variables and complex geometries. To avoid loss of surface information, the user should also be careful when selecting which surface features from the CAD model are exposed to the surrogate model training.

Importing a geometry from a different CAD tool is not a seamless process since CAD tools may be built over different geometry kernels [50]. This causes discrepancies in the

geometry representation due to the different mathematical formulations and tolerances, especially regarding the definition of intersection curves. Even though B-spline patches have a parametric description, intersections among B-spline patches do not, and the approximations used to describe the intersection curve are different for each CAD software. This results in gaps between intersecting components, hindering the use of automated mesh generation tools [40]. The imported geometry frequently needs to be manually fixed (“healed”) to get a closed surface, for instance, by recomputing intersection curves and redefining certain B-spline patches.

Manufacturing requirements are the main drivers for CAD software development, and these are usually different from mesh generation requirements [51]. For instance, the presence of singularities in B-spline patches is not an issue for manufacturing purposes, but this impairs the use of automatic mesh generation tools. Acceptable tolerances for manufacturing purposes are usually two orders of magnitude above the height of the initial layer of wall cells used for RANS analysis. These discrepancies cause issues for a direct integration between a CAD-based geometry manipulation module and a mesh manipulation module within an optimization framework.

CDGT are an alternative to quickly generate geometries during the conceptual aircraft design studies [50]. These tools use analytical surface descriptions, such as B-spline patches, to build predefined shapes common in aircraft design, such as lifting surfaces and fuselages. OpenVSP [52] and GeoMACH [53] are examples of such tools.

From a designer’s perspective, the use of predefined shapes facilitate the geometry generation procedure compared to a full CAD model since the former has more specific interfaces suited for conceptual-aircraft-design parameters. It is also easier to ensure that the overall geometry is composed by closed surfaces since the components are always built upon these predefined shapes.

Even though some CDGT provide analytical derivatives (such as GeoMACH), their application in optimization frameworks can be hindered by the loss of generality regarding the creation and manipulation of arbitrary shapes. As a consequence, it is challenging to import externally provided geometries. A parameter fitting process, such as least-squares approximation, is necessary to reconstruct the external geometry using the predefined shapes of the CDGT [54]. In addition, the definition of design variables may be limited by the fixed description of the predefined shapes [11].

Another approach to geometry manipulation is by deforming a baseline geometry instead of regenerating the entire geometry for every design point. The FFD is a technique commonly used in computer graphics for soft object animation [48, 55].

The first step is the generation a box of control points that encompasses the surface

that should be manipulated. This FFD box defines a $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ mapping from physical to parametric coordinates (u, v, w) within this volume. Then, the physical coordinates of each surface point inside the FFD box are used in an inverse mapping procedure to compute the corresponding parametric coordinates.

Once the physical coordinates of the FFD control points are modified, the parametric coordinates of the embedded surface points are used to recompute their physical coordinates for the deformed configuration. Thus, the position of the FFD control points become the design variables of the optimization problem.

One important feature of the FFD method is that it is agnostic to the source of the surface points. It also requires no information regarding connectivities among the surface points. Therefore, the same FFD can be used to consistently deform multiple representations of the same component, such as a meshes used for aerodynamic analysis and another for structural analysis [48].

This also facilitates the use of externally provided geometries [55]. For cases in which just a CFD mesh is available, the surface points can be directly embedded in the FFD. There is no need to reverse-engineer a CAD model that fits these mesh points.

Another advantage of the FFD approach for shape optimization is the availability of analytical derivatives. Most FFD applications use analytically differentiable mappings, such as trivariate B-splines or non uniform rational basis splines (NURBS), facilitating the propagation of derivatives across the geometry manipulation module.

Nevertheless, this method operates simultaneously on the entire geometry, thereby ignoring valuable information on component subdivision. For instance, in a wing-fuselage configuration, using FFD to translate the wing is impossible without changing the fuselage shape embedded in the same FFD block.

Even if the wing and body are embedded in separate FFD blocks, additional nontrivial steps are required to track changes in intersection curves. Since FFDs manipulate only surface points, they do not carry any information regarding the surface description itself, what is necessary for intersection computation.

In this thesis, this issue is circumvented by adding another module, called *pySurf*, into the optimization chain that uses triangulated surfaces to recompute intersections during the optimization cycle. The points of the triangulated surfaces are still manipulated by FFDs, but *pySurf* retains the connectivity information to recreate the surface description. This process is detailed in Chapter 2.

1.4 Aerodynamic shape optimization of junctions

Koc et al. [56] show one of the first attempts to optimize junctions of tridimensional aircraft geometries. They use an Euler-based CFD solver for unstructured meshes to optimize the wing-pylon junction of the DLR-F6 configuration flying at Mach 0.74. The optimization modified the pylon cross section and the lower surface of the wing to reduce the peak Mach number on the wing-pylon junction and reduce drag. They use the surface mesh deformation procedure developed by Kim and Nakahashi [57]. In this approach, they deform the pylon mesh and then recompute the wing-pylon intersection line using the surface cells of the CFD mesh. Once they have the new intersection curve, they use a mesh deformation algorithm based on spring analogy to update the surface meshes using the new intersection curve as a boundary condition.

The displaced surface mesh nodes are not necessary on the aircraft outer mold line (OML), since the displacements are based solely on Hooke's Law. Therefore, they employ a surface reconstruction algorithm in which they use the undeformed surface mesh to define a quadratic approximation of the original surface, and the deformed mesh nodes are then projected onto this quadratic approximation. A problem arises when the design variables expose new parts of the pylon, for instance, by moving it downwards with respect to the wing. In this situation, they linearly extrapolate the external edges of the pylon mesh to compute the intersection curve with the wing. The extrapolated edges tend to show oscillations among them, and an additional smoothing process is necessary before the intersection computation. Since neither the user nor the design variables have control over the extrapolated shape, this approach is not appropriate for cases with significant geometric variation.

Brezillon and Dwight [29] optimized the wing shape and the body shape of the DLR-F6 as two separate problems using unstructured meshes in a RANS solver. They use FFDs to displace control points of the CAD representation of the wing and the body. Then a CAD engine recomputes the wing-body intersection curve and a meshing tool regenerates the entire CFD mesh (including the volume nodes) based on the new surface configuration. This process is repeated for every design evaluation. Even though they use the adjoint method to compute sensitivities of the functions of interest with respect to the mesh nodes ($\partial f / \partial \mathbf{r}_{\text{vol}}$), they use finite-differences to compute sensitivities of the mesh nodes with respect to the FFD design variables ($\partial \mathbf{r}_{\text{vol}} / \partial \mathbf{X}$) to complete the chain rule. Consequently, for n design variables, they would have to regenerate n meshes for each sensitivity evaluation. Each mesh regeneration process takes on the order of minutes, what makes this method unfeasible for problems with hundreds of design variables.

Xu et al. [58] optimized the junction of the DLR-F6 configuration also using a RANS solver for unstructured meshes. At the beginning of the optimization, they compute the parametric coordinates of the surface nodes of the CFD mesh on the B-spline patches that compose the geometry. Later, they use the control points of the B-spline patch of the body as design variables to modify the surface shape. Once the B-spline patch is deformed, they recompute the parametric coordinates of the wing-body intersection nodes for the deformed shape. This variation in parametric coordinates is propagated to the surface nodes surrounding the intersection via an inverse distance weighting (IDW) method. Once they have the deformed position of the surface nodes, they use mesh deformation methods to displace the volume nodes, thus avoiding the volume mesh regeneration step employed by Brezillon and Dwight [29]. Nevertheless, derivatives across the intersection computation and mesh deformation modules are still computed by finite-differences. Although this method uses mesh deformation rather than mesh regeneration, its cost still scales with the number of design variables.

Other recent work has studied the feasibility of differentiating CAD software with AD. Mykhaskiv et al. [59] use the same IDW mesh deformation approach from Xu et al. [58] described in the previous paragraph, but they include OpenCascade, which is an open source CAD kernel, to handle the geometry manipulation and intersection computation. Then they use ADOL-C [60] to differentiate OpenCascade using operator overloading. This allows the inclusion of the CAD engine directly into the optimization framework. As an example, they optimize the wing-body junction of the CRM configuration using three control points as design variables and an Euler-based CFD solver.

The aerospace community is actively pursuing solutions to add more flexibility to the geometry and mesh manipulation tools used in ASO frameworks, with junction optimization being one of the most direct application of these techniques. This thesis contributes to this line of research by developing techniques to automate the handling of overset meshes, collar mesh generation, and triangulated surface intersections embedded in a fully differentiable ASO framework.

1.5 Thesis objectives

The main objective of this thesis is to show that component-based parametrization allied to overset meshes can extend the flexibility of ASO frameworks. The following intermediate objectives were outlined to achieve this goal:

1. Develop a collar mesh generation module that automatically detects intersection curves among primary components and use these curves as starting features for the

collar mesh generation. This module should be differentiable to be included in the reverse AD chain of the optimization framework.

2. Implement a component-based geometry manipulation module that allows independent parametrization of the aircraft component and embed it in an existing ASO framework.
3. Address the outcomes of allowing overset connectivity updates in aerodynamic shape optimization problems, since the independent manipulations of the components leads to changes in overlapped regions.
4. Demonstrate the capabilities of the framework for aerodynamic shape optimization of complex configurations in two test cases: the design of the wing-body junction of a conventional aircraft configuration and the optimization of a strut-braced wing (SBW) geometry.

1.6 Thesis outline

To achieve the objective stated above, the multidisciplinary design optimization of aircraft configurations with high fidelity (MACH) optimization framework [61] is expanded by the addition of the pySurf module, which is dedicated to component-based geometry and mesh manipulation.

The methodology to allow component-based shape optimization is discussed in Chapter 2, addressing objective 1. This includes the introduction of the pySurf module and its capabilities for intersection computation and collar mesh generation. The reverse AD is selectively applied to the subroutines of the pySurf module to obtain an efficient code, and these details are also discussed in this chapter.

The next step is the inclusion of the pySurf module into the MACH optimization framework to accomplish objective 2. Chapter 3 describes the framework modules and the interactions among them. This chapter also discusses how derivatives are computed within this framework using AD techniques and the hybrid adjoint method. An univariate optimization problem is defined to test the framework and also to address objective 3. This problem allows us to quantify the noise in the CFD results caused by changes in overset connectivities and it also shows the relation between mesh refinement and the noise magnitude. This problem is finally used to verify if the optimizer manages to improve the design despite the noise in the functions of interest.

Next, the optimization framework capabilities is accessed in two ASO problems involving junction design. Chapter 4 brings an application of the proposed methodology in the design of the wing-body junction of a conventional aircraft configuration flying at transonic conditions, while Chapter 5 shows the ASO of a SBW configuration using the optimization framework. The effectiveness of the methodology in removing shock and flow separation in junction regions is demonstrated, achieving objective 4 of this thesis. Design insights obtained by comparing the baseline and optimized designs are also discussed.

Finally, the main results and contributions of this thesis are summarized in Chapter 6. This chapter also suggests possible future work regarding the improvement of the proposed methodology and also its application in challenging optimization problems that will benefit from the new geometry manipulation module.

CHAPTER 2

Component-based parametrization

As mentioned in the introduction, our aim is to fully exploit the overset mesh capabilities by extending the component-based approach used in overset CFD solvers to the geometry manipulation module within the optimization framework. Consequently, we also need to automatically track changes in junctions among these primary components during optimization to properly modify the collar meshes.

Automatically producing a collar mesh between two components is a multifaceted challenge: We must find the intersection curve, march a surface mesh on the surfaces of both components, and project the mesh nodes back onto the surfaces during the marching process. Furthermore, we need the partial derivatives of the mesh nodes with respect to the design variables to enable gradient-based aerodynamic shape optimization with the adjoint method. For this purpose, the entire process of finding the intersection curve and producing a collar mesh must be differentiated.

We identified and implemented the necessary operations to automatically generate collar meshes for each design evaluation, as explained in Sec. 2.1. Then we use reverse mode AD to efficiently compute the derivatives of the generated mesh points with respect to the design variables. The derivative computation procedure is detailed in Sec. 2.4.

2.1 Collar mesh generation overview

To address the difficulty of automatically creating collar meshes, we developed a new geometry module called *pySurf* that performs the following sequence of operations:

1. For each primary component, receive triangulated surfaces that represent the full vehicle geometry. *pySurf* may either read these surfaces from a file at the beginning of the optimization or, at optimization runtime, receive updated triangulated surfaces provided by a separate geometry-manipulation module, such as FFDs.

2. Compute the intersections between the triangulated surfaces of the primary components.
3. Automatically generate surface collar meshes for the intersections. These meshes should be structured to be compatible with the CFD solver used in this work.

These operations are represented in steps a–c of Fig. 2.1. A separate module called *pyHyp* generates the volume mesh (step d in Fig. 2.1) and is introduced in Sec. 3.3. The resolution requirements for the triangulated surface are discussed in Chapter 3 and Sec. 4.3.

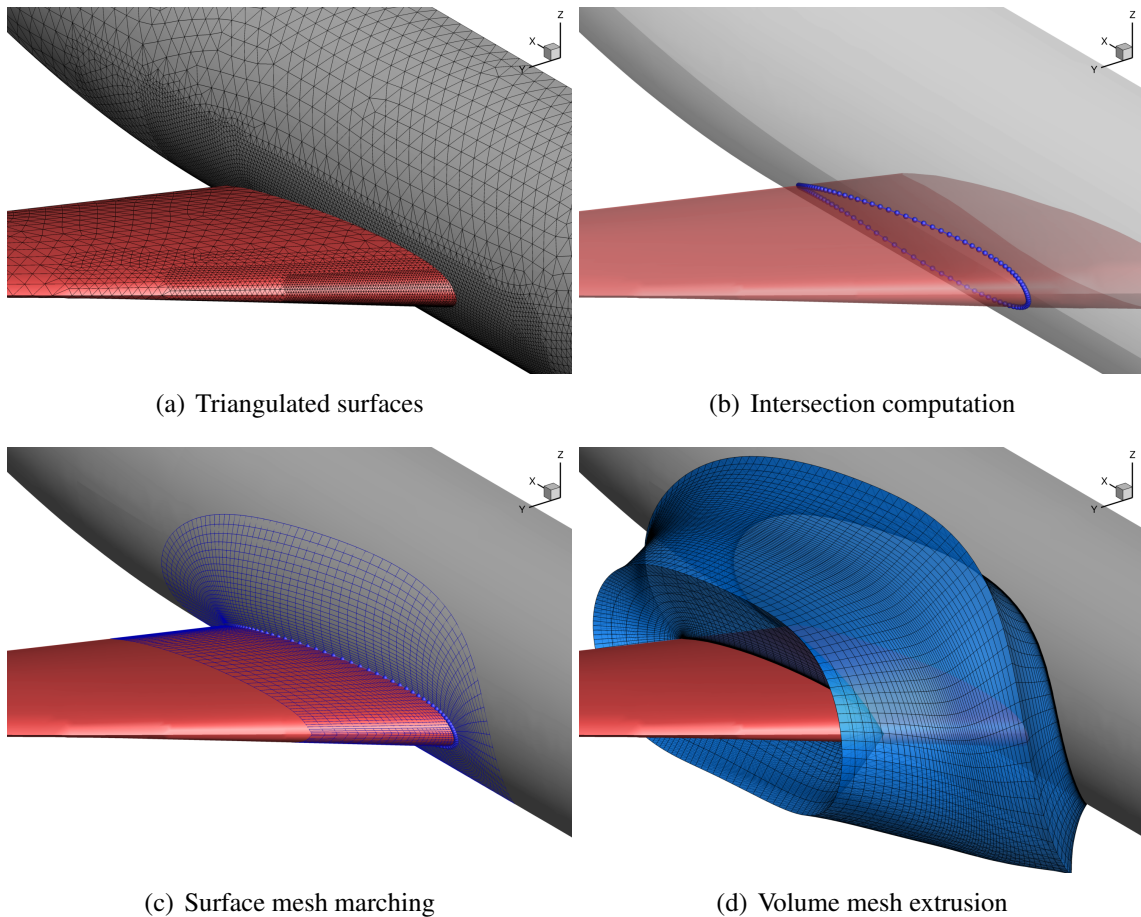


Figure 2.1: Steps for generating collar mesh for a wing-fuselage junction. *pySurf* is responsible for steps a–c. The mesh extrusion module used in step d (*pyHyp*) is covered in Sec. 3.3.

2.2 Intersection computation

Once *pySurf* receives the triangulated surfaces, it has to compute intersections among them. The triangulated surface of a primary component may have tens of thousand elements.

Applying triangle-triangle intersection algorithms directly to all possible triangle pairs is costly. Therefore, the intersection computation in pySurf is performed in three steps to avoid unnecessary triangle-triangle intersection verification. The first two steps filter which triangles are likely to intersect by using methods based on Cartesian bounding boxes and digital tree searches, and we only use the triangle-triangle intersection algorithm in the third step.

Here we describe the steps to compute intersections between two components (for instance, component A and component B), since this process is repeated for every pair of primary components. First, we compute Cartesian bounding boxes for the two primary components, then we determine the intersection between these bounding boxes. Next we flag the triangles from both components that belong to the bounding box intersection region. This step is relatively quick since we just need to compare maximum and minimum values of nodal coordinates against the Cartesian box bounds to flag the triangles.

In the second step, we take the flagged triangles of the component that has fewer flagged triangles (for example, component A) and build an alternating digital tree (ADT) [62] to minimize the tree complexity. This tree structure groups discrete elements based on their spatial location so we can efficiently find which pairs of elements are likely to intersect. We then take the bounding boxes of every flagged triangle of component B and perform ADT searches to find which flagged triangles from component A are close to it. At the end of this step, every flagged triangle of component B has a corresponding list of triangles from component A for the intersection search.

The third step consists of using fast pairwise triangle-triangle intersection algorithms [63] on the candidate elements given by the ADT searches to identify the two points that determine intersection line between pair of triangles. We then concatenate the lines given by the pairwise triangle-triangle intersections to determine the entire intersection curve.

An arbitrary number of primary components can be intersected using this method. Each pairwise component intersection may have multiple intersection curves that could be selected as possible starting curves for the collar mesh generation detailed in the next section.

We also added features dedicated to manipulating these intersection curves, such as merging, splitting, and remeshing, so that the user can accurately control the topology and the number of nodes describing the intersection, since this curve is the starting point for the hyperbolic surface marching. The number of nodes distributed along the intersection curve remains the same throughout the optimization to keep the same number of nodes in the CFD mesh.

2.3 Hyperbolic surface mesh generation

Having identified the intersection curves, we can now use hyperbolic mesh-marching algorithms [37] to produce the surface collar meshes. This type of mesh generation algorithm starts from a baseline curve and then uses a marching scheme to generate the next layer of the surface mesh. This process is repeated until the desired number of layers and mesh extension is reached. One advantage of hyperbolic marching schemes over other mesh generation methods (such as transfinite interpolation [64]) is that they do not require the outer boundary of the mesh domain to be defined, making the process easier to automate, especially for collar meshes.

We project every new layer of nodes onto the triangulated surfaces before the generation of the next layer to ensure consistency with the underlying geometry. For the projection step, finding the nearest surface triangle by brute force would not be tractable, especially when we must regenerate the mesh for each optimization iteration. Thus, we use the ADT algorithm once again to accelerate the search process. The ADT searches provide the closest surface elements to a given point, then we only need to compare projections on those filtered elements to find the nearest projection of the node onto the triangulated surface.

We added features to the standard hyperbolic marching scheme to improve control over the mesh generation, as discussed in Appendix A. For instance, the user can choose to preserve special surface features, such as trailing edge corners, as the surface mesh is marched. First we create a discrete representation of these curves using line segments. Then, we identify which node from the baseline curve is closest to the guide curve segments and then we project this node onto the guide curve. In addition, we locally modify the marching equations for this node so that it marches in a direction tangent to the guide curve. This process is repeated for every new layer of the marched mesh.

We also added a feature to preserve the relative node spacings from the baseline curve throughout the entire mesh. This feature works as follows: After every new layer is generated, we redistribute the nodes of this new layer using the same relative arc-lengths of the nodes in the baseline curve. This redistribution is important when generating meshes near trailing edges, since the artificial dissipation associated with the marching scheme smooths the node intervals near the trailing edge despite the local high refinement of the baseline curve. Once the redistributed nodes are computed, we project them to the reference surface to ensure the consistency between the mesh and the surface representation.

A comparison of meshes generated with and without these geometry-preserving options is shown in Fig 2.2. In Fig 2.2a, we simply extrude the mesh without considering any geometric features. In Fig 2.2b, we set the upper and lower edges of the blunt trailing

edge as guide curves. This forces the nearest point of the extruded mesh layer to coincide with the guide curve, which preserves the edge geometry. However, the bunched spacing at the leading and trailing edges dissipates as we march away from the intersection curve. Fig 2.2c shows the effect of preserving the relative node spacing, what produces a surface mesh better suitable for CFD.

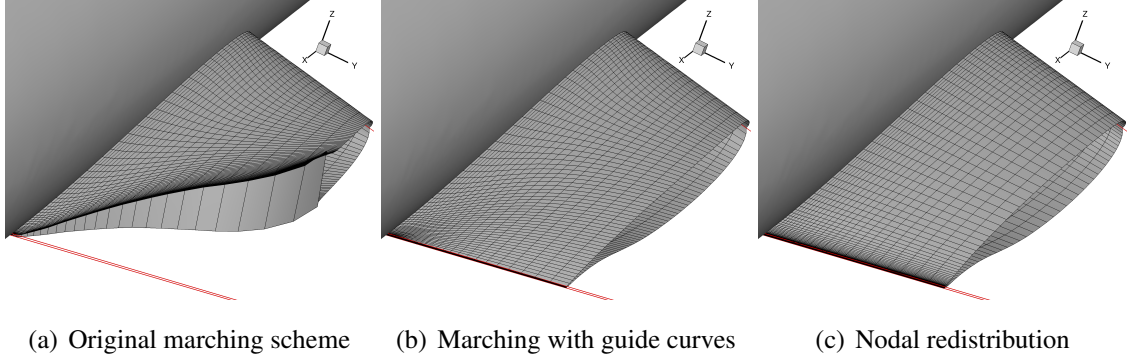


Figure 2.2: Additional features implemented in the collar marching scheme to improve mesh quality for CFD analyses.

2.4 Automatic differentiation

We need to efficiently compute derivatives of the objective and constraints to enable effective gradient-based aerodynamic shape optimization. We apply the reverse mode AD throughout the entire analysis chain for this purpose, what we discuss in detail in Sec. 3.7. Therefore, pySurf also uses the same AD method to internally backpropagate partial derivatives.

Figure 2.3 outlines the notation used in this work to represent AD versions of a generic subroutine F that receives a vector of inputs $\mathbf{X} = [x_1 \ x_2 \ \dots \ x_n]^T$ and outputs the vector $\mathbf{Y} = [y_1 \ y_2 \ \dots \ y_m]^T$, without loss of generality, since we always can concatenate multiple inputs and outputs in a single vector. In this figure, \dot{F} represents the forward AD version of the subroutine F , while \bar{F} represents the reverse AD version.

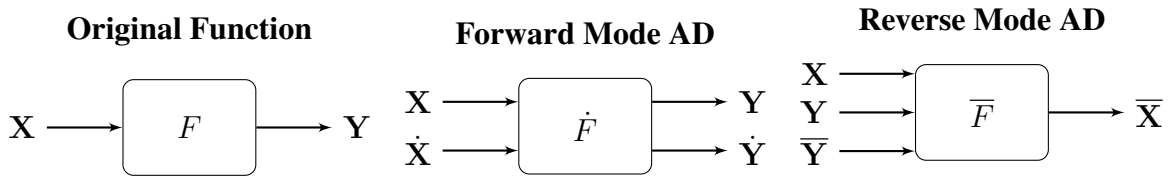


Figure 2.3: Representation of a generic subroutine and its corresponding AD versions.

The forward AD code propagates derivative seeds from inputs ($\dot{\mathbf{X}}$) to outputs ($\dot{\mathbf{Y}}$), while the reverse AD code propagates derivatives from outputs ($\bar{\mathbf{Y}}$) to inputs ($\bar{\mathbf{X}}$). The derivative seeds can be correlated with the following expressions [65]:

$$\begin{array}{lll}
 \text{Original Function} & \text{Forward Mode AD} & \text{Reverse Mode AD} \\
 \mathbf{Y} = F(\mathbf{X}) & \dot{\mathbf{Y}} = J \cdot \dot{\mathbf{X}} & \bar{\mathbf{X}} = J^T \cdot \bar{\mathbf{Y}}
 \end{array} \quad (2.1)$$

where J is the Jacobian matrix correlating inputs and outputs of the subroutine F :

$$J = \frac{\partial \mathbf{Y}}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \dots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}. \quad (2.2)$$

In Sec. 2.2 we mentioned that pySurf has several tools to control the intersection curves, such as merging multiple curves to create a new one, splitting curves based on sharp corners or intersections, and redistributing nodes along the curve to refine more important regions. During forward execution, pySurf stores all the intermediary steps required by the user to generate the appropriate curve for hyperbolic marching, then it uses the same steps, but in reverse order, to perform the reverse propagation of derivatives.

The automatically differentiated code for pySurf is generated using the Tapenade AD tool [65], but we have to selectively alter the differentiation process in each pySurf module to get an efficient final code. The next sections provide detailed explanation regarding these exceptions.

2.4.1 Projection subroutine

One of the steps of hyperbolic surface marching routine is the projection of the newly-generated points back to the reference triangulated surface. The initial step of the projection algorithm consists of an ADT search to find the triangulated surface elements that are closest to the point to be projected. Then, we run the point-triangle projection algorithm on those candidate triangles and take the closest projected point. This point-to-triangle projection subroutine can be algorithmically described as:

$$\mathbf{X}_{\text{proj}}, \mathbf{n}_{\text{proj}} = F_{\text{proj}}(\mathbf{X}_0, \mathbf{X}_A, \mathbf{X}_B, \mathbf{X}_C), \quad (2.3)$$

where F_{proj} represents the projection routine, \mathbf{X}_0 are the Cartesian coordinates of the point to be projected, \mathbf{X}_A , \mathbf{X}_B , and \mathbf{X}_C are coordinates of the three nodes from the triangle that will receive the projection, and \mathbf{X}_{proj} is the projected point. We use the same projection subroutine to also compute the surface normal vector \mathbf{n}_{proj} of the triangle, since this vector is used by the hyperbolic mesh marching subroutine to determine the marching direction.

During the forward execution of the projection code, we store the triangles that receive the projections, so that we do not have to repeat the ADT searches during the reverse propagation of derivatives. Therefore, we only need to differentiate the point-to-triangle projection routine. The reverse-differentiated version of Eq. (2.3) is:

$$\delta\bar{\mathbf{X}}_0, \delta\bar{\mathbf{X}}_A, \delta\bar{\mathbf{X}}_B, \delta\bar{\mathbf{X}}_C = \bar{F}_{\text{proj}}(\mathbf{X}_0, \mathbf{X}_A, \mathbf{X}_B, \mathbf{X}_C, \mathbf{X}_{\text{proj}}, \mathbf{n}_{\text{proj}}, \bar{\mathbf{X}}_{\text{proj}}, \bar{\mathbf{n}}_{\text{proj}}), \quad (2.4)$$

where the bars represents reverse derivative seeds, and the δ represents variations of the reverse derivative seeds that should be accumulated, since each triangle node may get contributions from multiple projections. That is:

$$\begin{aligned} \bar{\mathbf{X}}_0 &\leftarrow \bar{\mathbf{X}}_0 + \delta\bar{\mathbf{X}}_0 & \bar{\mathbf{X}}_A &\leftarrow \bar{\mathbf{X}}_A + \delta\bar{\mathbf{X}}_A \\ \bar{\mathbf{X}}_B &\leftarrow \bar{\mathbf{X}}_B + \delta\bar{\mathbf{X}}_B & \bar{\mathbf{X}}_C &\leftarrow \bar{\mathbf{X}}_C + \delta\bar{\mathbf{X}}_C \end{aligned} \quad (2.5)$$

2.4.2 Hyperbolic surface marching

The hyperbolic surface mesh marching generates the surface mesh layer-by-layer. Let \mathbf{R}_{j-1} be a vector with $3n_i$ elements representing the Cartesian coordinates of the n_i nodes of the layer $j - 1$ of the surface mesh. The nodal coordinates of the next layer are given by:

$$\mathbf{R}_j = \mathbf{R}_{j-1} + \Delta\mathbf{R}_j, \quad (2.6)$$

where the displacement $\Delta\mathbf{R}_j$ is obtained via the solution of a linear system defined by the discretized hyperbolic equations, as introduced in Appendix A:

$$\mathbf{K}_j \cdot \Delta\mathbf{R}_j = \mathbf{F}_j. \quad (2.7)$$

A dedicated routine computes the matrix \mathbf{K}_j and the right-hand side vector \mathbf{F}_j based on the nodal coordinates of the $(j - 1)$ -th layer (\mathbf{R}_{j-1}) and on the normals of the underlying triangulated surface at the projections of each node (\mathbf{n}_{j-1}):

$$\mathbf{K}_j, \mathbf{F}_j = F_{\text{hyp}}(\mathbf{R}_{j-1}, \mathbf{n}_{j-1}) \quad (2.8)$$

During the reverse execution of the code, we initially have the derivative seeds of the

j -th layer, and we should propagate these seeds to the previous layer and also to the underlying triangulated surface.

The linear system of Eq. (2.7) is solved with a LU-factorization algorithm from the linear algebra package (LAPACK) [66]. In a naive AD approach, we would have to build a differentiated version of the linear system solver to propagate the derivative seeds from the solution of the linear system ($\Delta\bar{\mathbf{R}}_j$) back to the matrix \mathbf{K}_j and the vector \mathbf{F}_j . However, we can use the fact that we know the solution of the linear system from the forward pass to avoid this. The corresponding reverse algorithm for Eq. (2.7) is [67]:

$$\mathbf{K}_j^T \cdot \bar{\mathbf{F}}_j = \Delta\bar{\mathbf{R}}_j \quad (2.9a)$$

$$\bar{\mathbf{K}}_j = -\bar{\mathbf{F}}_j \cdot \Delta\mathbf{R}_j^T. \quad (2.9b)$$

First we solve the linear system in Eq. (2.9a) to compute $\bar{\mathbf{F}}_j$ with the same solver used for Eq. (2.7), then we find $\bar{\mathbf{K}}_j$ with Eq. (2.9b). The advantage of this approach is that we only use the native routines from LAPACK, without any modification. Once we have the derivative seeds of the components of the linear system, we can use the reverse version of Eq. (2.8), which is given by Tapenade:

$$\bar{\mathbf{R}}_{j-1}, \bar{\mathbf{n}}_{j-1} = \bar{\mathbf{F}}_{\text{hyp}}(\mathbf{R}_{j-1}, \mathbf{n}_{j-1}, \mathbf{K}_j, \mathbf{F}_j, \bar{\mathbf{K}}_j, \bar{\mathbf{F}}_j) \quad (2.10)$$

This concludes the backpropagation of derivatives from the j -th layer to the $(j - 1)$ -th layer. We repeat this process until we accumulate derivatives seeds into the intersection curve, which is the source of the hyperbolic marching.

2.4.3 Intersection computation

During the forward execution of the intersection computation code, we execute all three steps described in Sec. 2.2. The first two steps use bounding boxes and ADT searches to identify triangles that likely intersect, but they do not compute the intersection curve per se. Therefore, these steps are kept outside of the differentiated routines, similar to what was discussed in Sec. 2.4.1, since we can store the intersecting triangles during the forward (original) execution of the code.

The execution of the third step of the intersection code gives a line (defined by two points) for every pair of intersecting triangles. The inputs and outputs of this intersection routine can be algorithmically described as follows:

$$\mathbf{X}_{i1}, \mathbf{X}_{i2} = \mathbf{F}_{\text{int}}(\mathbf{X}_{a1}, \mathbf{X}_{a2}, \mathbf{X}_{a3}, \mathbf{X}_{b1}, \mathbf{X}_{b2}, \mathbf{X}_{b3}), \quad (2.11)$$

where \mathbf{X}_{i1} and \mathbf{X}_{i2} represent the Cartesian coordinates of the two points that define the intersection line, F_{int} is the intersection routine, \mathbf{X}_{a1} , \mathbf{X}_{a2} , and \mathbf{X}_{a3} are coordinates of the three nodes of a triangle from one primary component, and \mathbf{X}_{b1} , \mathbf{X}_{b2} , and \mathbf{X}_{b3} are nodal coordinates of a triangle from another primary component. This is the only subroutine of the pySurf intersection module that is automatically differentiated with Tapenade. The reverse-differentiated version of Eq. (2.11) can be written as:

$$\begin{aligned} \delta\bar{\mathbf{X}}_{a1}, \delta\bar{\mathbf{X}}_{a2}, \delta\bar{\mathbf{X}}_{a3}, \delta\bar{\mathbf{X}}_{b1}, \delta\bar{\mathbf{X}}_{b2}, \delta\bar{\mathbf{X}}_{b3} = \\ = \bar{F}_{\text{int}}(\mathbf{X}_{a1}, \mathbf{X}_{a2}, \mathbf{X}_{a3}, \mathbf{X}_{b1}, \mathbf{X}_{b2}, \mathbf{X}_{b3}, \mathbf{X}_{i1}, \mathbf{X}_{i2}, \bar{\mathbf{X}}_{i1}, \bar{\mathbf{X}}_{i2}) \end{aligned} \quad (2.12)$$

Once again the δ indicates increments to the derivative seeds of each triangulated surface node. The derivative seeds of the triangulated surface nodes might get contributions from both Eq. (2.4) and Eq. (2.12).

The reverse derivative seeds of the intersection points should be erased once they are used, otherwise they might seed the same triangle nodes twice. Figure 2.4 brings an example to illustrate this point. Let triangles $\triangle ABC$ and $\triangle CBD$ belong to one component, and triangle $\triangle EFG$ belong to another component. The triangles $\triangle ABC$ and $\triangle EFG$ define the intersection line \overline{IJ} , while triangles $\triangle CBD$ and $\triangle EFG$ define the intersection line \overline{JK} . We store these relationships between intersection lines and triangles during the forward pass, so that we do not need to repeat the intersection search.

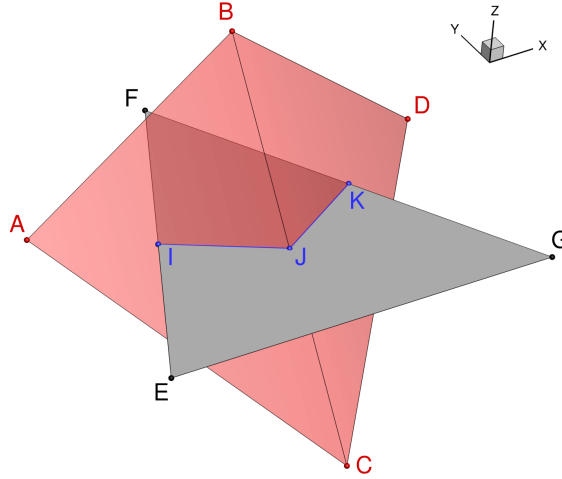


Figure 2.4: Intersection between triangles of two components.

The reverse hyperbolic surface mesh marching routine provides derivative seeds for the intersection points, namely $\bar{\mathbf{X}}_I$, $\bar{\mathbf{X}}_J$, and $\bar{\mathbf{X}}_K$. When we execute the reverse intersection routine (\bar{F}_{int} from Eq. (2.12)) for the intersection line \overline{IJ} , we propagate derivative seeds

from nodes I and J to the nodes of the parent triangles that define this line (in this case nodes A, B, C, E, F , and G). Then, we need to set $\bar{\mathbf{X}}_I = 0$ and $\bar{\mathbf{X}}_J = 0$ before the reverse intersection computation for line \overline{JK} , otherwise node J would seed derivatives of nodes B, C, E, F , and G once again, what is incorrect.

2.4.4 Tests for derivative validation

We perform the validation of the differentiated code in two steps: comparison with finite difference results, and the dot product test [68]. Even though we only need the reverse AD version of the code for optimization, we also generate forward AD version for these tests.

The finite-difference test checks consistency between the original subroutine and its forward AD version. We can locally approximate the generic subroutine introduced of Fig. 2.3 by:

$$F(\mathbf{X} + h\mathbf{g}) \approx F(\mathbf{X}) + J \cdot h\mathbf{g}, \quad (2.13)$$

where \mathbf{g} is a generic vector and h is a small step size, according to the design space scale. If we use the forward AD seeds in Eq. (2.13) ($\mathbf{g} = \dot{\mathbf{X}}$), we get:

$$\frac{F(\mathbf{X} + h\dot{\mathbf{X}}) - F(\mathbf{X})}{h} \approx J \cdot \dot{\mathbf{X}} = \dot{\mathbf{Y}}. \quad (2.14)$$

This equation allows a comparison between finite difference results (on the left side) and forward AD results (on the right side). This usually allows us to verify derivatives up to 5 digits, due to subtractive cancellation errors.

Once we use the finite-difference test to compare the forward AD results against the original subroutine trends, we can check the consistency between the forward AD code and the reverse AD code with the dot product test. For the generic subroutine introduced in Fig. 2.3, the dot product test is:

$$\dot{\mathbf{X}}^T \cdot \bar{\mathbf{X}} = \dot{\mathbf{Y}}^T \cdot \bar{\mathbf{Y}}, \quad (2.15)$$

which can be derived from Eq. (2.1). Now we first generate a random vector $\dot{\mathbf{X}}$ and use the forward AD code to obtain $\dot{\mathbf{Y}}$. Next, we generate a random vector $\bar{\mathbf{Y}}$ and then compute the corresponding $\bar{\mathbf{X}}$ with the reverse AD code. Then, we verify if Eq. (2.15) is met to machine precision to ensure consistency between both AD versions, thus passing the dot product test.

These two tests ensure that the reverse AD code is consistent with the original subroutine, and we perform this verification for every differentiated module in pySurf.

2.4.5 Gradient verification: CRM case

We use pySurf to examine the wing-body intersection curve of the Common Research Model (CRM) [69] as we move the wing from a low-wing to a high-wing position, as shown in Fig. 2.5. The wing and fuselage are both triangulated surfaces and we recompute the intersection curve and collar mesh for each configuration.

Because the process is fully differentiated, we can calculate the gradients of the coordinates of the collar mesh nodes with respect to the design variables. For the CRM example, we computed the derivative of the trailing edge point Y -coordinate with respect to the vertical position of the wing (Z). We also computed derivatives of the spanwise location of some collar mesh nodes for this same example. The derivatives obtained for both cases are plotted in Fig. 2.6 for the CRM wing-fuselage model. Even though the wing and fuselage surfaces are inherently discretized, we found that the computed gradients are indeed suitable for gradient-based optimization since they locally point in the correct direction.

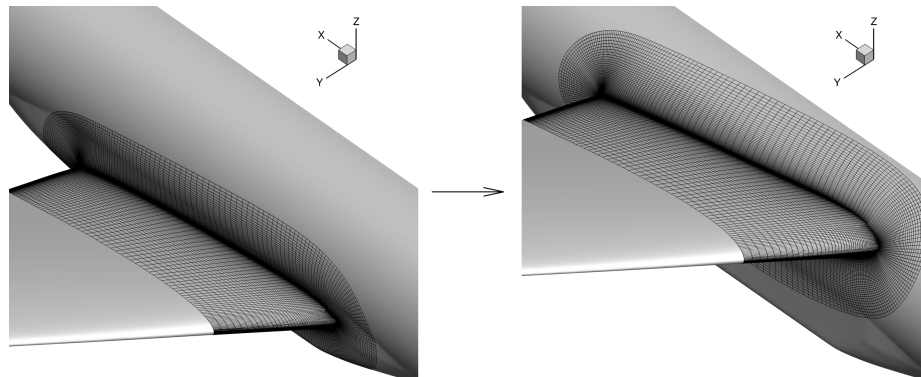


Figure 2.5: Low-wing configuration shifted to high-wing configuration with automatically generated collar meshes for the CRM geometry.

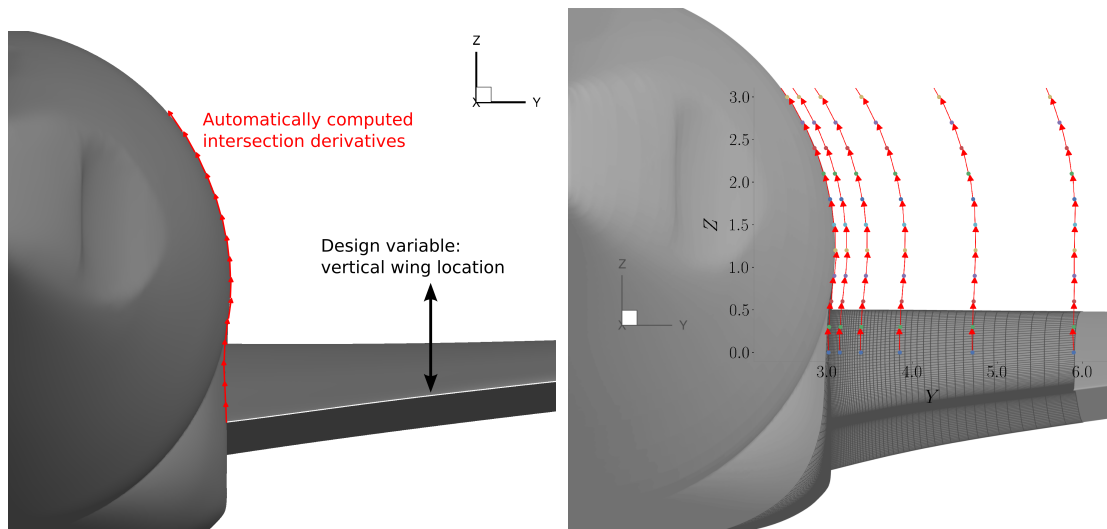


Figure 2.6: Aft view of the CRM wing-body junction. The left figure shows the design variable definition and the tracking of the trailing edge intersection point. The figure on the right shows how surface mesh points move as the wing is translated. The red arrows indicate the AD-computed derivatives for the trailing edge intersection position (left) and surface mesh points (right) as we vertically translate the wing. The computed gradients for triangulated surface intersections are smooth and accurate enough for gradient-based optimization.

CHAPTER 3

Optimization Framework

pySurf on its own is just a geometry module, so it has to be coupled with other analysis tools and an optimizer to create a fully capable framework for aircraft design optimization. Because of the high computational cost of each analysis, high-fidelity aircraft design optimization requires a computationally efficient and massively parallelized multidisciplinary framework. Furthermore, gradient-based optimization is necessary due to the large number of design variables [17, 21, 23]. Thus, this framework must also be able to compute first-order derivatives of both objective and constraint functions with respect to each design variable.

The MACH framework [61] was developed with these needs in mind. This framework uses Fortran and C++ routines wrapped in a Python interface to perform efficient high-fidelity aerostructural optimization. Specifically for this work, we only use modules relevant to ASO, which include a geometry modeler (*pyGeo*), volume mesh generator (*pyHyp*), volume mesh deformation module (*pyWarp*), aerodynamic solver (*ADflow*), optimizer (*SNOPT*), and the newly added *pySurf* module as the collar mesh generator. These modules are described in detail in the following sections.

To use this framework, the user should provide the following for each primary component:

Structured surface mesh with the desired resolution for CFD analysis. This surface mesh is used to generate the volume mesh by the hyperbolic extrusion module (*pyHyp*). The mesh should be refined near intersections to facilitate the overset hole cutting process.

Triangulated (unstructured) surface mesh, which is used as the reference for intersection computation and collar mesh generation by *pySurf*. This triangulated mesh should be finer than the structured surface mesh in order to appropriately represent high-curvature regions (such as leading edges) for the collar mesh generation process.

Free-form deformation (FFD) box, which envelops the surfaces described above and whose control points give the necessary shape control resolution for the design optimization.

Here we should clarify why we need two descriptions—one structured and the other unstructured—for each primary component. The structured meshes of the primary components have the surface resolution required by the CFD solver, which is already an approximate description of the underlying continuous surface representation. pySurf cannot use the same resolution to march new hyperbolic surface meshes for the collars because this would cause an additional loss of information compared to the original continuous representation. Therefore, the user also needs to provide another surface representation, finer than the surface meshes used by the CFD solver, so that the hyperbolic marching algorithm has access to a relatively more accurate surface description compared to the structured meshes.

Because this finer triangulated surface mesh is not actually used by the CFD solver, it can be unstructured, making it is easier to generate and refine in critical regions, such as high-curvature areas. The use of a heavily refined triangulated surface mainly impacts the setup time of the optimization due to the large number of Newton searches required to embed all surface points into the FFDs. The increase in computational time of the subsequent optimization steps is much less noticeable because of the linear nature of the FFD method and its derivatives.

The use of triangulated surfaces for automatic mesh generation is recognized as a possible practice for automatic mesh generation [40, 70] because most geometry manipulation packages can output files with this representation. Furthermore, discrete representations are less susceptible to geometry interpretation problems, such as discrepancies in the representation of intersection curves caused by the use of different geometry kernels [50].

The overall computational framework arrangement is shown in the extended design structure matrix (XDSM) [71] diagram of Fig. 3.1. Elements with the (0) superscript refer to the baseline configuration, while the * superscript denotes optimized values. Stacked elements indicate that the given data or module should be defined for each aircraft component and collar mesh. This figure also highlights how the volume mesh generation process is only used at the initialization step, and then replaced by the mesh deformation module during the optimization.

Further details regarding the framework are also given by Kenway et al. [61], and this framework has been used in several aerodynamic [7, 9, 72, 73] and aerostructural [31, 74, 75] design optimization studies. The framework modules are described in the next sections, including the details of how they use the three sets of inputs.

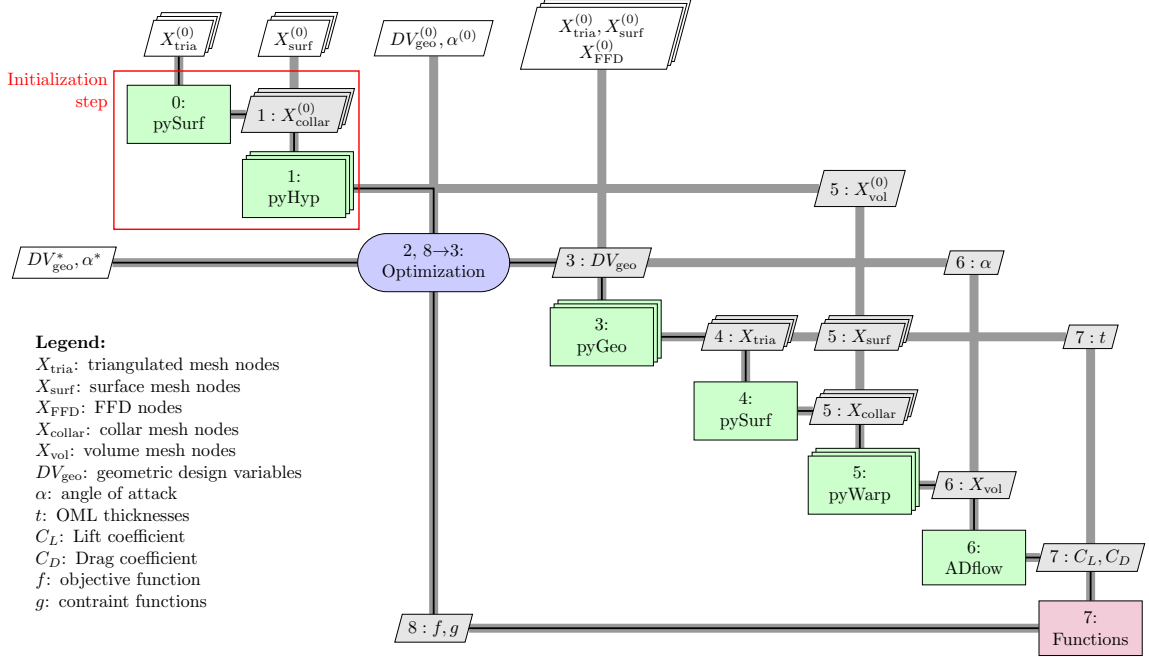


Figure 3.1: XDSM of the overall optimization framework. The initialization step generates the reference volume nodes for the mesh deformation.

3.1 Geometry modeler—pyGeo

pyGeo is a Python module capable of manipulating geometries with the FFD approach [55]. This module complements pySurf by handling the primary component geometries, whereas pySurf focuses on intersections and collar meshes. pyGeo uses structured hexahedral boxes of control points (which we call FFD boxes) surrounding the geometry we want to manipulate to define a tri-variate B-spline mapping in the space. We then use a Newton search to determine the parametric coordinates of the structured surface mesh nodes within their corresponding B-spline boxes. We repeat the same process for the triangulated surface nodes so that both surface representations are parametrized in the same B-spline mapping and thus get consistent displacements.

Once we modify the position of the FFD control points, we use the parametric representation to recover the physical coordinates of the embedded surface nodes for the deformed configuration. Therefore, the positions of the FFD control points are the primary aerodynamic shape variables of the problem: moving a node from the FFD box deforms the embedded surface. The updated triangulated surfaces are used by the pySurf module to regenerate surface collar meshes, while the updated structured surface meshes are used by the mesh deformation module (pyWarp) to update the volume nodes.

The FFD deformation process of each component is independent because we use sepa-

rate overset meshes, triangulated surfaces, and FFD boxes. This gives the designer and the optimizer the freedom to control components' variations individually.

The FFD method does not generate a new geometry by itself. It parametrizes *changes* to a baseline geometry instead, and this allows the optimization framework to be agnostic with respect to geometry generation tools. Furthermore, the B-spline mapping is easily differentiable [48], which allows for efficient computation of the derivatives of the surface node positions with respect to control point positions.

pyGeo can also measure the thicknesses of the embedded components by embedding pairs of points and tracking their distances during the optimization. This allows us to define thickness constraints to prevent unfeasible designs from the structural point of view, as we discuss in Sec. 5.1.

3.2 Collar mesh generator—pySurf

pySurf, which is the module developed in the present work, produces collar meshes between intersected geometry components as described in Chapter 2. pySurf uses the unstructured surface meshes that are updated by pyGeo's FFDs throughout the optimization to compute intersections among primary components.

Prior to the optimization, the automatically generated surface collar mesh is used by the volume mesh generation module (pyHyp) to generate the baseline collar volume mesh, as shown in Fig. 3.1. During the optimization, on the other hand, the mesh deformation module (pyWarp) uses the updated position of the collar mesh surface nodes to deform the cells of the volume mesh.

3.3 Volume mesh generator—pyHyp

The previous modules operate on surface meshes; however, we need volume meshes for CFD analysis. We use a hyperbolic mesh marching scheme [38] to extrude structured surface meshes into volume meshes.

This mesh generation method is applied to the structured surface meshes of each primary component and collars (Figure 3.2). This step is done once at the beginning of the optimization to get the baseline volume meshes, as indicated by the initialization step of Fig. 3.1. These meshes are subsequently deformed by the mesh deformation module (pyWarp) throughout the optimization, since this is computationally less expensive than regenerating new volume meshes at each design point. Therefore, there is no need to differentiate the pyHyp module, as discussed in Sec. 3.7.

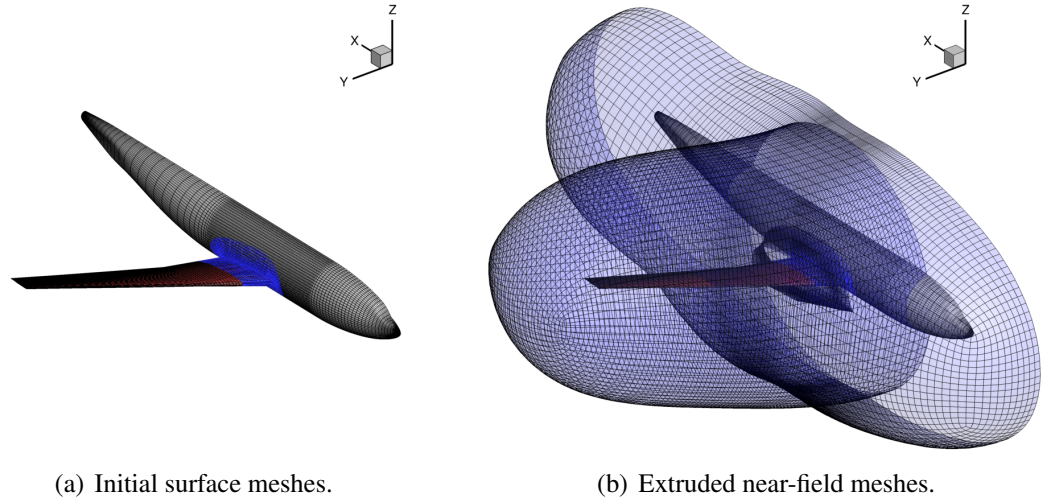


Figure 3.2: Hyperbolic extrusion of the surface meshes into volume meshes for a wing-body configuration.

The same advantages of hyperbolic mesh generation over other methods previously discussed for surface meshes in Sec. 2.3 apply here, especially those regarding the degree of automation and robustness. The main user-defined parameters are the height of the initial layer of cells and the extrusion distance. The initial marching distance is chosen to yield a y^+ value of 1.0 for a friction velocity estimated using turbulent flat plate regressions. The selected marching distance should allow a reasonable amount of overlap between the collar mesh and the primary component meshes, and it is usually from 1 to 3 mean aerodynamic chords. Additionally, cells within the collar mesh should be smaller than the cells in the primary component meshes so the collar cells are preserved during the overset hole cutting process. We usually achieve this by using an initial cell height for the collar meshes equal to 95% of the cell height of the primary components. We apply the overset interpolation boundary condition at the external surface of these meshes.

After the extrusion of all surface meshes, pyHyp uses the bounding box of these near-field volume meshes to create a background mesh that reaches an appropriate distance for the far-field boundary conditions. The background mesh is generated in two steps. First, we compute the bounding box defined by the near-field meshes and fill this box with Cartesian cells based on an user-provided cell size. Second, we use hyperbolic extrusion once again to march the exposed surface of this Cartesian block to the far-field distance. This distance is usually around 100 mean aerodynamic chords, as recommended by the Third Drag Prediction Workshop (DPW3) [76]. Figure 3.3 shows the symmetry plane of an overset mesh of a wing-body configuration in which we can see the O-grid topology of the background mesh.

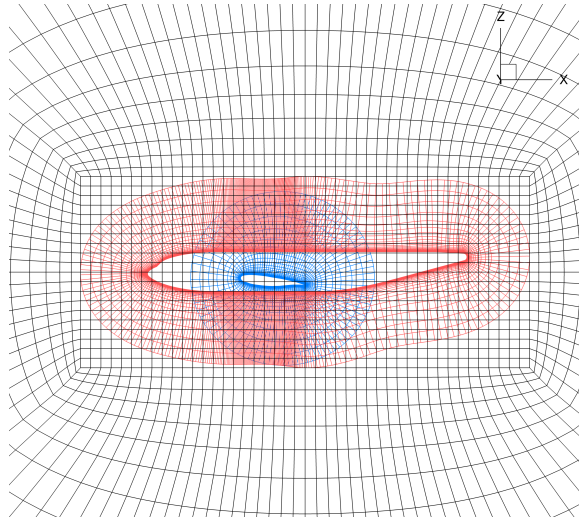


Figure 3.3: Symmetry plane showing the fuselage near field mesh (red) the wing near field mesh (blue) and the background mesh (black) made by a Cartesian block and an O-mesh.

3.4 Volume mesh deformation—pyWarp

Within the optimization loop, we use pyWarp to deform the initial volume meshes based on the updated component surfaces. pyGeo provides surface node updates for the primary components based on the FFD deformations, whereas pySurf provides updated surface nodes for the collar meshes based on the recomputed intersections, as indicated in Fig. 3.1. Then, pyWarp propagates the deformations of the surface nodes to the volume nodes using an explicit interpolation algorithm [77] (as shown in Fig. 3.4).

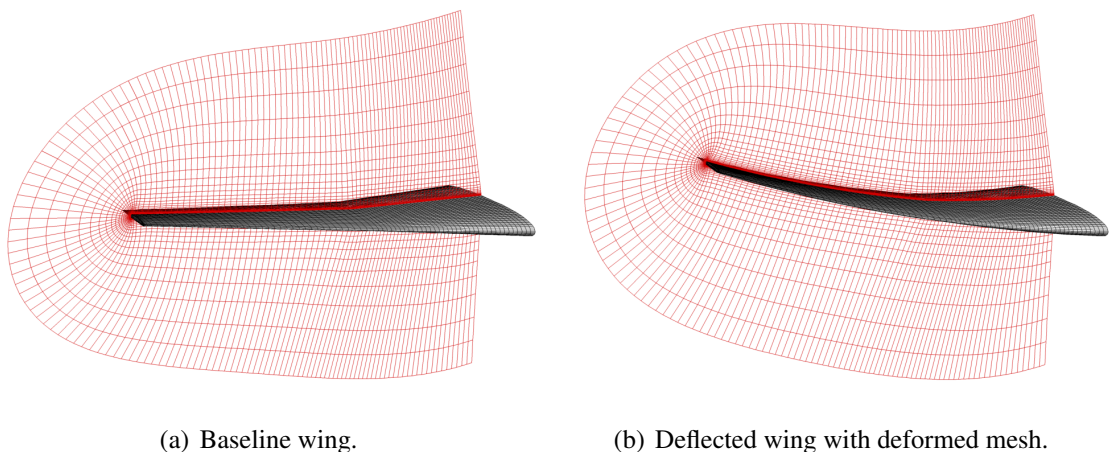


Figure 3.4: Application of the mesh deformation algorithm for a wing deflection case.

Given a baseline surface mesh and its corresponding deformed configuration, we can compute displacements and surface normal rotations for each surface node. Each surface

node defines a displacement field in the volume based on its own displacements and rotations. The displacement of each volume node is computed as a weighted average of the displacements predicted by the displacement fields of the surface nodes at the undeformed volume node location. The weights of the averaging process decay with the distance between the volume node and the corresponding surface node.

This mesh deformation procedure is applied in a component-wise manner, so that each overset mesh is deformed based only on its own wall surfaces. For instance, the wing surface nodes only affect the deformation of the wing volume mesh nodes, the fuselage surface nodes only deform the fuselage volume mesh nodes, and the collar surface nodes only deform the collar volume mesh nodes. This allows for relative motion between primary components. In other words, if the wing is translated with respect to the fuselage, the movement of the wing volume mesh nodes is not influenced by the fixed position of the fuselage surface nodes. The background mesh remains fixed throughout the optimization because it has no associated surface.

The component-based deformation methodology applied to overset meshes can be advantageous in geometries with oblique junction angles between two distinct components, such as the wing-strut junction of the SBW configuration. Volume cells in the gap between these components do not have opposing surfaces driving their displacements, as in the multiblock mesh case, thereby allowing for additional freedom of movement. Furthermore, the overset mesh cells in the junction regions are of higher quality than a patched multiblock mesh, which allows the former to tolerate a wider range of deformations before the mesh becomes invalid for CFD analyses.

pyWarp is also automatically differentiated by using Tapenade in reverse AD mode so it can compute the derivatives of volume node positions with respect to surface node movements.

3.5 CFD solver—ADflow

Once volume meshes are generated and deformed to reflect the changes in geometry, we use a CFD solver to perform aerodynamic analysis on the new configuration and to obtain lift and drag coefficients. These coefficients are then used by the objective and constraint functions of the optimization problem.

The aerodynamic solver used in this work is a newly implemented version of Sumb [27, 78] called ADflow [44]. It is capable of solving Euler, laminar Navier–Stokes, or RANS equations in multiblock structured overset meshes in a parallelized fashion using a second-order cell-centered finite volume formulation. The inviscid fluxes are discretized with

artificially dissipated central-differencing [79], whereas the viscous fluxes use standard central-differencing.

ADflow initially solves steady problems by using time-marching schemes (such as the diagonalized diagonal dominant alternating direction implicit scheme [80] or the multistage explicit Runge–Kutta scheme [81]) to approach the basin of attraction. It then switches to a Newton–Krylov algorithm to converge to the steady solution [82]. The switching criterion is defined as the point when the time residuals drop below a user-specified threshold.

This CFD solver uses implicit hole cutting [39, 83] to determine which cells should be blanked, interpolated, or actually computed, as explained in Sec. 1.2. We use trilinear interpolation in the dual mesh to compute the interpolated cell values. In other words, each interpolated cell will compute its value by interpolating the flow state on a stencil of eight cells from an overlapping mesh.

Once the flow converges, zipper meshes are used to fill the surface gaps among surface compute cells, yielding a watertight surface for the integration of the aerodynamic forces and moments [84] (Fig. 3.5). In this work, we update the overset connectivities and zipper meshes at each optimization iteration. Further details regarding the overset implementation in ADflow are presented by Kenway et al. [44].

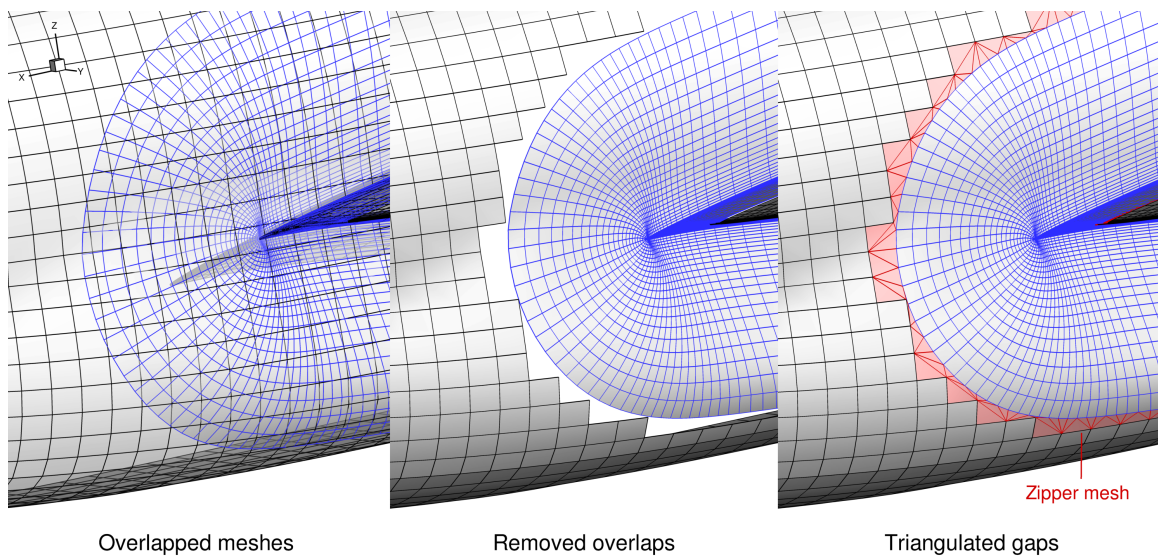


Figure 3.5: Zipper meshes are used to fill the gaps among overlapped meshes for force integration. They are not employed during the flow solution.

ADflow’s drag predictions were previously evaluated in the Sixth Drag Prediction Workshop (DPW6) [85]. One important conclusion from DPW6 was that the standard Spalart–Allmaras (SA) turbulence model [86] could overestimate the size of separation bubbles in wing–fuselage junctions [87], and that this could be mitigated by applying modifications to

the turbulence model, such as the rotation correction (R) [88], and the quadratic constitutive relation (QCR) [89]. Therefore, we use the SA-R-QCR2000 turbulence model for all simulations in this work, as discussed in Sec. 4.3.

ADflow computes the sensitivities of the aerodynamic forces and moments with respect to the nodal coordinates with the hybrid adjoint approach [27, 30], in which we use Tape-nade’s reverse mode AD to compute the partial derivatives in the discrete adjoint equations, as explained in Sec. 3.7.

The derivatives currently computed by ADflow assume frozen overset interpolation weights. In other words, the linearized code does not consider how changes in mesh coordinates affect the overset interpolation weights. The interpolation expression is linear with respect to the flow state variables of the donor cells, and this part is properly linearized in the reverse AD code. However, finding the interpolation weights with respect to the mesh coordinates involves an iterative Newton search, since this requires a solution of a nonlinear system. Therefore, the computation of the partial derivatives of interpolation weights with respect to nodal coordinates using reverse AD requires additional considerations that are under development in ADflow. This discrepancy in the reverse AD code becomes less significant as the overset mesh is refined because flow state variations within the interpolation stencils become smaller.

3.6 Optimizer—SNOPT

We use Sparse Nonlinear Optimizer (SNOPT) [90], which is a gradient-based optimizer that implements the sequential quadratic programming method. The user must provide functions of interest and their gradients. SNOPT can handle large-scale nonlinear optimization problems with thousands of constraints and design variables, making it suitable for aerodynamic shape and aerostructural optimizations [9, 44, 61, 75].

3.7 Derivative computation throughout the framework

ASO problems usually involves a number of design variables that is much greater than the number of functions of interest. The adjoint method is an efficient way to compute derivatives [23, 26, 27] in this scenario. However, a significant effort is required to implement this method because it needs partial derivatives of the residual equations and other quantities used in the CFD code.

Let $f = f(\mathbf{X}, \mathbf{W})$ be a function of interest for the optimization problem (such as drag coefficient). This function depends on the design variables \mathbf{X} and the flow state variables

\mathbf{W} . The flow state variables are obtained from the solution of the discretized flow equations: $\mathbf{R}(\mathbf{X}, \mathbf{W}) = \mathbf{0}$. The total derivative of the function f with respect to the design variables is:

$$\frac{df}{d\mathbf{X}} = \frac{\partial f}{\partial \mathbf{X}} - \boldsymbol{\psi}^T \cdot \frac{\partial \mathbf{R}}{\partial \mathbf{X}}, \quad (3.1)$$

where the adjoint variables $\boldsymbol{\psi}$ can be obtained by solving the discrete adjoint system:

$$\begin{bmatrix} \frac{\partial \mathbf{R}}{\partial \mathbf{W}} \end{bmatrix}^T \cdot \boldsymbol{\psi} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{W}} \end{bmatrix}^T. \quad (3.2)$$

Our optimization framework uses a hybrid-adjoint approach [27], where the partial derivatives in Eqs. (3.1) and (3.2) are computed using the reverse-mode AD of Tapenade.

In order to use the hybrid adjoint method, it is useful to cast the optimization modules from Fig. 3.1 in a different manner. We only need to gather the subroutines that relevant for the direct computation of the flow solver residuals (\mathbf{R}) and the function of interest (f), since we only want to compute partial derivatives of these quantities. This means that we can leave the iterative methods for flow solution and the ADT searches used in the intersection code outside of the reverse chain.

Figure 3.6 shows such arrangement for the modules used in this work. If we apply the reverse AD method to each module and reorganize them in the reverse order, we get the chain shown in Figure 3.7. The backpropagation of derivatives starts at the last code of the analysis chain, which is the CFD solver.

For given derivative seeds of the CFD residuals ($\bar{\mathbf{R}}$) and function of interest (\bar{f}), ADflow uses reverse AD versions of its subroutines to obtain derivative seeds for three parameter sets: flow state variables ($\bar{\mathbf{W}}$), volume mesh nodes ($\bar{\mathbf{r}}_{\text{vol}}$), and flow condition design variables ($\bar{\mathbf{X}}_{\text{aero}}$), such as angle of attack. Lyu et al. [30] give specific details on differentiating the RANS equations in ADflow.

The volume mesh derivative seeds ($\bar{\mathbf{r}}_{\text{vol}}$) are split based on their corresponding components (such as wing, fuselage, and collar), and then transformed into surface node derivatives ($\bar{\mathbf{r}}_{\text{surf}}$) by pyWarp. Next, pySurf receives the derivatives of the collar surface mesh points and executes its own reverse-mode AD to backpropagate the derivative information to the triangulated surface nodes ($\bar{\mathbf{r}}_{\text{tria}}$).

Then, pyGeo accumulates the derivatives coming from the triangulated surfaces (given by pySurf) and from the structured surface meshes (given by pyWarp) of each primary component. Finally, pyGeo uses the FFD mapping to backpropagate derivatives to the FFD control points, which are translated into derivatives with respect to design variables ($\bar{\mathbf{X}}_{\text{w}}$ for wing design variables and $\bar{\mathbf{X}}_{\text{f}}$ for fuselage design variables).

The derivative seeds of the geometric design variables of the wing and fuselage, ($\overline{\mathbf{X}}_w$ and $\overline{\mathbf{X}}_f$), are then concatenated with the flow condition design variables ($\overline{\mathbf{X}}_{\text{aero}}$) to yield the derivative seeds of the complete design variable vector ($\overline{\mathbf{X}}$), closing the reverse AD propagation. pyHyp is not included in Fig. 3.7 because it is only used during the initialization step, not during the optimization itself.

We now explain how to use the chain of Fig. 3.7 to solve Eqs. (3.1) and (3.2). According to Eq. (2.1), the reverse derivative seeds from Fig. 3.7 are related as follows:

$$\overline{\mathbf{W}} = \left[\frac{\partial \mathbf{R}}{\partial \mathbf{W}} \right]^T \cdot \overline{\mathbf{R}} + \left[\frac{\partial f}{\partial \mathbf{W}} \right]^T \cdot \overline{f}, \quad (3.3a)$$

$$\overline{\mathbf{X}} = \left[\frac{\partial \mathbf{R}}{\partial \mathbf{X}} \right]^T \cdot \overline{\mathbf{R}} + \left[\frac{\partial f}{\partial \mathbf{X}} \right]^T \cdot \overline{f}. \quad (3.3b)$$

If we feed the reverse AD chain with $\overline{\mathbf{R}} = \mathbf{0}$ and $\overline{f} = 1$, then the resulting $\overline{\mathbf{W}}$ will be the right-hand side of Eq. (3.2). Now, if we use $\overline{\mathbf{R}} = \mathbf{g}$ and $\overline{f} = 0$, then $\overline{\mathbf{W}}$ will be the matrix-vector product of the left-hand side of Eq. (3.2) for an arbitrary vector \mathbf{g} . Therefore, we can use matrix-free linear system solvers to solve Eq. (3.2) and obtain ψ . In this work we use the generalized minimal residual method (GMRES) [91] implementation in PETSc [92–94] to solve the linear system. Finally, if we apply $\overline{\mathbf{R}} = -\psi$ and $\overline{f} = 1$ to the reverse AD chain, we get, according to Eqs. (3.1) and (3.3b), $\overline{\mathbf{X}} = [df/d\mathbf{X}]^T$, which is the total derivative of the function of interest required by the optimizer.

This procedure must be applied for every function of interest to assemble and solve the corresponding adjoint systems. Because we use reverse AD throughout the entire analysis chain, the cost of computing derivatives scales with the number of functions of interest and not with the number of design variables.

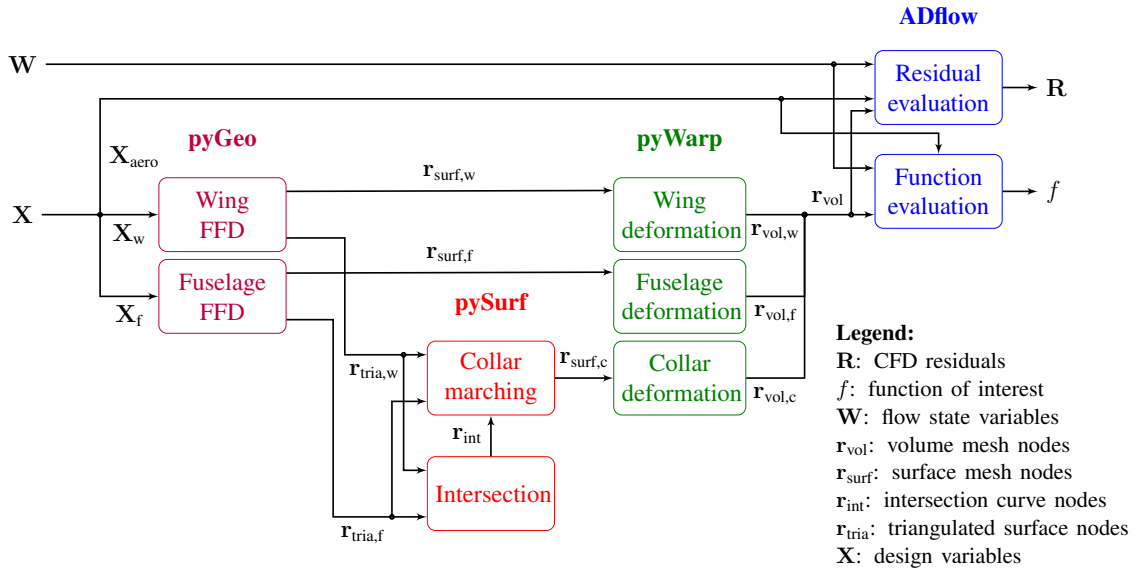


Figure 3.6: Subset of the framework subroutines required for the computation of CFD residuals (\mathbf{R}) and functions of interest (f).

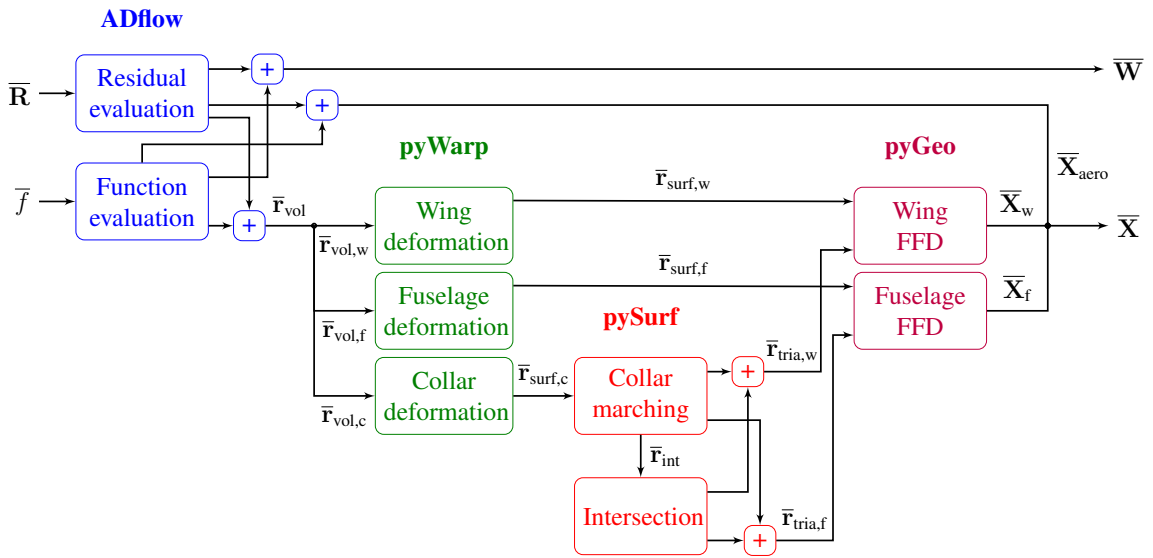


Figure 3.7: Derivative backpropagation chain. The modules represented here are the reverse counterparts of the modules shown in Fig. 3.6. The plus signs indicate that the derivative seed should be accumulated from multiple sources. Subscripts ‘w’, ‘f’, ‘c’, and ‘aero’ indicate that quantities refer to wing, fuselage, collar, and flow conditions, respectively.

3.8 Noise issues in the functions of interest

We integrated pySurf into the framework and now seek to verify the overall consistency of the derivatives. We propose a simple case to verify the accuracy of the computed derivatives in a transonic flow application. We generate a simple geometry consisting of a tapered, swept wing based on that of a Boeing 717 and a cylindrical fuselage, which is then simulated in transonic flow conditions (Mach 0.8). We define a single design variable to the geometry: the wing vertical position with respect to the fuselage (Fig. 3.8). We generate the surface meshes of the primary components in ICEMCFD. The fuselage mesh remains fixed while the wing mesh is vertically translated according to the design variable value. The fuselage-wing intersection curve and the collar mesh are automatically generated by pySurf for each design point.

Then, we run flow simulations and derivative computations for a range of wing positions; the results are shown in Fig. 3.9. The derivatives are consistent with the overall trend of the function. However, the function shows noisy patterns with amplitudes of hundredths of drag counts for small displacements (right side of Fig. 3.9).

We ran a similar design variable sweep with a finer version of the structured meshes. Figure 3.10 shows that the noise level is smaller for finer meshes. This indicates that the noise is neither associated with the physics of the model nor with the triangulated surface discretization, since they remained the same for both cases.

The drag coefficient curve shown in Fig. 3.9 suggests that there is a minimum around $y_{wing} = 0.5$ m. Thus, we use this same test case to verify the effects of the noise on optimization problems in the next section.

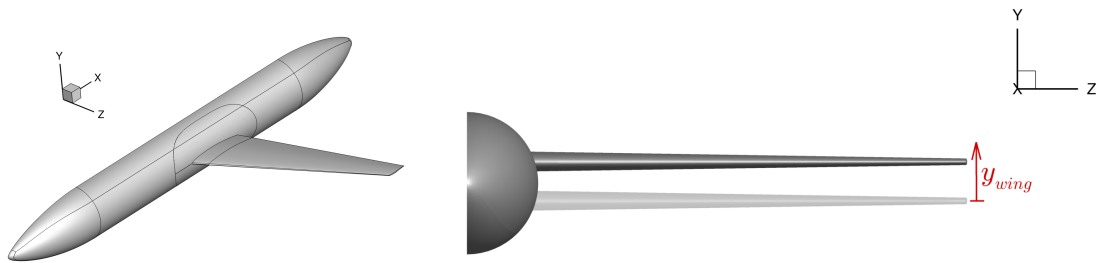


Figure 3.8: Transonic configuration used for wing translation study. The y_{wing} design variable controls the vertical displacement of the wing.

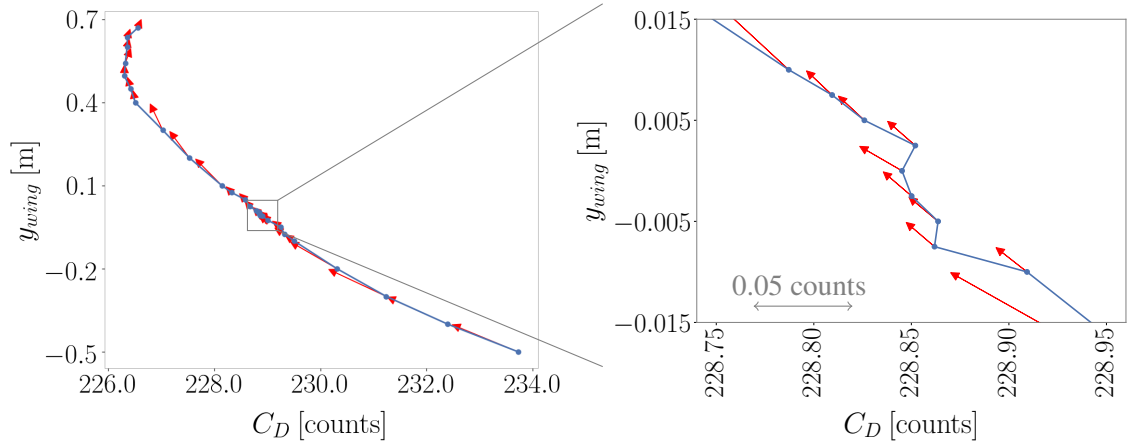


Figure 3.9: Drag variations due to the vertical translation of the wing of the transonic airplane configuration of Fig. 3.12. The red arrows represent gradients. We observe noise at small steps (on the right), but the gradients are still consistent with the overall trend.

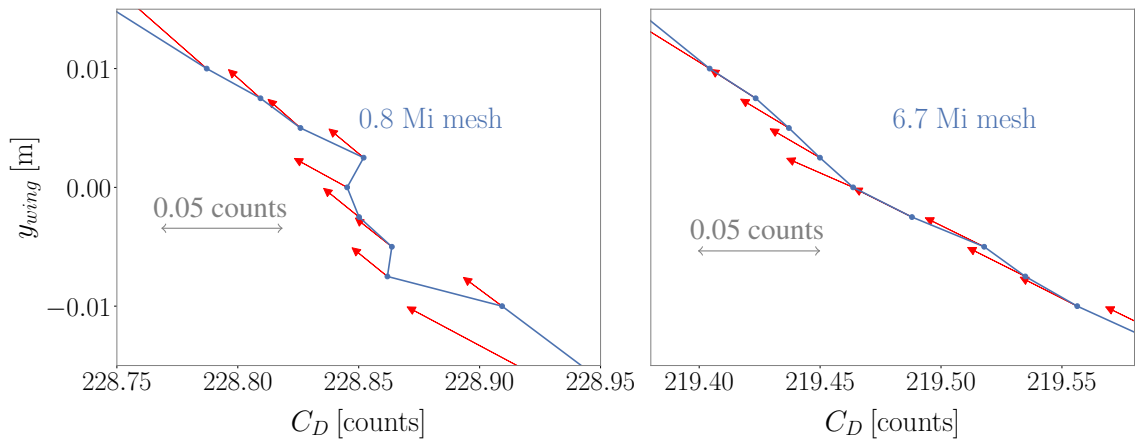


Figure 3.10: Effects of the CFD mesh refinement on the noise levels for the same wing translation problem shown in Fig. 3.12. The noise levels are smaller when we use finer versions of the CFD structured meshes.

3.8.1 Effects of numerical noise on a univariate optimization problem

We study the effects of the noise on a univariate optimization problem before moving to a more complex case. Here we use the same transonic configuration introduced in the previous section. The coarse CFD mesh (0.8 million cells) is used for the optimization so that the noise effect becomes more evident. We control the vertical position of the wing relative to the fuselage to minimize drag for a fixed angle of attack.

Figure 3.11 shows the optimization progress. The optimizer gets trapped in a valley generated by the noise when it reaches the shallow design space surrounding a potential

optimum, where it halts due to numerical difficulties caused by inconsistencies between the gradient and the actual trend of the function.

Noise becomes an issue in regions of the design space where the gradients are small, such as near the optimum, since the relative error between the gradient and the actual trend of the noisy function becomes significant. The function variation predicted by gradient gets smaller as we approach the optimum point, where the gradient magnitude should be zero (assuming an unconstrained problem). At some point, the noise amplitude exceeds the variations predicted by the gradient. Therefore, even though the gradient indicates the descent direction for the function, the noise may actually increase the function value in that direction. This inconsistent trend traps the optimizer in the noise ravines shown in Fig. 3.11.

The optimization still improves the baseline design and converges towards the expected optimum. This indicates that the noise only becomes an issue at the later steps of the optimization, when the optimizer is only making small changes to narrow the optimum point. At this stage, most of the optimization benefits have already been achieved. This gives us confidence to employ the same methodology to more complex cases, as we show in Chapter 4 and Chapter 5.

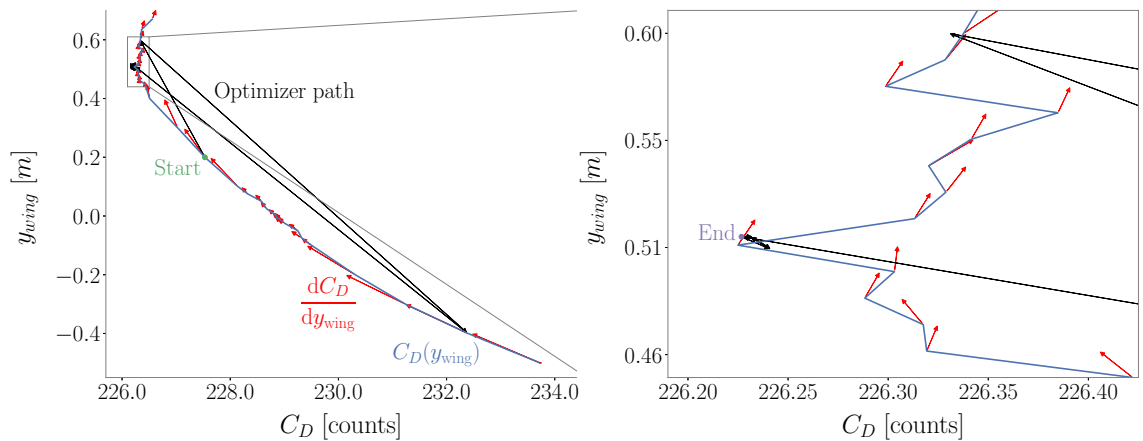


Figure 3.11: Optimization path for the wing translation problem. The plot on the right is a zoomed-in view showing the last iterations of the optimizer. The optimizer is trapped in a valley caused by the noise, but it manages to improve the baseline design.

3.8.2 Identification of the noise source

We showed that the functions of interest obtained from the CFD evaluations contain noise. However, the results from Sec. 2.4.5 suggest that no significant noise appears in the mesh point coordinates due to the mesh regeneration based on triangulated surfaces. Therefore,

we suspect that the noise originates from changes in the overset connectivity.

To clarify this point, we create another test case in which the regeneration of collar meshes with triangulated surfaces is not necessary. We take the same wing-body geometry introduced in Sec. 3.8 and define a different design variable for the geometry: the longitudinal wing position with respect to the fuselage (Δx_{wing} shown in Fig. 3.12). The fuselage mesh remains fixed while the wing mesh is horizontally translated according to the value of the design variable.

This time, the intersection line retains the same shape because the wing is translated in the longitudinal direction along the cylindrical fuselage. Therefore, there is no need to regenerate the collar mesh using the triangulated surfaces, though the effects of changes in overset connectivity remain. The overset mesh used for the analysis has only 0.3 million cells so that the effects of changes in connectivity become more evident.

We next run flow simulations for a range of wing positions; the results are shown in Fig. 3.13. The function shows noisy patterns with amplitudes on the order of tenths of drag counts for small wing displacements (see right side of Fig. 3.13). The noise drops to the order of hundredths of drag counts if the mesh is refined to 1.3 million cells because the interpolation layers become thinner. Therefore, we conclude that the main contribution to the noise comes from changes in overset connectivity.

One possible way to solve this problem is by switching to a frozen overset connectivity approach near the end of the optimization. However, this would also require the simultaneous deformation of all overset meshes in order to keep the relative position of the CFD cells, which in its turn prevents the use of the component-based parametrization since all components have to be deformed all at once. Therefore, this methodology only supports small deformations, and it would only be suitable for the latter stages of the optimization.

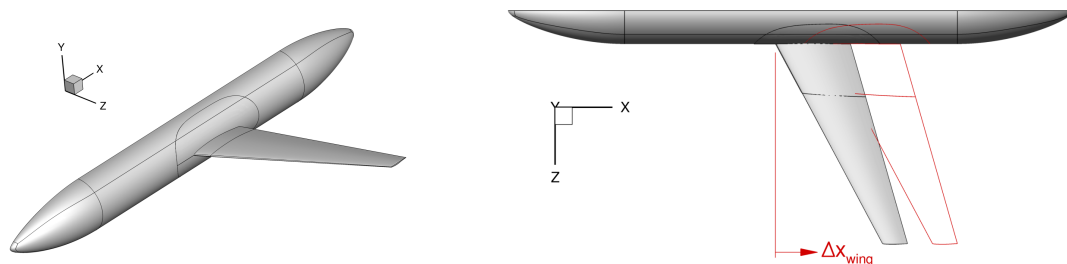


Figure 3.12: Transonic configuration used to study wing translation. The design variable Δx_{wing} controls the horizontal displacement of the wing.

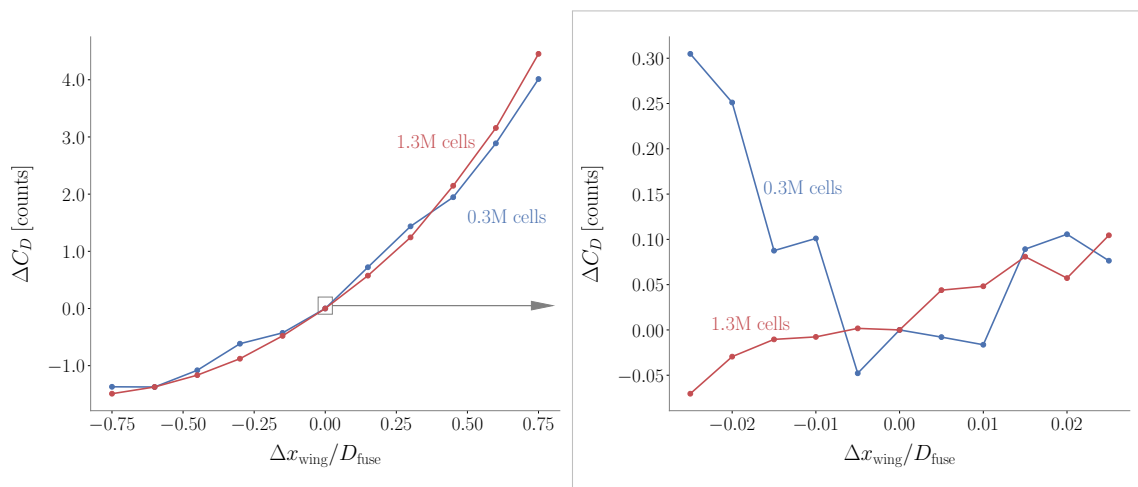


Figure 3.13: Variations in drag for different horizontal wing positions normalized by the fuselage diameter (D_{fuse}). The right plot shows drag variations for small displacements. The noise caused by changes in overset connectivity decreases as the mesh is refined.

CHAPTER 4

Wing-body junction optimization

In this chapter, we demonstrate the capabilities of the pySurf module by optimizing the shape of the wing-body junction of a conventional aircraft configuration. We choose the DLR-F6 wing-body configuration [95] since this model was extensively used for drag prediction benchmarks [76, 96] and also as a baseline for aerodynamic shape optimization problems [29, 41, 97].

The baseline DLR-F6 configuration shows a separation bubble at the trailing edge of the wing-body junction at $C_L = 0.5$ [96], which makes this problem particularly interesting for wing-body junction optimizations, as discussed in Sec. 1.4. A fairing was designed for the DPW3 to remove the recirculation region and improve drag convergence studies [98], and this configuration is named DLR-F6-FX2B.

The wing-body separation bubble seen on the DLR-F6 configuration served as motivation to use the MACH framework, now with the pySurf module, to perform aerodynamic shape optimization of the DLR-F6 wing-body junction starting from its original geometry. Differently from the previous cited approaches, our tool uses the reverse AD chain throughout the entire framework for efficient derivative computation. Furthermore, the component-based parametrization approach also allows the simultaneous optimization of the wing and the junction.

Since pySurf enables independent manipulation of the wing and fuselage, we define four major groups of design variables: angle of attack, wing twist, wing shape, and fuselage shape variables. We conduct a series of optimizations using combinations of these groups to investigate the effects of each design variable in the overall improvement of the design. The objective of all optimizations described here is the minimization of drag coefficient (C_D) at $C_L = 0.5$, $M = 0.75$, and $Re = 5 \times 10^6$, following the same standards as DPW3. Table 4.1 shows a summary of the overall optimization problem.

Table 4.1: DLR-F6 aerodynamic shape optimization problem.

	Variable/function	Description	Quantity
Minimize	C_D	Drag coefficient	
with respect to	α	Angle of attack	1
	n_{fuse}	Normal displacement of fuselage FFD control points	16
	τ_{wing}	Wing section twist	8
	z_{wing}	Vertical displacement of wing FFD control points	128
		Total design variables	153
subject to	$C_L = 0.5$	Lift constraint	1
	$t/t_{\text{init}} \geq 0.99$	Wing thickness constraint	100
	$\Delta z_{\text{wing,TE,upper}} = -\Delta z_{\text{wing,TE,lower}}$	Fixed trailing edge	8
	$\Delta z_{\text{wing,LE,upper}} = -\Delta z_{\text{wing,LE,lower}}$	Fixed leading edge	8
	$0.00 \text{ m} \leq n_{\text{fuse}} \leq 0.04 \text{ m}$	Variable bounds	16
		Total constraints	133

4.1 Geometric design variables and constraints

The geometric design variables affect the position of the FFD control points. The wing FFD is composed of a volumetric block with 8 spanwise by 8 chordwise by 2 vertical nodes (Fig. 4.1). The 8 wing twist variables (τ_{wing}) apply uniform rotations to each chordwise section of FFD control points. This rotation of each section is centered around its quarter-chord position. The 128 wing shape variables (z_{wing}) move each control point of the FFD in the vertical direction to modify the airfoil shape.

To avoid shear twist, control points on either side of the leading and trailing edges are constrained to move equal amounts but in opposite directions. This ensures that the airfoil chordline is affected only by twist variables and not by shape variables [99].

Because this is a purely aerodynamic shape optimization problem, we also need to enforce thickness constraints to obtain a realistic design. In this case we define a grid of 10×10 points over the wing planform (Fig. 4.2) and then require the thickness at these points to remain at or above 99% of the baseline thicknesses.

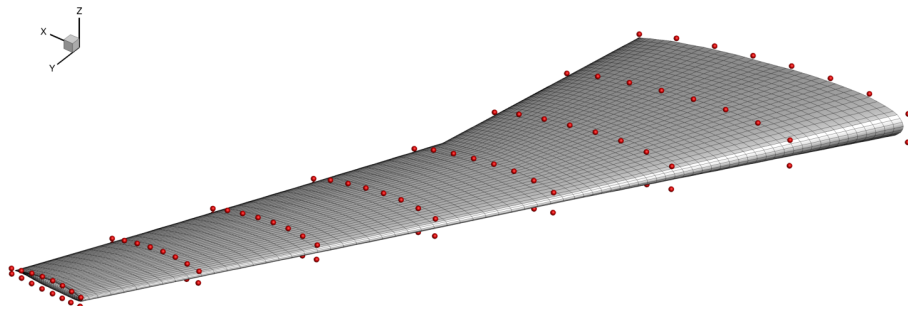


Figure 4.1: Wing FFD showing the control points (red dots), which are manipulated by the twist and vertical displacement variables.

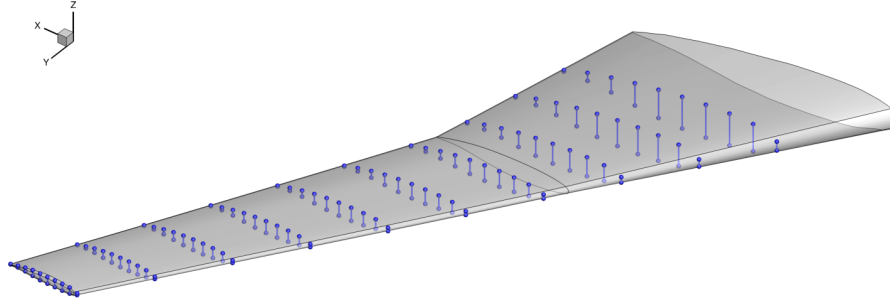


Figure 4.2: Pairs of points used to define thickness constraints. The distance between the pair of points cannot decrease during the optimization.

The fuselage FFD consists of a block of $8 \times 6 \times 2$ nodes that encompasses the region around the wing intersection (Fig. 4.3). The nodes in the outer two layers of the FFD are fixed to guarantee C_1 continuity with the undeformed part of the fuselage [100], which leaves a total of 16 free control points. These control points are only allowed to move in a direction normal to the fuselage surface and away from its center, thereby preserving its original volume.

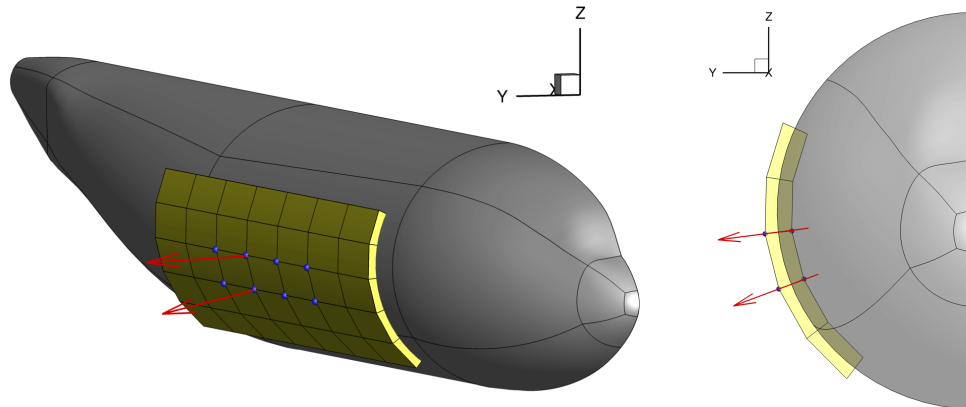


Figure 4.3: Fuselage FFD showing the free control points as blue dots. The other control points remain fixed to guarantee C_1 continuity within the undeformed region.

To better understand this design optimization problem, we progressively increase the problem complexity by introducing groups of design variables until we reach the desired formulation expressed in Table 4.1. The sequence of five optimization problems is listed in Table 4.2.

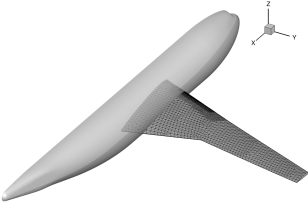
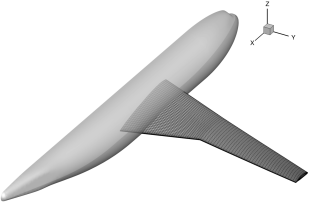
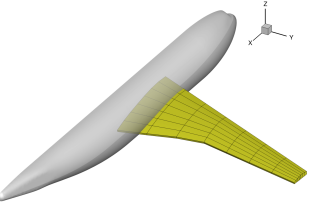
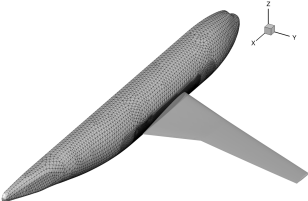
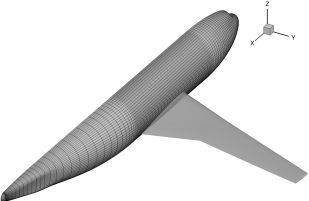
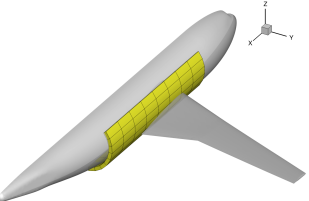
Table 4.2: Identification tags for each optimization problem.

Tag	Active design variables	Description
B	α	C_L matching for the baseline configuration
F	α, n_{fuse}	Fairing optimization
F+T	$\alpha, n_{\text{fuse}}, \tau_{\text{wing}}$	Fairing and wing twist optimization
F+T+S	$\alpha, n_{\text{fuse}}, \tau_{\text{wing}}, z_{\text{wing}}$	Fairing, wing twist, and wing shape optimization
T	$\alpha, \tau_{\text{wing}}$	Wing twist optimization
T+S	$\alpha, \tau_{\text{wing}}, z_{\text{wing}}$	Wing twist and shape optimization

4.2 Problem setup

To set up the analysis and optimization cycle in the MACH framework, we must provide triangulated surfaces, structured surface meshes, and FFD boxes for all primary components. We therefore import the IGES files of the DLR-F6 configuration in ICEM CFD [101] to generate these inputs, which are listed in Table 4.3. The wing structured surface mesh has 127,488 quadrilaterals, and the fuselage structured surface mesh has 105,344 quadrilaterals. The triangulated surface representation of the wing has 9,948 elements, whereas the fuselage triangulated mesh has 19,902 elements, making a total of 29,850 triangles, which are mainly concentrated in the region of the wing-fuselage intersection. The wing and fuselage triangulated surfaces are also refined in the high-curvature regions (such as the leading edge), where we expect the collar mesh to grow.

Table 4.3: Inputs required by pySurf; these are generated by ICEM CFD using the original IGES representation of the DLR-F6 model.

	Triangulated surfaces	Structured surface meshes	FFD blocks
Wing			
Fuselage			

While generating the triangulated surface, we also create line segments to outline important features of the geometry, such as the two corners of the blunt trailing edge. We project the marched collar mesh nodes to the line segments to ensure that the corresponding features are represented in the CFD mesh, as explained in Sec. 2.3.

In the next step, we use pySurf to perform the following operations on the triangulated surfaces to generate the collar mesh:

1. Compute the intersection between the wing and fuselage triangulated surfaces.
2. Split the intersection curve at the wing leading edge and at the trailing edge corners; this splits the intersection curve into three segments: upper skin, lower skin, and blunt trailing edge.
3. For each of these segments, create node distributions that are appropriate for CFD analysis (for instance, with refined leading and trailing edges).
4. Merge the three remeshed curve segments back into a single curve.
5. Use the merged curve as a starting feature for the hyperbolic surface mesh marching on the wing and on the fuselage to generate the collar mesh.

The execution of these steps generates the collar mesh shown in Fig. 4.4. These operations are repeated on every optimization iteration to regenerate the collar meshes. pySurf also saves the order of these operations to execute them in the reversed order during the reverse propagation of derivatives. The collar mesh generation parameters are manually adjusted so that the cells from overlapped meshes have similar sizes at the boundaries.

In the initialization step of the optimization, we use pyHyp to automatically extrude the surface meshes into near-field volume meshes. The hyperbolic extrusion parameters, such as initial cell height and growth ratios, follow the DPW3 guidelines for the coarse mesh level. We enforce slightly smaller values of these parameters (95%) for the collar mesh to ensure that its cells have higher priority of preservation during the implicit hole cutting process. The near field meshes are extruded up to 2.5 times the mean aerodynamic chord of the model.

The background mesh, which extends the domain to the far-field, is generated with the two steps discussed in Sec. 3.3: The background mesh reaches 100 mean aerodynamic chords, which is the far-field distance specified by DPW3. The complete overset mesh has 1.1 million cells. Figure 3.3 displays the symmetry plane of the overset meshes. Now that we have a volume mesh, we can proceed to the aerodynamic analysis.

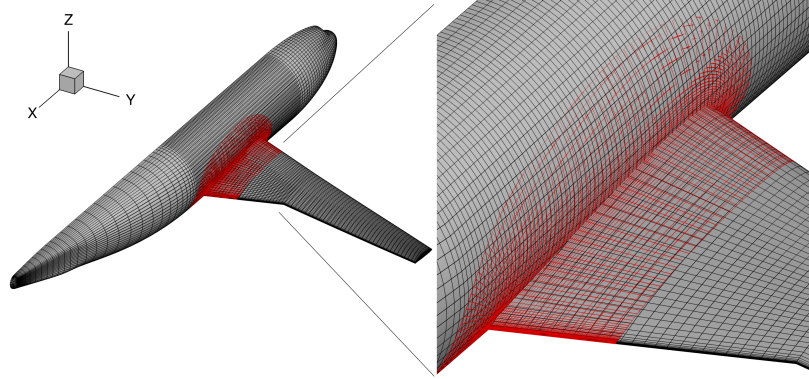


Figure 4.4: Structured surface meshes used for the volume mesh extrusion. The primary component meshes (wing and fuselage) are shown in black, and the automatically generated collar mesh is shown in red.

4.3 Baseline configuration studies

We run preliminary aerodynamic analysis on the baseline configuration to compare results with the DPW3 data set. This allows us to verify if the flow solution procedure is consistent with other CFD codes and wind tunnel measurements.

We use the procedure described in the previous section to generate meshes for both the baseline DLR-F6 configuration and the one with the FX2B fairing, since both configurations were studied in DPW3. The mesh topology and number of cells is the same for both geometries. ADflow simulations show the same trends reported in DPW3: the baseline version shows separation on the wing-body junction trailing edge, while the FX2B fairing removes it (Fig. 4.5).

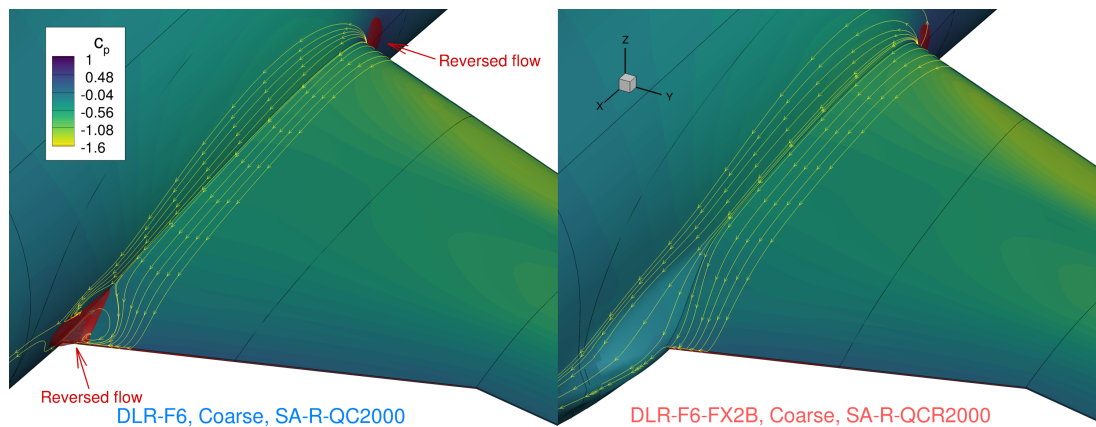


Figure 4.5: Wing-body junction flow patterns predicted by ADflow for the DLR-F6 configuration. The baseline geometry shows separation on the junction trailing edge (left), while the addition of the FX2B fairing removes it (right). This is the same trend observed in DPW3.

One important conclusion from previous drag prediction workshops was that the standard Spalart–Allmaras (SA) turbulence model [86] could overestimate the size of separation bubbles in wing-body junctions compared to wind tunnel measurements [85, 87, 102]. This effect could be mitigated by applying modifications to the turbulence model, such as the rotation correction [88], and the quadratic constitutive relation (QCR) [89]. The anisotropy in the normal turbulent stress introduced by these corrections generates secondary flow features at the wing-body junction corner that accelerate the flow and reduce the boundary layer thickness in this region, thus delaying flow separation at the junction trailing edge [87, 102]. Therefore, we modified the turbulence model in ADflow to take these corrections into account.

We generate three mesh levels for a grid refinement study (Table 4.4). We compare the effects of grid refinement for the standard SA model and also for the modified version with rotation correction and QCR relation (SA-R-QCR2000). The separation bubble of the baseline configuration obtained from different mesh levels and turbulence models is shown in Fig. 4.6. The standard SA model predicts a larger separation bubble as the grid is refined, while the bubble size has small variations due to grid refinement for the SA-R-QCR2000 model. The latter yields better agreement with experimental data and better drag convergence characteristics, as seen in the DPW6 [85].

Table 4.4: Mesh levels used for drag convergence study. The maximum y^+ values are computed based on the converged CFD results.

Level	Number of cells (million)	$\max y^+$
Coarse	1.1	1.4
Medium	3.0	1.1
Fine	8.8	0.6

Figure 4.7 shows the drag coefficients obtained for different mesh levels and turbulence models along with the extrapolated values for the infinitely refined mesh ($N^{-2/3} = 0$). The results of the baseline configuration obtained with standard SA model (DLR-F6 SA curve in Fig. 4.7) show a change in slope caused by the unphysical growth of the separation bubble after the mesh refinement. Conversely, the QCR model curves do not show the same change in slope, since the bubble growth is contained by the modified turbulence model. The same applies to the DLR-F6-FX2B problem because the separation bubble is negligible in all mesh levels due to the fairing.

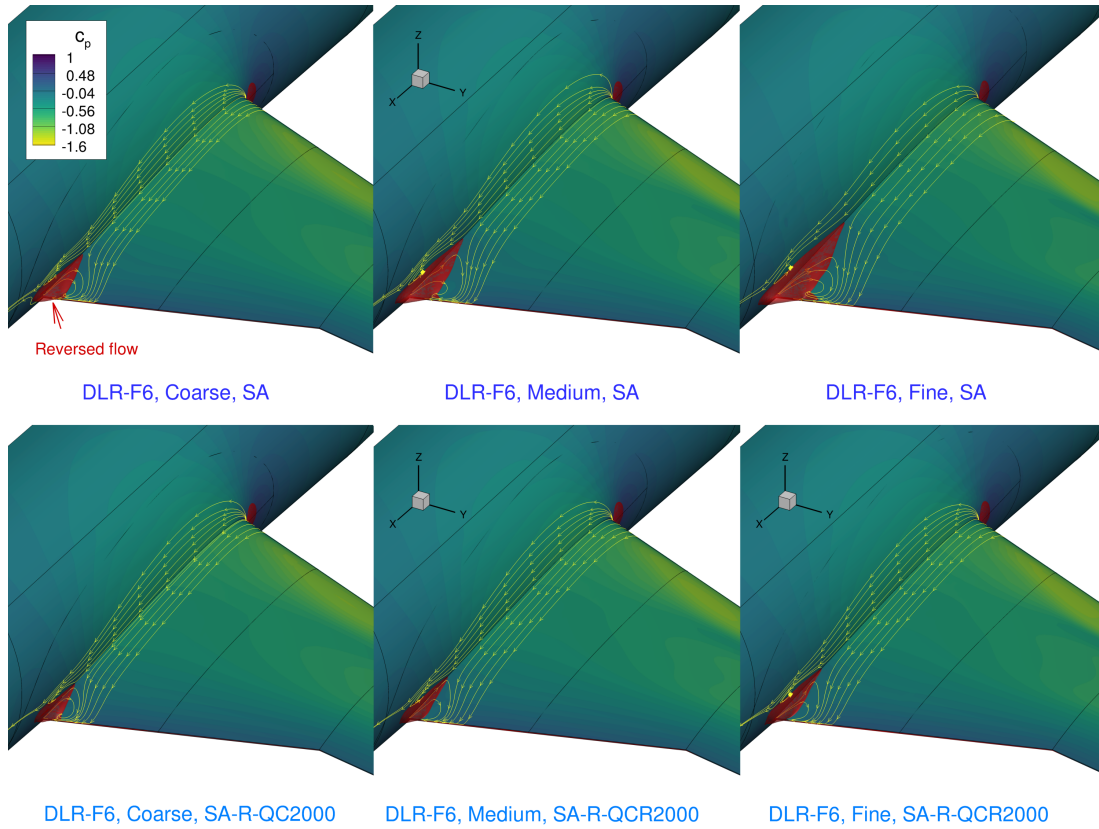


Figure 4.6: Effect of grid refinement and turbulence model variant in the recirculation bubble size.

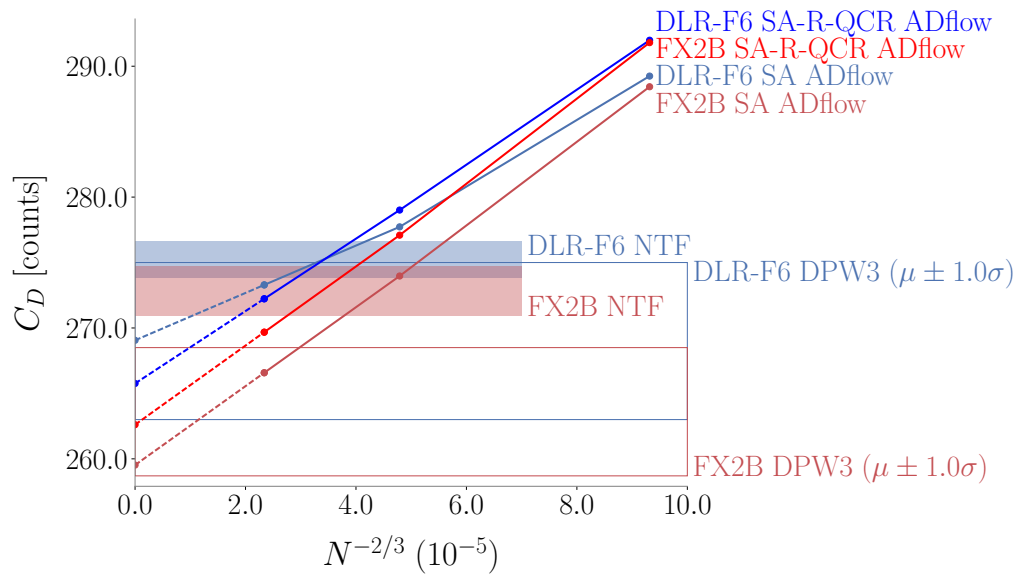


Figure 4.7: Drag convergence study for the DLR-F6 configurations. N represents the number of cell in the CFD mesh. Blue colors refer to the baseline DLR-F6 configurations and red colors refer to the DLR-F6-FX2B configuration.

Figure 4.7 also compares ADflow results against the average values from DPW3 [76] and experimental data from the National Transonic Facility (NTF) [103]. The shaded area represents the uncertainty in the wind tunnel measurements. ADflow predictions are within one standard deviation from the average values reported in DPW3. The drag coefficient for the infinitely refined mesh predicted by ADflow with SA-R-QCR2000 for the baseline configuration is 9.4 counts lower than the wind tunnel measurement. On the other hand, ADflow predicts a reduction of 3.1 drag counts due to the addition of the fairing, while the drag reduction measured in wind tunnel tests is of 2.4 counts. This difference of 0.7 drag counts between the experimental and computational results corresponds to 0.2% of the total drag value measured in the wind tunnel. This indicates that the relative variations predicted by the numerical tools used in this work have enough accuracy for practical applications, and is thus suitable for optimization purposes.

We also addressed the effect of the triangulated surface refinement on the CFD results. Refining the triangulated surface from 29 thousand to 1.3 million triangles for the collar mesh generation causes an average displacement of 0.02 mm for the CFD mesh nodes at the wing-fuselage intersection line and a variation of 0.05 drag counts for the aerodynamic analysis of the baseline configuration, as shown in Fig. 4.8. Therefore, the optimization results obtained with the former triangulated surface fall within analysis and manufacturing tolerances.

All optimization cases to be shown in the next sections use the SA-R-QCR2000 turbulence model. In addition, we use the coarse mesh level (1.1 million cells) and to limit the computational cost of each CFD solution. We also use the coarse triangulated surface representation (29 thousand cells) to reduce the pre-processing cost of the optimization.

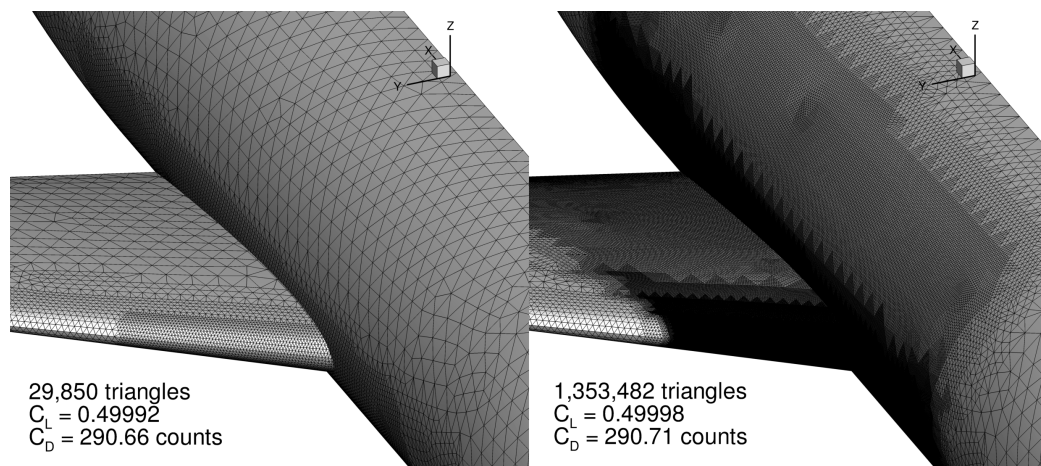


Figure 4.8: Effect of refinement of triangulated surface over CFD results. The coarse triangularization is used for optimization.

4.4 Optimization results

We first optimize the DLR-F6 configuration just with respect to the fuselage shape variables (Problem F in Table 4.2). All other design variables presented in Table 4.1 are fixed in this case. Figure 4.9 shows the convergence of the optimization problem, and we achieve a reduction of approximately 5 drag counts. Figure 4.10 shows details of the optimized fairing. The designed fairing completely eliminates the recirculation regions, as shown in Fig. 4.11.

Figure 4.9 also shows that some design variables reach the prescribed bounds. The fairing extends even more if we relax these bounds, but excessive stretching leads to mesh warping and oversight hole cutting problems. The explanation for this behavior is that since the optimizer does not have the freedom to modify the wing geometry, it is (inefficiently) using the fairing to change the wing lift distribution. The fairing gets bigger to unload the inboard region of the wing and shift the lift distribution towards the tip, as shown in Fig. 4.12.

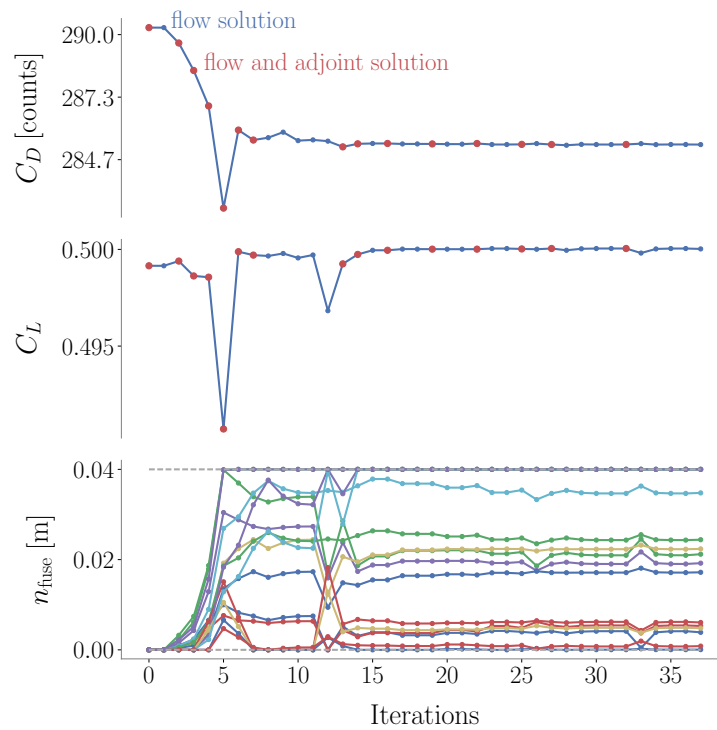


Figure 4.9: Fairing-only optimization history. Some normal displacement design variables went to the lower boundary of 0.04 m.

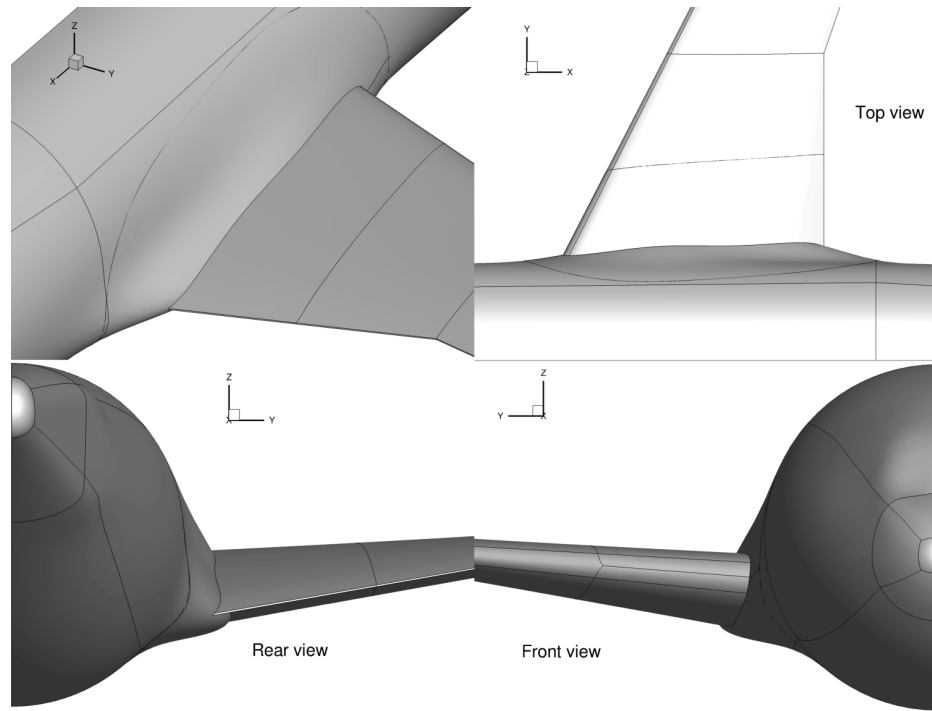


Figure 4.10: Optimized fairing for the fairing-only optimization (Problem F).

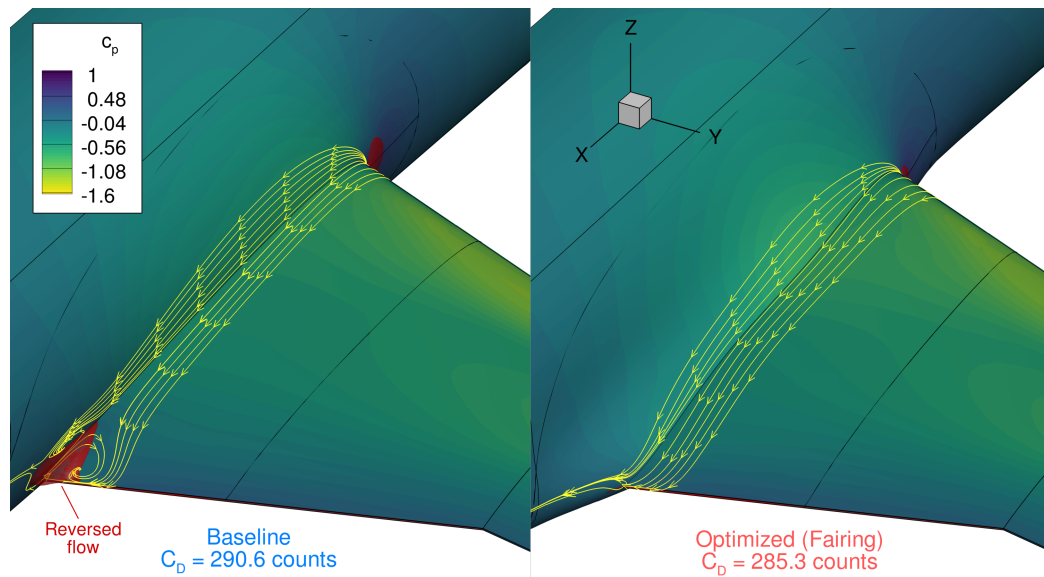


Figure 4.11: Wing-body junction flow before and after the fairing optimization (Problem F). Red regions indicate reversed flow. The redesigned fairing reduces the recirculation bubble.

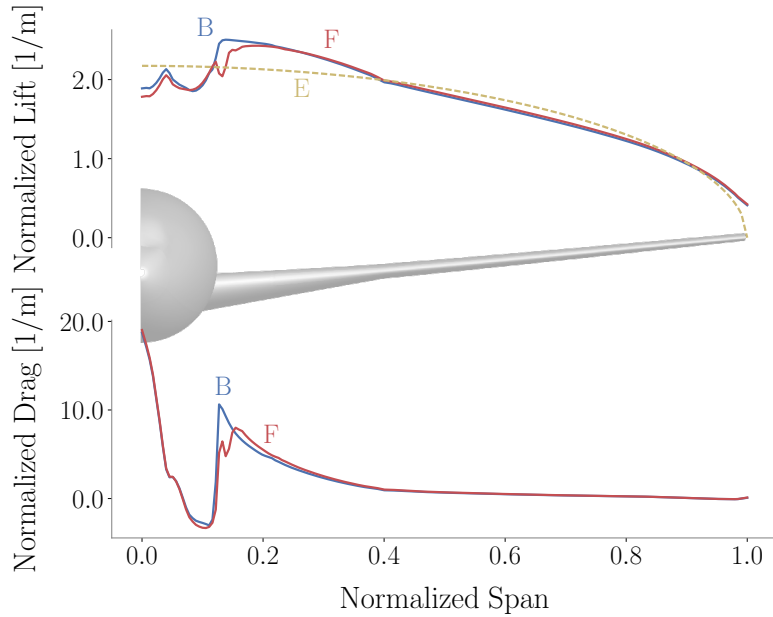


Figure 4.12: Comparison of lift and drag distributions between the baseline (B) and the fairing-optimized configuration (F).

Next, we solve the additional optimization problems listed in Table 4.2 to investigate the effect of adding the other design variable groups. Some relevant performance figures regarding these optimizations, such as optimality and wall time, are listed in Table 4.5. All cases are executed with 192 processors distributed among 4 Intel Xeon Skylake Nodes.

Figure 4.13 shows the relative computational time of the tasks performed on each optimization iteration that includes an adjoint solution. The geometry and mesh manipulation task has a small impact to the overall time of the iteration due to the use of efficient intersection detection, collar mesh generation, and mesh warping algorithms.

No optimization from Table 4.5 reaches the desired reduction in optimality of 1.0×10^{-5} since they terminate due to numerical difficulties caused by the noise in the functions of interest. This follows the same trend observed in the univariate optimization discussed in Sec. 3.8.1. Nevertheless, the optimizer still achieves drag reduction in every case.

The summary of the drag coefficients obtained for each optimized configuration is shown in Table 4.5 and Fig. 4.14. These results indicate that adding design variables tends to reduce the drag. The full optimization (F+T+S) achieves a 16 drag count (5%) improvement compared to the baseline configuration.

Table 4.5: Aerodynamic coefficients obtained for each optimization.

Problem	C_L	C_D (counts)	C_D reduction (%)	Initial optimality	Final optimality	Wall time (minutes)
B	0.4999	290.65	0.0	-	-	1
F	0.5000	285.31	1.8	1.9×10^{-3}	2.4×10^{-5}	32
F+T	0.5000	277.80	4.4	1.6×10^{-2}	3.8×10^{-4}	68
F+T+S	0.4999	274.81	5.4	1.3×10^{-2}	3.1×10^{-3}	96
T	0.5000	279.53	3.8	1.5×10^{-2}	3.6×10^{-5}	71
T+S	0.5000	276.54	4.8	6.9×10^{-3}	8.9×10^{-4}	80

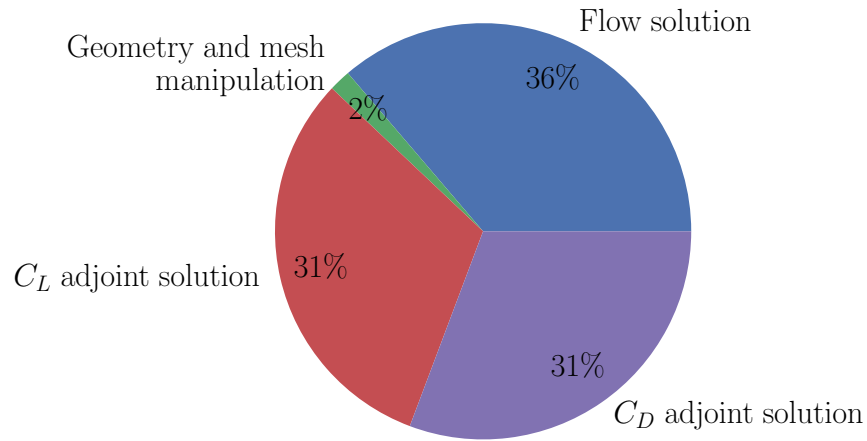


Figure 4.13: Relative computational time of the tasks performed during and optimization iteration involving an adjoint solution. The average time of the iteration is of 85 seconds.

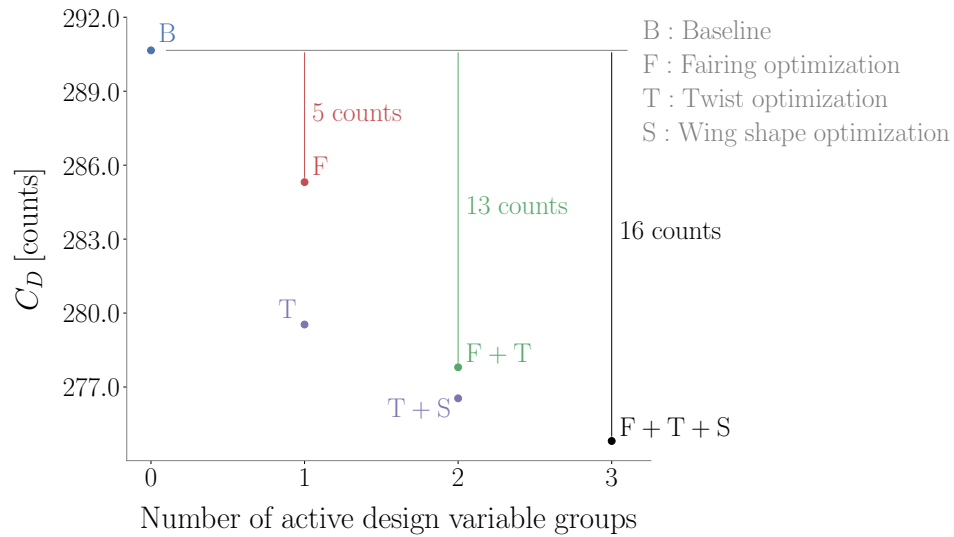


Figure 4.14: Progression of the optimized drag value due to the additional active design variables. Drag decreases as we add more degrees of freedom to the optimization.

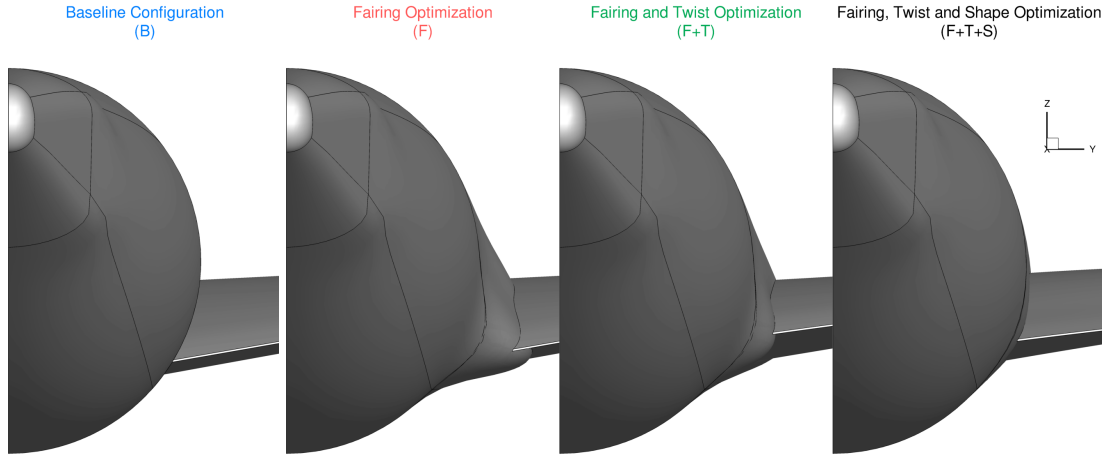


Figure 4.15: Rear views comparing the fairing sizes obtained for different optimizations. The fairing gets smaller as the optimizer gets more control over the wing properties.

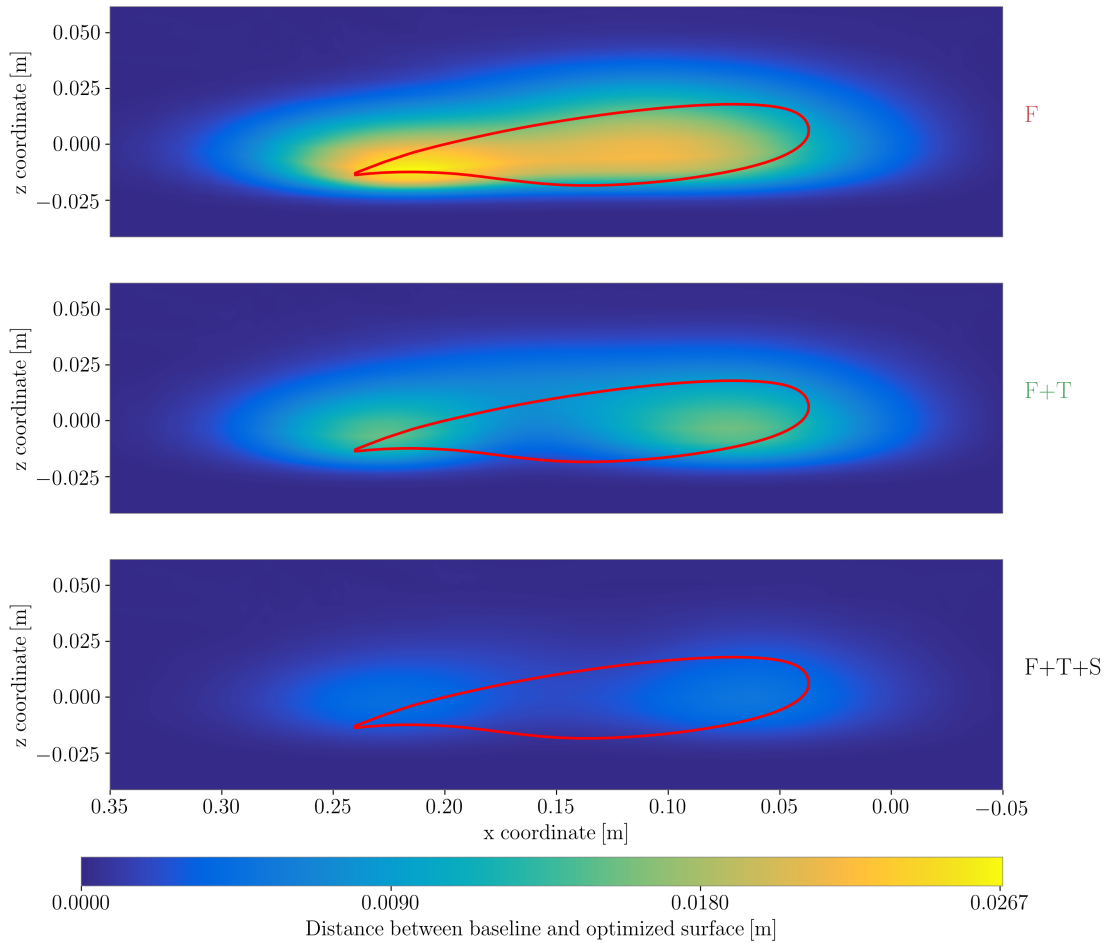


Figure 4.16: Contour plot showing the distance that the fuselage surface moved during optimization. The fuselage surface moves a relatively large amount when the optimizer can only control the fairing.

Figure 4.15 shows how the optimized fairing geometry changes due to the addition of wing twist design variables and wing shape design variables. The fuselage deformation maps of Fig. 4.16 show that the optimizer adds two distinct bumps, one near the leading edge to decrease the suction peak gradient and other near the trailing edge to prevent recirculated flow. The fairing size progressively reduces as we include more degrees of freedom in the optimization, and the fairing design variables do not reach the prescribed bounds anymore, as shown in Fig. 4.17. Since the optimizer has additional degrees of freedom to tailor the wing, the fairing design variables only make local adjustments to the wing-body junction flow pattern, what requires smaller deformations.

We observe similar behavior for twist variables. Figure 4.17 shows that the magnitude of twist variation also decreases when we activate shape design variables.

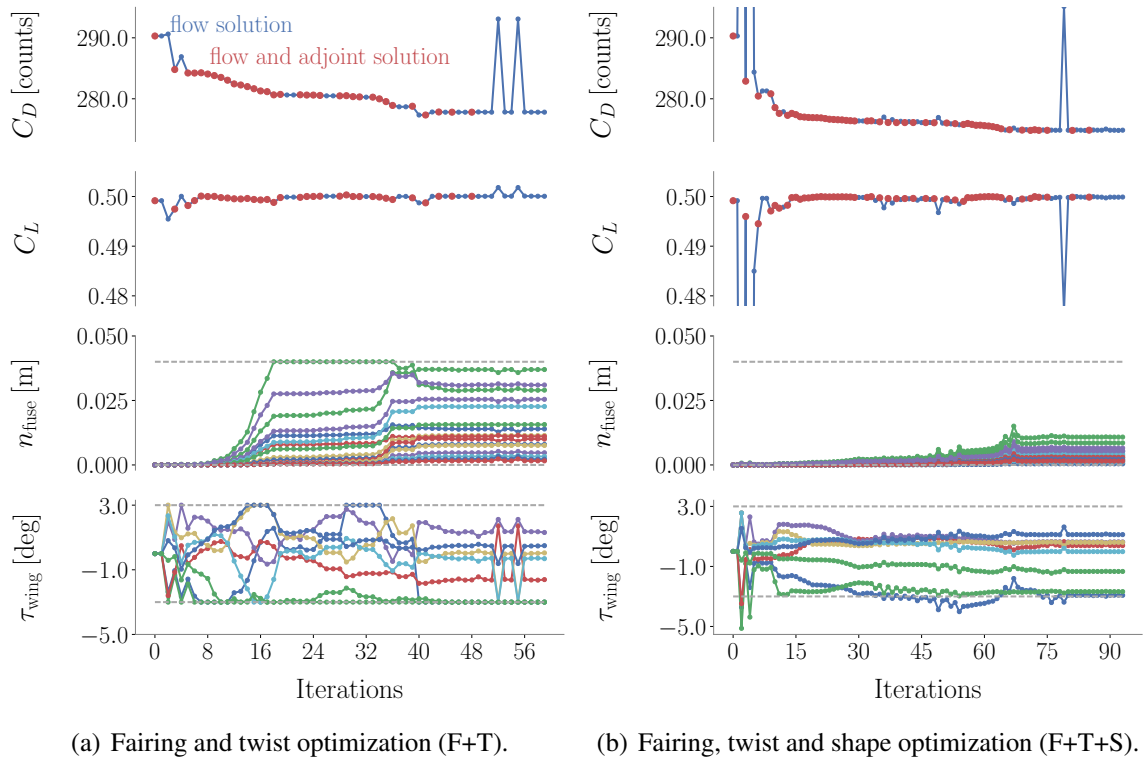


Figure 4.17: Optimization histories for the F+T and F+T+S problems. The fairing design variables do not reach the upper bound in either problem.

Figure 4.18 shows the lift and drag distributions for the different optimized configurations. The inclusion of twist variables allows the optimizer to reach a more elliptical lift distribution to reduce induced drag. In addition, optimizations with twist variables reduce the fuselage contribution to the overall lift and transferred this load to the wing, where lift can be generated more efficiently.

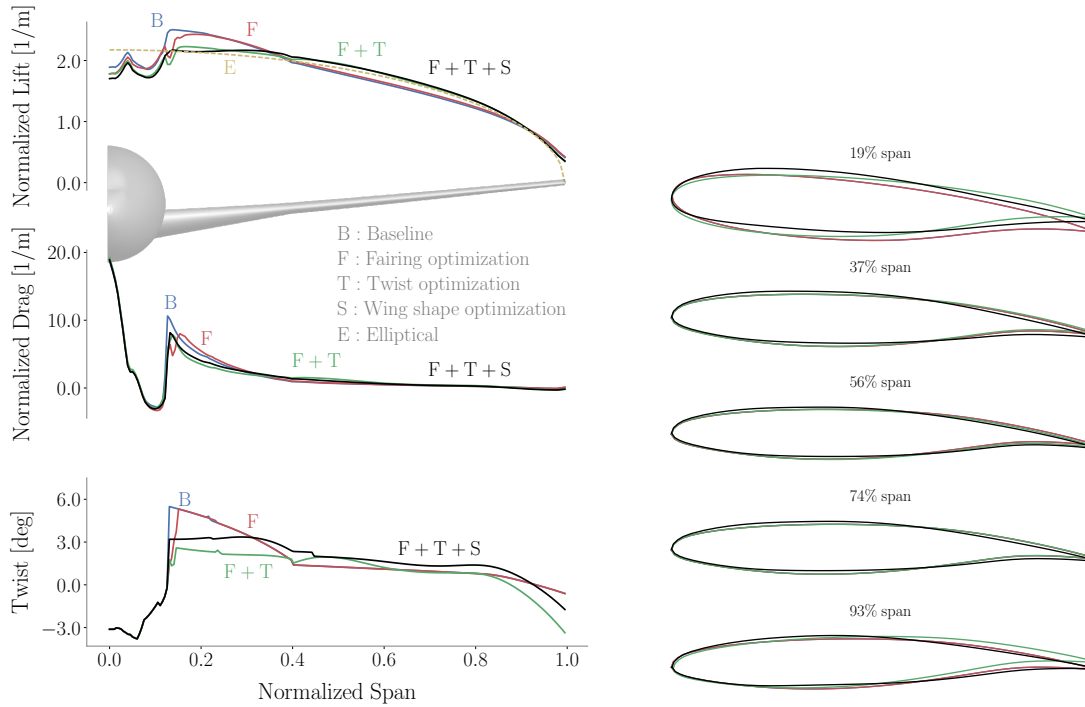


Figure 4.18: Comparison between lift and drag distributions for all fairing optimizations. The inclusion of twist variables (T) allows the optimizer to achieve more efficient lift distributions, which are closer to an elliptical one.

The shape variables control the airfoil camber, which allows the optimizer to reach the desired lift distribution more efficiently than using twist alone. This is because shape variables can further improve the airfoil aerodynamic efficiency (L/D) while still attaining an elliptical lift distribution. The wing cross sections shown in Fig. 4.18 indicate that the airfoils of the optimization with active shape variables (F+T+S) have increased the camber since both the upper and lower surfaces are above their original position.

The introduction of wing shape design variables also removes shocks and results in designs with smoother lift distributions since the optimizer can fine-tune the thickness and camber along the wing (see Fig. 4.19). Smoother pressure distributions have smaller pressure gradients that prevent accelerated boundary layer growth, thus reducing drag.

The fairing design variables effectively remove the separated region of the junction trailing edge in all optimization problems in which they are active, as seen in Fig. 4.20. Wing twist and shape optimization without the fairing design variables might still show minor separated regions (see Fig. 4.21). In fact, the main differences between drag distributions for the T+S and the F+T+S configurations shown in Fig. 4.22 are located around the junction region. Problems T+S and F+T+S converge to practically the same wing ge-

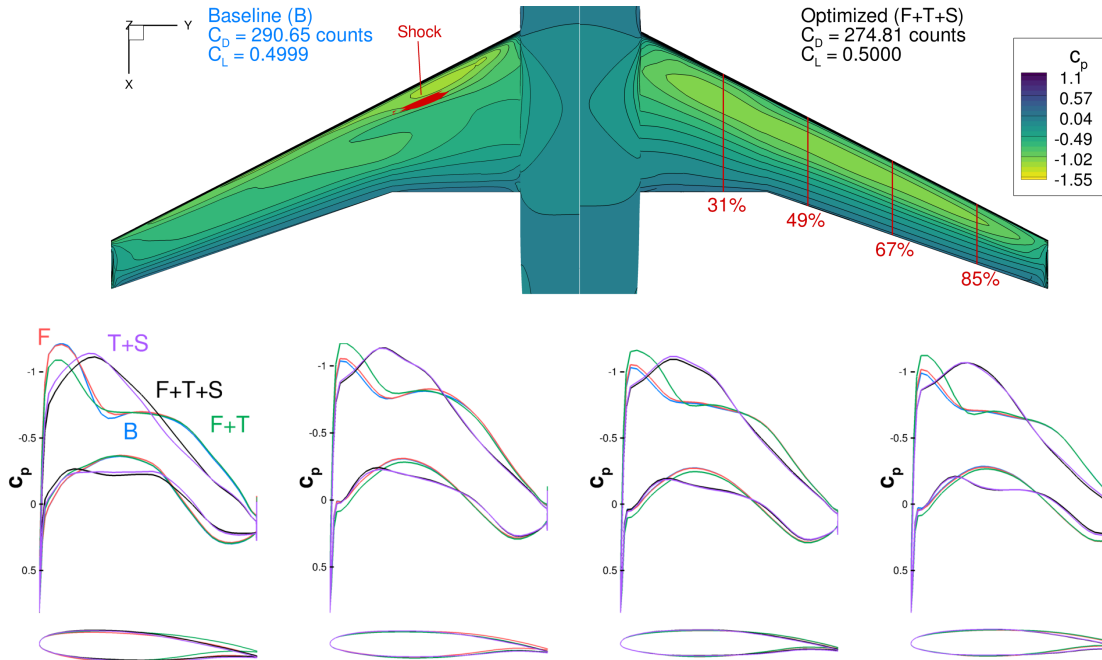


Figure 4.19: Pressure coefficient slices of all optimized configurations. The optimizer designs wings with smoother pressure distributions as we activate wing shape variables (S). The shock on the upper wing surface is also removed by the shape variables. Even though Problems T+S and F+T+S have different design spaces, they show similar airfoils and pressure distributions since they converged to practically the same wing design.

ometry. Figure 4.22 indicates that the wing twist distribution and the airfoil sections of the optimized configurations are similar, except in the vicinities of the wing-body junction, where the effects of the fairing design variables are noticeable. This may be an indication of the convexity of the wing optimization problem, because we get similar wing shapes despite having different design spaces for each problem.

Finally, we show that the drag reduction of the optimized configurations is still present at finer mesh levels. Figure 4.23 plots the mesh convergence study for the optimized configurations and for the baseline configuration. The simultaneous wing and fairing design (case F+T+S) reduces drag by 16 counts (5%) at the coarsest mesh level, while the continuum estimates indicate that the optimized configuration reduces drag by 11 counts (4%). The relative ranking of the configurations in terms of drag remains the same for all mesh levels.

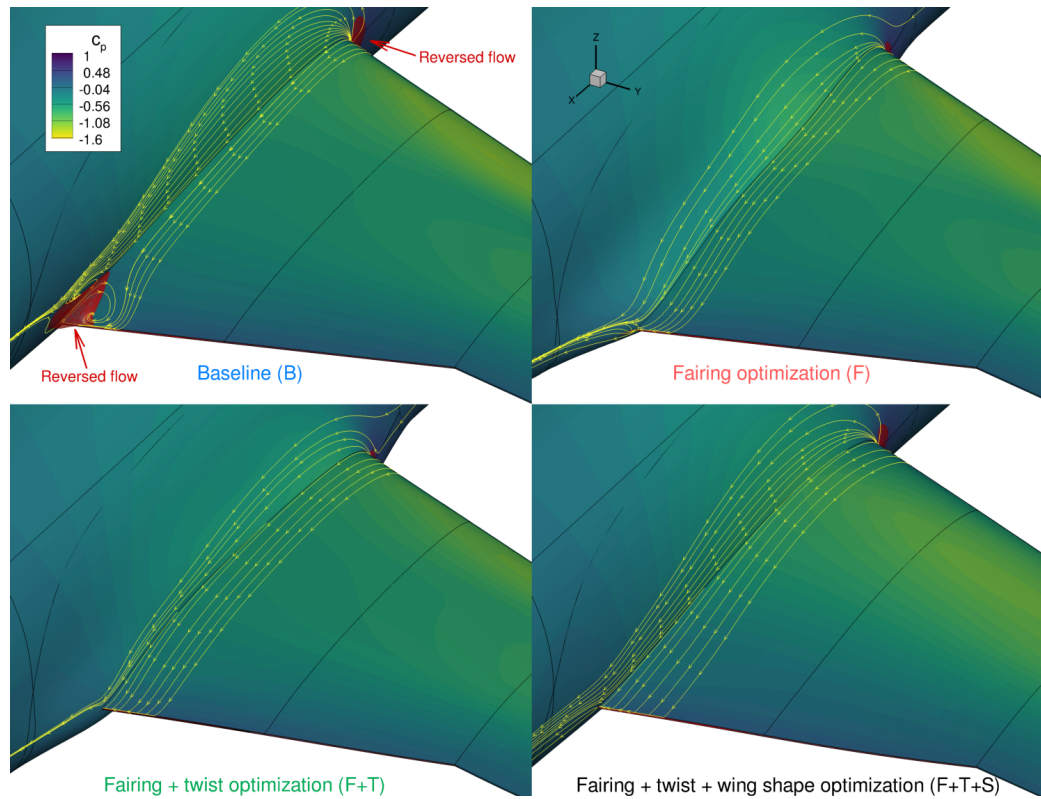


Figure 4.20: Wing-body junction trailing edge after every optimization problem. Red regions indicate reversed flow. The redesigned fairings eliminate the recirculation bubble in all problems. The wing shape variables achieve smoother chordwise pressure distributions.

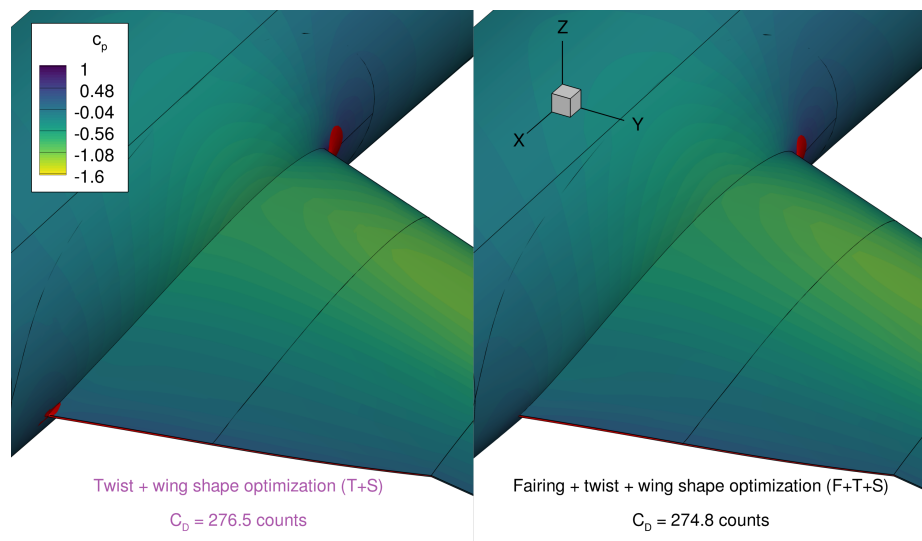


Figure 4.21: Comparison between the optimized configuration with and without fairing design variables. Optimization without the fairing design variables still shows trailing edge separation region (red).

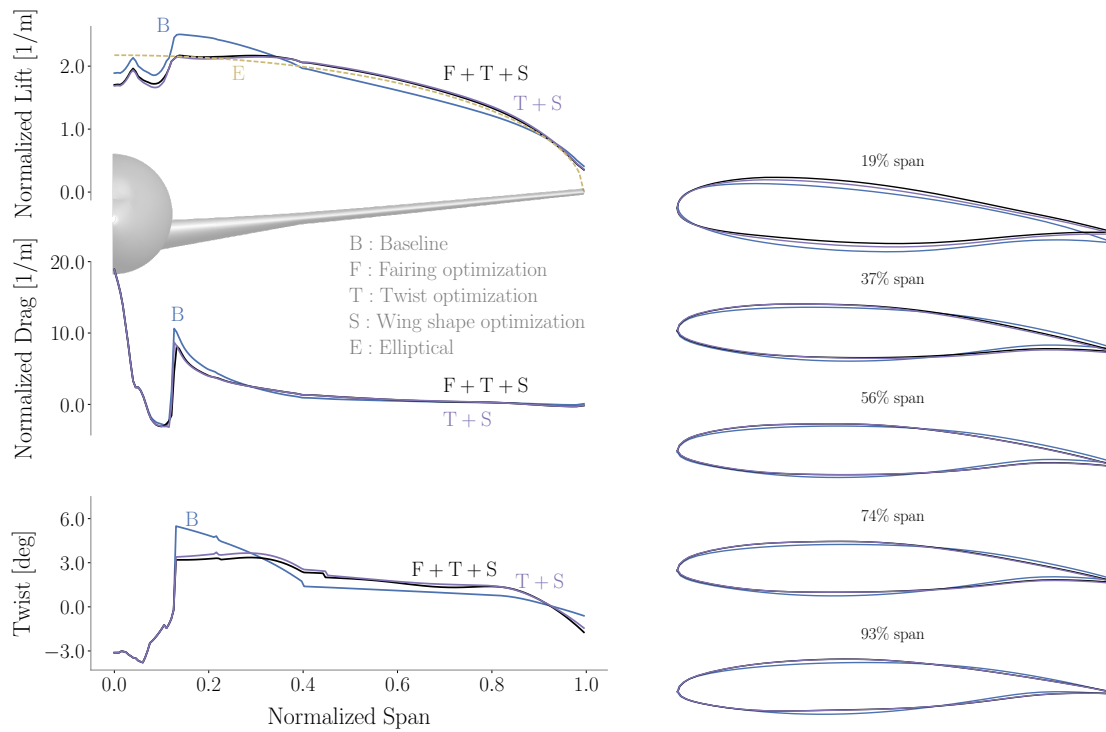


Figure 4.22: Comparison between lift and drag distributions of the twist and shape optimized configurations with (F+T+S) and without (T+S) fairing variables. The resulting twist and airfoil distributions get progressively more similar as we move towards the tip. The F+T+S configuration has a smaller drag in this region as well, which leads to the improvement seen in Fig. 4.14.

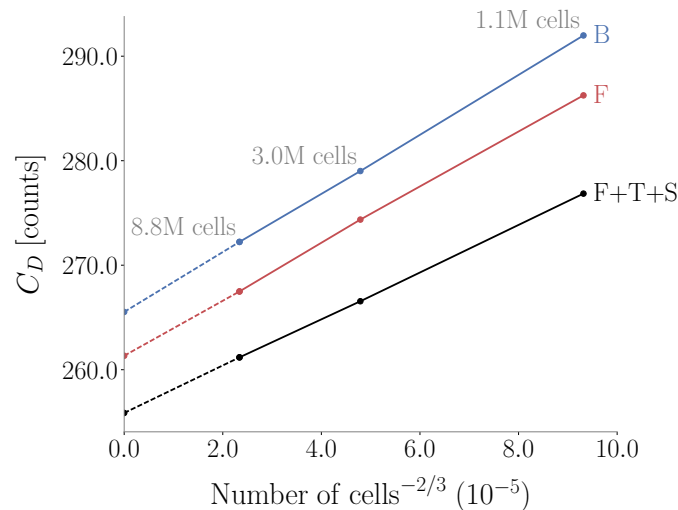


Figure 4.23: Mesh refinement study for baseline (B) and optimized configurations (F and F+T+S). Dashed lines represent continuum estimates. The optimized design improvements are still present at finer mesh levels.

4.5 Summary

In this chapter we used pySurf to perform the aerodynamic shape optimization of the wing-body junction of the DLR-F6 configuration. We demonstrate that independent component manipulation and automatic collar mesh generation can be integrated in gradient-based optimization tasks.

We show that a fairing-alone optimization might try to solve issues from the overall configuration, which is unproductive. Adding wing design variables provides more options for the optimizer to solve these issues, leaving the task of locally adjusting the wing-fuselage junction flow to the fairing design variables, as originally intended. Therefore, we conclude that one should include as many variables as possible to avoid this issue. When all variables are active, the optimizer reduces drag by 16 counts (i.e., 5%) compared with the baseline design.

CHAPTER 5

Strut-braced wing optimization

One approach to decrease aircraft fuel burn is the reduction of induced drag through longer wing spans. However, when aerostructural trade-offs are considered, there is a limit to the span that can be achieved before drag reduction is overwhelmed by structural weight increase [74, 104]. The SBW configuration addresses this issue by adding a strut to alleviate the bending moment in the main wing. The additional strut allows the wing to have a longer span, lowering induced drag [105]. The strut also enables the reduction of the main wing thickness-to-chord ratio, thus decreasing structural weight and wave drag. Despite these benefits, the wing-strut junction increases interference drag; thus, the feasibility of this type of aircraft depends heavily on the synergy between the wing and the strut, as well as on the careful design of the junctions.

The TBW is a variant of the SBW configuration in which the main wing has additional supporting elements besides the strut, called juries [106]. The juries are used to stabilize buckling modes of the wing and strut [107]. However, the additional junctions and the blockage effect of the juries in the wing-strut gap result in even more complex trade-offs between aerodynamics and structures.

The Hurel–Dubois HD-34, built in the 1950’s and primarily used for aerial photography, was one of the first TBW airplanes to be successfully flown [108]. Since then, multiple studies have been conducted to verify the feasibility of this configuration for commercial transport, motivated by the desire to reduce fuel burn. For instance, a study conducted by NASA in 1980 concluded that the SBW configuration would have a lower fuel consumption when compared to conventional configurations, though the loss in productivity due to lower cruise speeds and increase in costs associated with the wing size and complexity could hinder the practicality of this configuration [109].

Despite these concerns, studies of SBWs and TBWs continued as new analysis and design tools became available. Efforts in the late 1990’s and early 2000’s assessed the feasibility of the SBW in a commercial transport mission profile using multidisciplinary tools involving aerodynamics, structures, propulsion, and stability considerations [110, 111]. The

complexity of the multidisciplinary environment often forced the use of medium-fidelity and semi-empirical aerodynamics, such as discrete vortex method for induced drag [112], Korn equation for wave drag, and skin-friction coefficients for parasite drag to obtain a tractable optimization problem. In some efforts, the interference drag associated with junctions was estimated via surrogate models [113, 114] for conceptual design of SBW and TBW configurations [105]. These models were often non-ideal as the computational cost associated with the surrogate model generation could limit the number of design parameters and their bounds.

More recently, NASA and Boeing have conducted analyses of the TBW configuration using both wind tunnel tests and CFD simulations as part of the Subsonic Ultra-Green Aircraft Research (SUGAR) project [107]. One TBW configuration may even be included in the next-generation of NASA X-planes to verify its potential for fuel burn reduction [115].

This increasing interest and investment in these configurations justifies efforts to increase the fidelity of models used in SBW and TBW analyses to understand in more detail the aerodynamic and structural interaction among the aircraft components and to correctly quantify the potential performance gains. For instance, Carrier et al. [108] used a Reynolds-averaged Navier–Stokes (RANS) solver to compute aerodynamic loads on the main wing which were then coupled with a finite-element model to compute structural deformation. Gagnon and Zingg [116, 117] performed Euler-based aerodynamic shape optimization of a SBW considering both the main wing and strut.

In a previous study, we conducted a RANS-based ASO of a TBW in the transonic regime by using shape and twist design variables [11]. Though this study offered insights into the performance of TBW configurations, it was limited in several respects that have since been addressed.

Firstly, it utilized a structured multiblock mesh for computational fluid dynamics (CFD) analyses that prevented changes in the wing-strut junction geometry due to complications in deforming the mesh near the concave regions of the junctions without generating negative volume cells. We have since extended our CFD solver to handle overset meshes [44], facilitating the generation and manipulation of meshes for this type of configuration.

Secondly, our previous TBW optimization was limited by characteristics of our geometry manipulation module. The previous geometry manipulation method operated on the entire geometry, rather than on individual components. Therefore, it was not possible to implement independent sets of design variables for each component. For instance, if we changed the spanwise wing-strut intersection position, the lower surface of the wing would be moved as well, causing unwanted compression and extension of surface cells on the wing CFD mesh. We recently developed a component-based geometry manipulation mod-

ule that allows the strut to freely move with respect to the wing, avoiding the prior geometry limitations [118].

In this chapter we perform RANS-based aerodynamic shape optimization of a SBW in the transonic regime using overset meshes and component-based geometry manipulation to allow wing-strut junction shape design. We use the baseline SBW geometry and flow conditions from the Platform for Aircraft Drag Reduction Innovation (PADRI) 2017 workshop [119]. This geometry, shown in Fig. 5.1, consists of three primary components: fuselage, wing, and strut. The fuselage geometry and the wing-fuselage junction is kept unchanged in this work. Table 5.1 shows the main dimensions of each component and the flow condition used for aerodynamic analysis.

The RANS analyses provide a more accurate quantification of the drag reductions that are possible for the SBW, thus improving upon Euler-based studies [116, 117, 120] and complementing previous low-fidelity multidisciplinary design optimization estimates [105, 106, 110, 111, 121].

We address two design optimization cases in this work. First, we optimize the shape of the entire wing and strut. Then, we just optimize the vicinities of the wing-strut junction, following the guidelines established by the PADRI 2017 workshop. This allows us to compare the drag reduction of an ad-hoc fix to the junction against the drag reduction achieved by the entire redesign of the wing and strut.

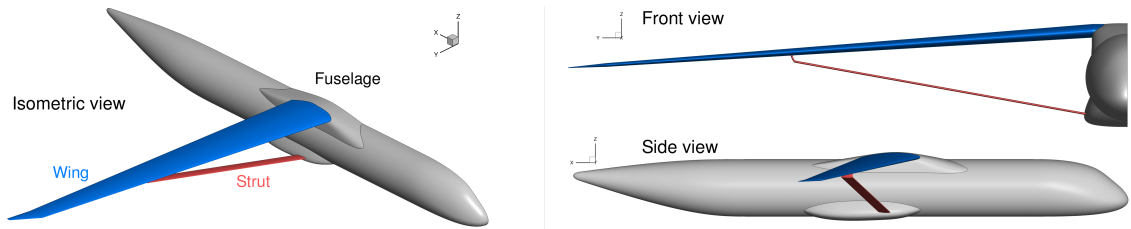


Figure 5.1: Baseline SBW configuration of the PADRI 2017 workshop. Views are not in the same scale.

5.1 Optimization problem definition

The optimization objective in this work is to minimize the drag of the SBW for a given Mach number, lift coefficient, and wing planform. In other words, the span and chord of the wing and strut remain fixed, while the optimizer can change airfoil shapes and spanwise twist distributions. For the first design optimization, we allow changes over the whole wing and strut, including the wing-strut junction region. The second design optimization follows

Table 5.1: Geometric characteristics and flight conditions used for the baseline SBW configuration analysis.

Wing		Strut		Fuselage	
Planform area [m ²]	161	Planform area [m ²]	28.9	Length [m]	41.7
Span [m]	55.6	Span [m]	33.4	Diameter [m]	4.3
Mean aerodynamic chord [m]	3.264	Mean aerodynamic chord [m]	0.865		
Aspect ratio	19.2	Aspect ratio	38.6		
Taper ratio	0.256	Taper ratio	1.00		
1/4 Chord sweep [deg]	15.0	1/4 Chord sweep [deg]	10.9		
Dihedral [deg]	-4.0	Dihedral [deg]	10.2		

Flight Condition	
Mach	0.72
α [deg]	1.0
Altitude [ft]	30,000

the PADRI 2017 guidelines, which only allows changes nearby the wing-strut junction. This second case is detailed in Sec. 5.3.

We start with the baseline IGES description of the SBW provided in the PADRI 2017 website [119] to create structured surface meshes (Fig. 5.2) and triangulated surface meshes (Fig. 5.3) for each primary component using ICEMCFD. We then embed both surface meshes of each component into their corresponding FFD boxes (Fig. 5.4) to allow pyGeo to manipulate their shape. The fuselage has no associated FFD since its geometry remains fixed for this optimization. Regardless, we still require the fuselage triangulated surface for collar mesh generation. The shape variables consist of hundreds of control points distributed over the FFD boxes. This leads to the general formulation of the optimization problem in Table 5.2.

The wing FFD is comprised of two layers of 16 chordwise by 20 spanwise control points, with one layer for the upper surface and one for the lower. These control points move vertically to change the underlying wing surface shape. Control points within the same chordwise station can also be simultaneously rotated around the quarter-chord to change the local twist. Wing twist is controlled in 10 spanwise locations along the FFD and the intermediate twist values are interpolated between these locations. To avoid shear twist, the pairs of control points on the leading and trailing edges must move equal distances in opposing directions. This requirement is enforced by the leading and trailing edge constraints listed in Table 5.2.

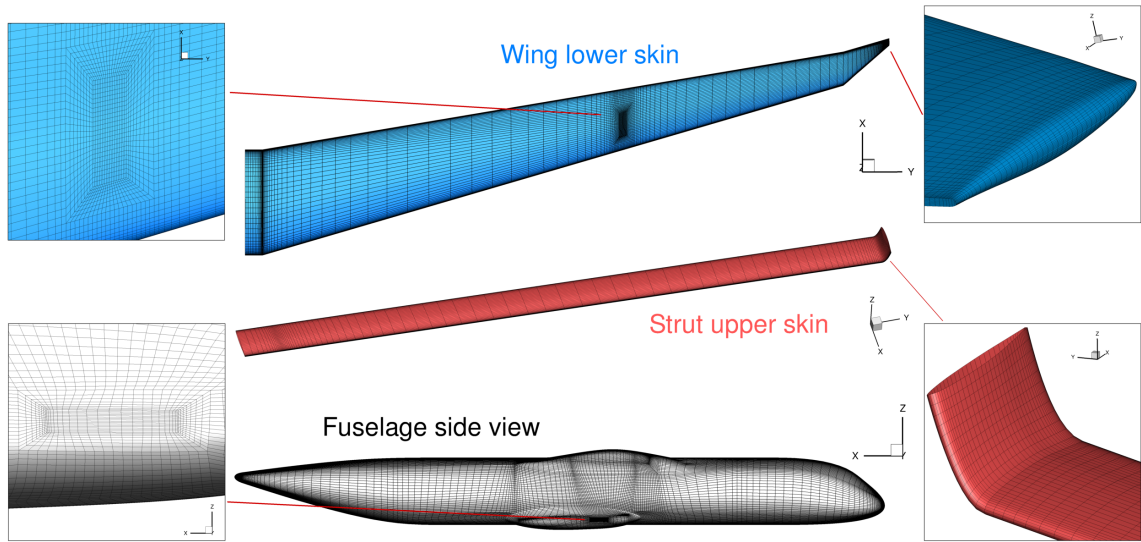


Figure 5.2: Structured surfaces meshes of the primary components of the aircraft. The O-grids near intersections increase the cell density to facilitate the overset hole cutting process.

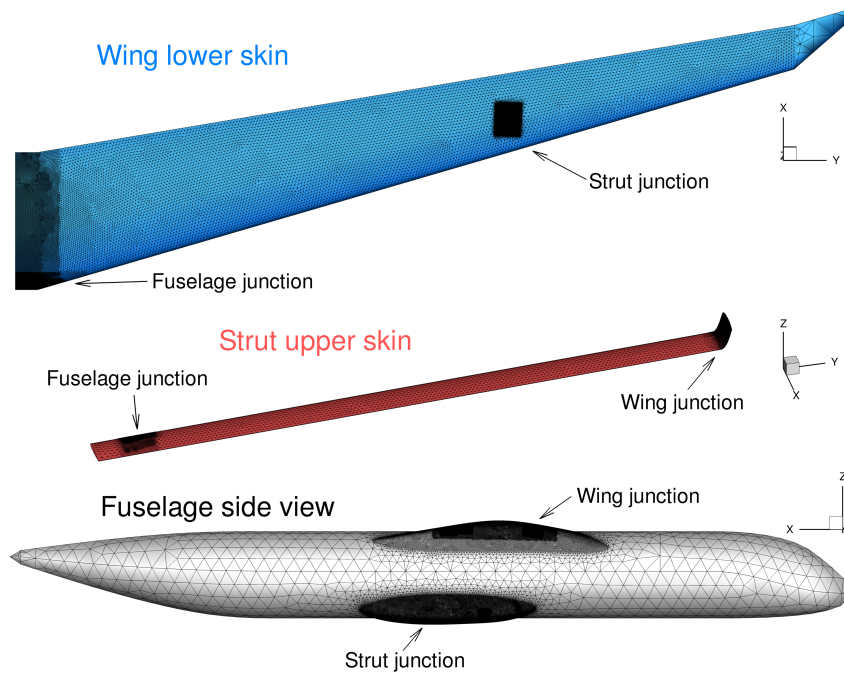


Figure 5.3: Triangulated surfaces used for intersection detection and collar mesh generation.

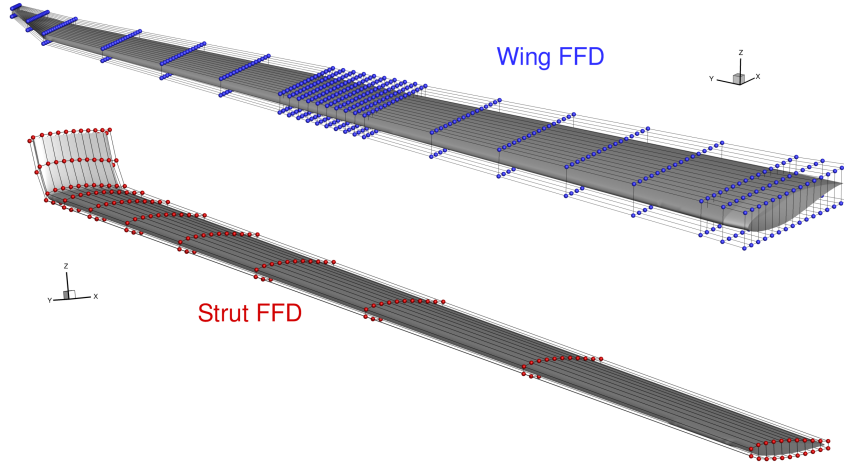


Figure 5.4: FFD boxes of the primary components whose shape will be optimized. The dots represent the FFD box control points.

Table 5.2: SBW aerodynamic shape optimization problem.

	Variable/function	Description	Quantity
minimize	C_D	Drag coefficient	
with respect to	$-1.0 \leq \alpha \leq 3.0$	Angle of attack [deg]	1
	$-5.0 \leq \tau_{\text{wing}} \leq 5.0$	Wing sections twist [deg]	10
	$-0.05 \leq \Delta z_{\text{wing}} \leq 0.05$	Vertical displacement of wing FFD control points [m]	640
	$-10.0 \leq \tau_{\text{strut}} \leq 5.0$	Strut sections twist [deg]	11
	$-0.06 \leq \Delta n_{\text{strut}} \leq 0.06$	Normal displacement of strut FFD control points [m]	242
		Total design variables	904
subject to	$C_L = 0.4058$	Lift constraint (baseline at $\alpha = 1.0$ deg)	1
	$t/t_{\text{init,wing}} \geq 0.999$	Wing thickness constraint	60
	$\Delta z_{\text{wing,TE,upper}} = -\Delta z_{\text{wing,TE,lower}}$	Wing fixed trailing edge constraint	20
	$\Delta z_{\text{wing,LE,upper}} = -\Delta z_{\text{wing,LE,lower}}$	Wing fixed leading edge constraint	20
	$t/t_{\text{init,strut}} \geq 0.999$	Strut thickness constraint	84
	$\Delta n_{\text{strut,TE,upper}} = -\Delta n_{\text{strut,TE,lower}}$	Strut fixed trailing edge constraint	11
	$\Delta n_{\text{strut,LE,upper}} = -\Delta n_{\text{strut,LE,lower}}$	Strut fixed leading edge constraint	11
		Total constraints	213

The strut has control points defined for both the upper and lower surfaces, similar to the wing. The control points of each surface are distributed in a mesh of 11 chordwise by 11 spanwise control points. Each control point can be independently moved in the direction normal to the strut surface to change the thickness and camber distribution along the strut. In addition, 11 twist variables are defined for the strut sections (one for each chordwise section). We also enforce constraints on the leading and trailing edge control points to

avoid shear twist of the strut sections. The aircraft angle of attack is an additional design variable to allow the optimizer to match the lift constraint.

Since structural analyses are not conducted during optimization, it is necessary to enforce artificial constraints to take structural considerations into account and to avoid unrealistically thin airfoil designs. For this optimization, we compute thicknesses in 144 locations along the wing and the strut, as shown in Fig. 5.5. The thicknesses at these points cannot go below their corresponding initial value.

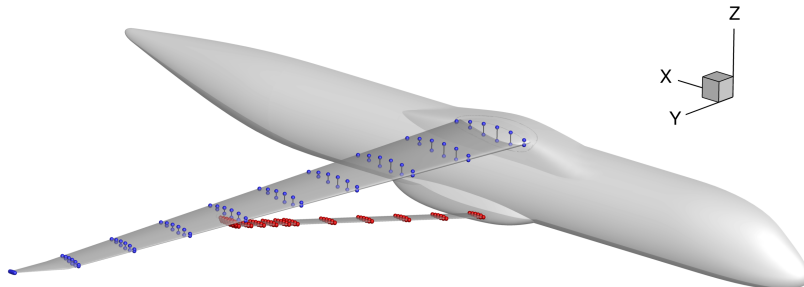


Figure 5.5: Points of the wing (blue) and strut (red) where thickness constraints are enforced. The distances between the pair of points cannot go below their initial value.

The C_L constraint ensures that the optimized configuration generates the same lift as the baseline configuration at 1.0 degree of angle of attack. Preliminary aerodynamic analysis of the baseline configuration with ADflow shows that the lift coefficient value for this condition is $C_L = 0.4058$.

5.2 Wing and strut optimization

In this first optimization problem, we modify the entire extent of the strut and wing. The optimization takes 27 hours using 1088 threads distributed among 16 Intel Xeon Phi 7250 KNL nodes. The optimizer requires 69 CFD simulations and 40 adjoint solutions to arrive at the final result. The optimization history (Fig. 5.6) shows that the drag coefficient is reduced by 14.7%, while maintaining the lift coefficient of the initial configuration. The assumption of frozen interpolation weights in the reverse AD code does not compromise the optimization since the optimizer takes several unit steps, as indicated by the consecutive flow and adjoint evaluations in Fig. 5.6.

The optimality conditions are not satisfied to the specified tolerance, since the optimizer stops prematurely due to inconsistencies in the gradient caused by noise in the functions of interest. This effect only becomes significant near the optimum, where variations caused by the noise exceed the shallow slopes of the underlying function. Nevertheless, the design

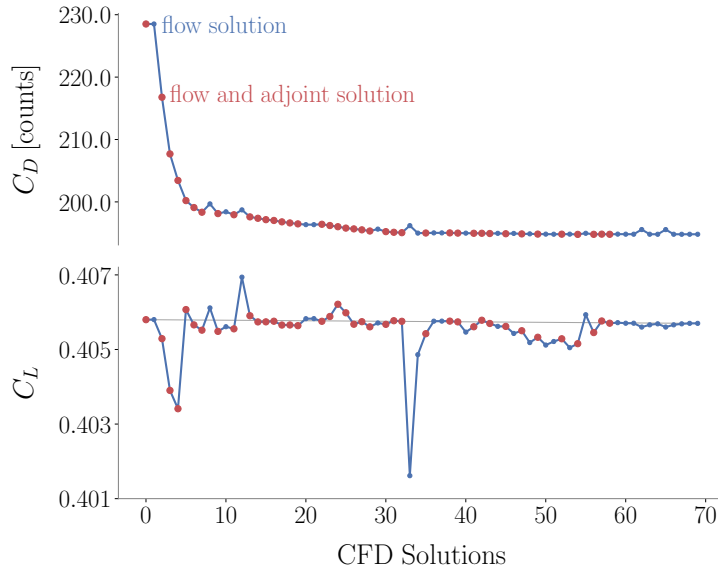


Figure 5.6: Optimization history of the SBW optimization. The drag decreases by 33 counts for the same lift coefficient.

already has significantly improved at this point of the optimization. Furthermore, during the final iterations we only see minor changes to the geometry and the drag value decreases by thousandths of drag counts between iterations. Therefore, the shape can be considered optimal for practical purposes.

The optimizer reduces the drag by improving the spanwise lift distribution, eliminating shock waves, and reducing the amount of separated flow. These correspond in turn to a reduction in induced drag, wave drag, and viscous drag, respectively.

The baseline configuration shows a shock wave near the wing-strut intersection region that is eliminated in the optimized design (see Fig. 5.7). The optimized shape reduces the flow acceleration in the gap between the two components by decreasing the strut incidence angle. Consequently, there is a pressure increase over the facing surfaces of the wing and the strut when compared to the baseline configuration (also shown in Fig. 5.7). The optimized solution also shows smoother pressure distributions on the lower surface of the wing, which slows boundary layer growth and thus decreases viscous drag.

The elimination of shocks has a positive impact on separation, as a large portion of separation observed in the baseline design is induced by shocks (Fig. 5.8). The wing-strut junction also induces separation on its trailing edge, and the optimizer manages to adjust the geometry of the intersection to reattach the flow in this region.

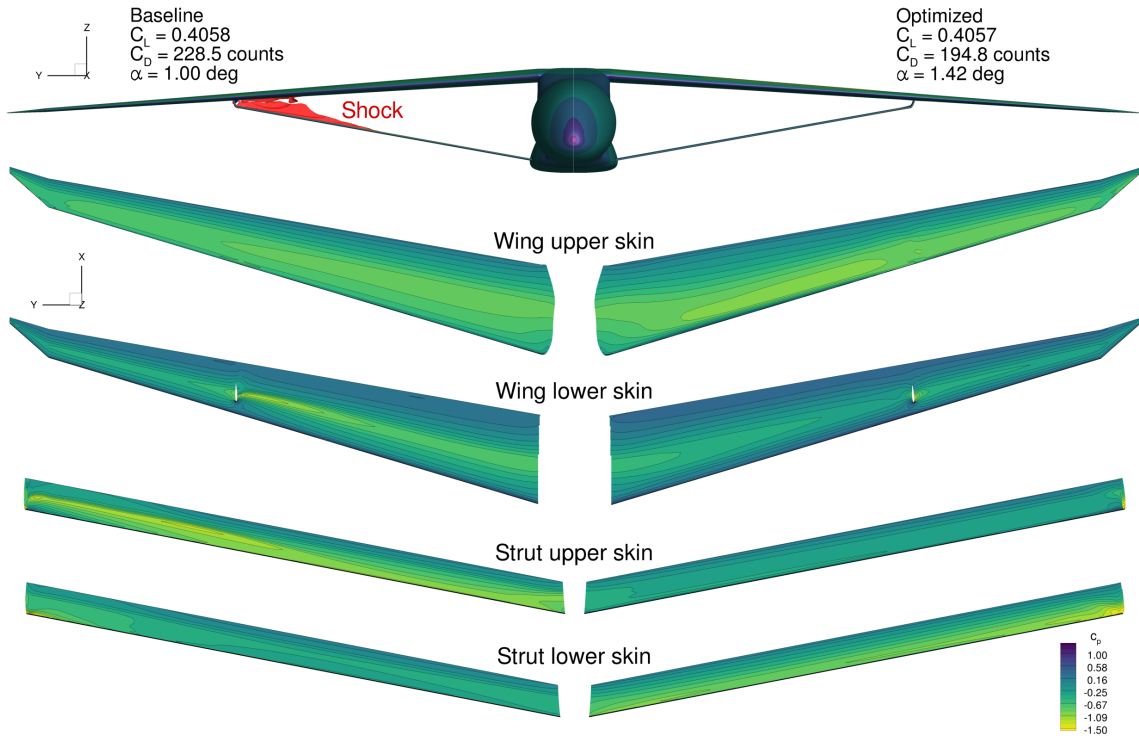


Figure 5.7: Comparison of shock and pressure distributions between the baseline (left) and optimized (right) designs. The strut is shown at a different scale.

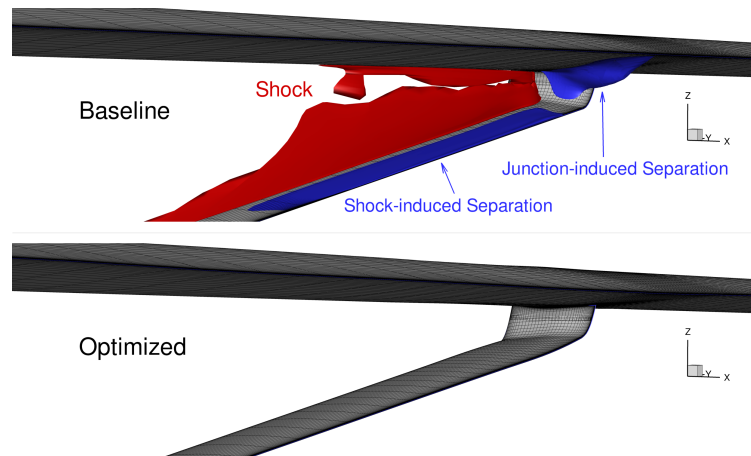


Figure 5.8: Separation on the trailing edge of the wing-strut junction before (top) and after (bottom) the optimization.

The incidence angle of the optimized strut is reduced to the point of generating negative lift, as shown in Fig. 5.9 and Fig. 5.10. This negative lift generation is compensated for by increased twist in the midspan part of the wing, resulting in an overall lift distribution that is close to elliptical. These trends were also observed in Euler- and RANS-based aerodynamic

shape optimizations of different SBW geometries [11, 116, 117, 120]. Parametric studies done by Ko et al. [122] also show that using a strut with negative incidence angles and a relatively flat upper surface reduces the shock strength near the wing-strut intersection. Most of the drag reduction occurs from the inboard region of the wing and also from the strut sections near the junction, as shown in the upper right plot in Fig. 5.9.

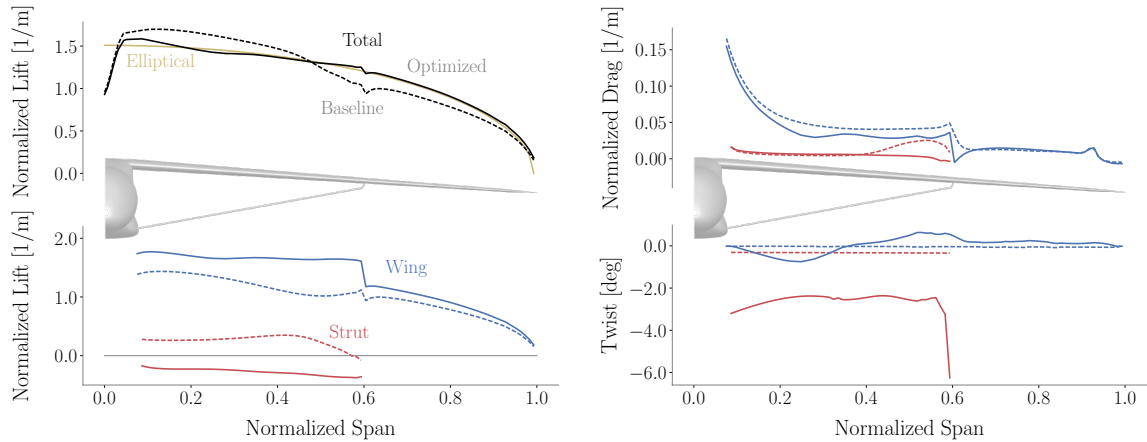


Figure 5.9: Spanwise distribution of lift, drag and twist for the baseline (dashed lines) and optimized configuration (solid lines). The strut generates downward lift, and the inboard region of the wing increases its lift distribution to compensate for that, yielding an overall elliptical lift distribution.

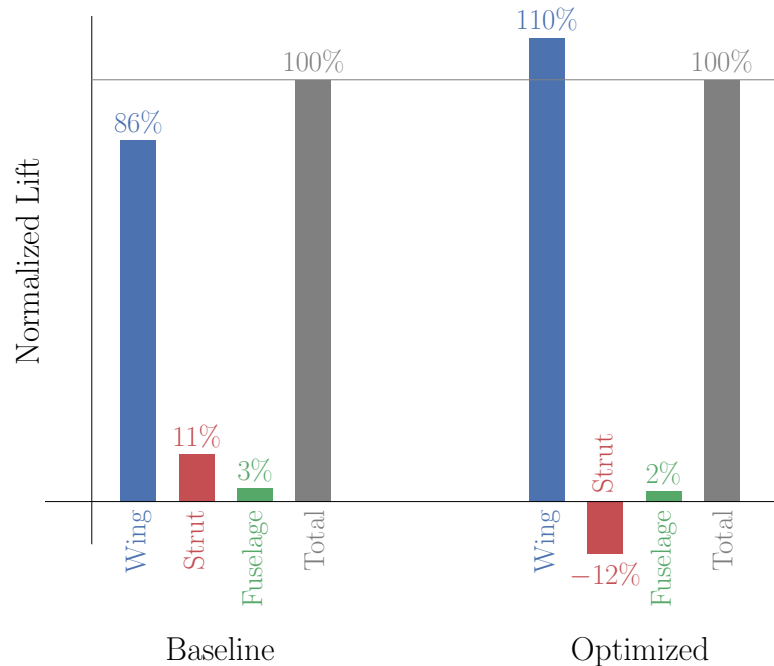


Figure 5.10: Component-wise lift distribution. The lift is normalized by the C_L constraint.

The optimization lowered the magnitude of the suction peaks on the lower surface of the wing and on the upper surface of the strut (Fig. 5.11). The wing slice at 59% of the semi-span of the optimized configuration has a subtle bump on the lower surface at approximately three quarters of the chord. The strut slice at 59% of the semi-span shows a bump near the trailing edge of the upper surface to increase the extent of favorable pressure gradient in this area, thus delaying separation.

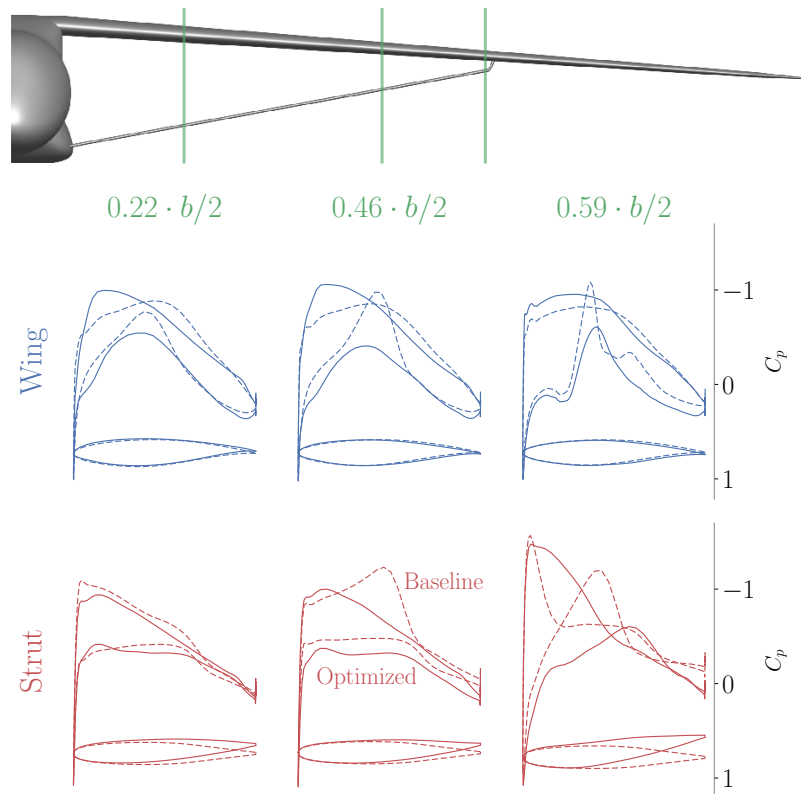


Figure 5.11: Cross-sectional slices of the wing and strut and corresponding pressure distributions for the baseline (dashed lines) and optimized (solid lines) configurations.

This trailing edge bump also extends to the vertical section of the strut (Fig. 5.12). The redesigned shape eliminated the flat pressure distribution regions on the trailing edge caused by separation. The suction side of this vertical segment of the strut is on the outboard surface, following the same trend as the rest of the strut, to avoid high-speed flow on the inner side of the junction. According to the cross-sectional slices shown in Fig. 5.11 and Fig. 5.12, the optimization makes more significant changes to the strut rather than the wing in the quest to reduce drag.

This study marks a notable improvement to our previous TBW investigation [11]. The previous optimization using multiblock meshes could not remove junction shocks and sep-

aration as effectively as the current approach based on overset meshes and component-based parametrization. In addition, the current work avoids the generation of negative volume cells that hindered the optimization in the original process. While the previous work seemed to obtain a greater relative drag reduction, the results are not quantitatively comparable to this study as the previous optimization referenced a different baseline configuration, a different flight condition, and more permissive thickness constraints.

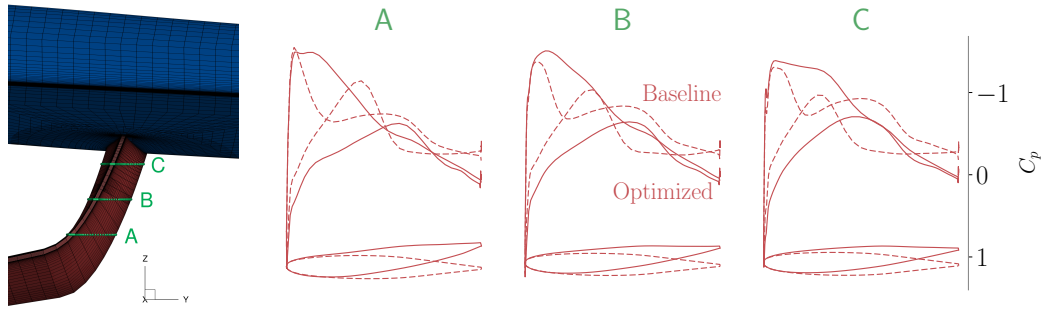


Figure 5.12: Cross-sectional slices of the vertical segment of the strut for the baseline (dashed lines) and optimized (solid lines) configurations.

5.3 Junction optimization according to PADRI 2017 guidelines

We also analyze a second optimization problem following the guidelines of the PADRI 2017 workshop, which establishes that the aircraft shape can only be modified between 52% and 63% of the semi-span. Furthermore, the twist and the upper skin of the wing cannot be changed at any place.

We achieve this requirement by fixing control points outside of the specified spanwise range. The first layer of control points inside the optimization range is also constrained due to the support of the B-splines used in the FFD mapping. The control points on the upper surface of the wing are also fixed, while the lower surface control points can only move downward to avoid reduction of the wing thickness. We also constrain the leading and trailing edge control points on the lower surface of the wing to avoid shear twist.

The bounds of the wing FFD control points already prevent thickness reduction and shear twist for this component. Therefore, we only need to keep the thickness constraints and shear twist constraints for the strut. In the end we have 233 design variables and 63 constraints, as shown in Tab. 5.3.

Table 5.3: SBW aerodynamic shape optimization problem after PADRI 2017 guidelines.

	Variable/function	Description	Quantity
minimize	C_D	Drag coefficient	
with respect to	$-1.0 \leq \alpha \leq 3.0$	Angle of attack [deg]	1
	$-0.05 \leq \Delta z_{\text{wing}} \leq 0.00$	Vertical displacement of wing FFD control points [m]	72
	$-10.0 \leq \tau_{\text{strut}} \leq 5.0$	Strut sections twist [deg]	6
	$-0.06 \leq \Delta n_{\text{strut}} \leq 0.06$	Normal displacement of strut FFD control points [m]	154
		Total design variables	233
subject to	$C_L = 0.4058$	Lift constraint (baseline at $\alpha = 1.0$ deg)	1
	$t/t_{\text{init, strut}} \geq 0.999$	Strut thickness constraint	48
	$\Delta n_{\text{strut, TE, upper}} = -\Delta n_{\text{strut, TE, lower}}$	Strut fixed trailing edge constraint	7
	$\Delta n_{\text{strut, LE, upper}} = -\Delta n_{\text{strut, LE, lower}}$	Strut fixed leading edge constraint	7
		Total constraints	63

This optimization takes 31 hours on 1088 threads distributed among 16 Intel Xeon Phi 7250 KNL nodes. It requires 91 CFD simulations and 57 adjoint solutions to arrive at the final result. The optimized configuration drag coefficient is 6.4% smaller than the baseline, as shown in the optimization history (Fig. 5.13). Figure 5.14 shows that the deformations only occur within the spanwise range defined by the PADRI 2017 guidelines. The constraints imposed by the PADRI 2017 guidelines prevents the optimizer from reaching a shock-free configuration. Figures 5.15 and 5.16 show that shocks and separation remain outside the optimized region.

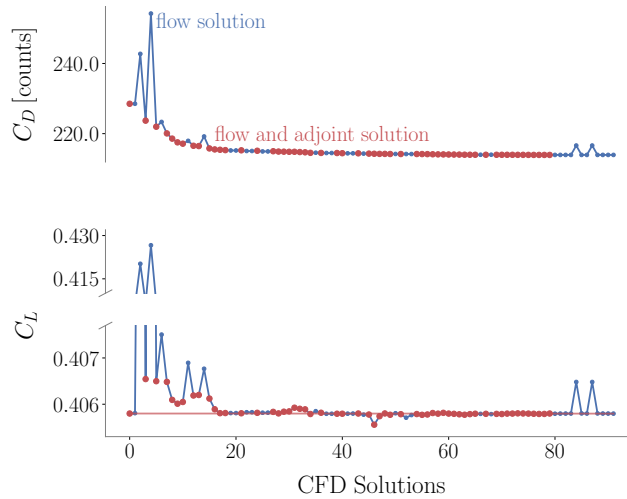


Figure 5.13: Optimization history of the SBW optimization for the PADRI 2017 guidelines. The drag decreases by 14 counts while maintaining the same lift coefficient of the baseline configuration (red line).

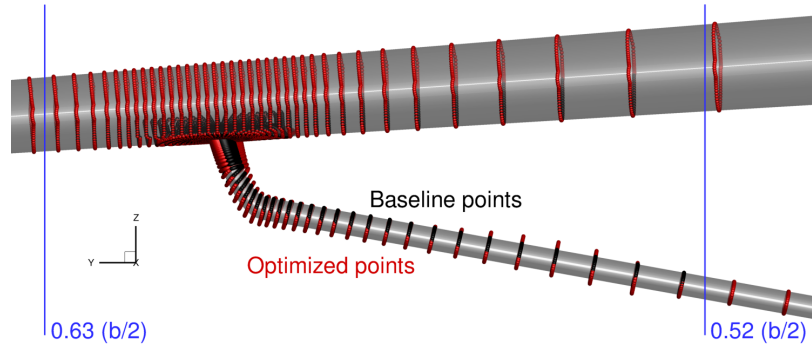


Figure 5.14: Discrepancies between the CFD surface nodes of the baseline (black) and optimized (red) configurations only occur on the lower surface of the wing and on the strut surface within the specified spanwise range.

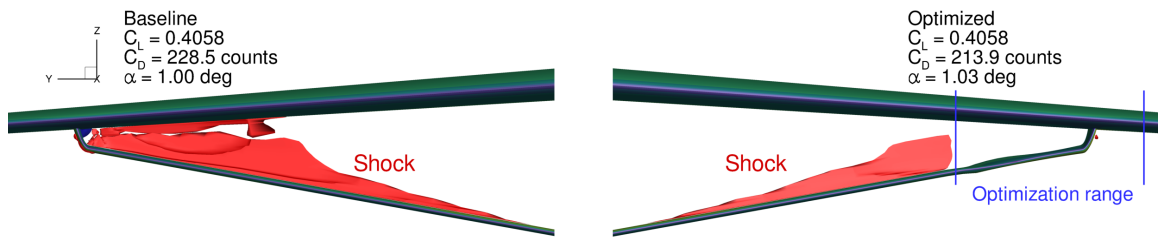


Figure 5.15: Comparison of shock waves between the baseline (left) and optimized (right) designs for the PADRI optimization case. Shocks still remain outside of the optimized region.

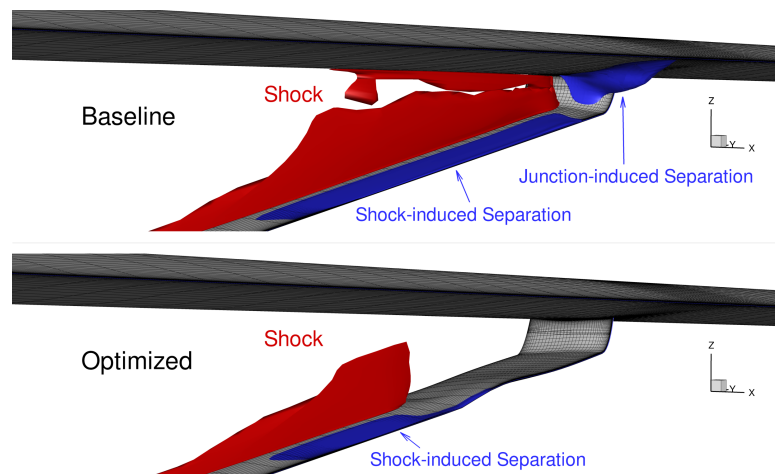


Figure 5.16: Rear view of the wing-strut intersection showing separation regions (blue). The optimizer manages to remove the trailing edge separation only within the spanwise range where the design variables are active.

Figure 5.17 shows cross-sectional slices of the previous optimized configuration (shown in Sec. 5.2) and the one obtained after imposing the PADRI 2017 constraints. Even though both optimized designs follow the same trends, the magnitude of these changes are more expressive in the latter due to the additional constraints. For instance, the optimized strut of the PADRI 2017 case needs additional twist to reduce the flow velocity in the wing-strut gap, since the wing twist is fixed. The fixed wing twist also forces the optimizer to increase the bump size on the lower surface of the wing near the strut intersection for a smoother pressure recovery, as shown in the wing slice at 59% of the semi-span (upper right chart of Fig. 5.17).

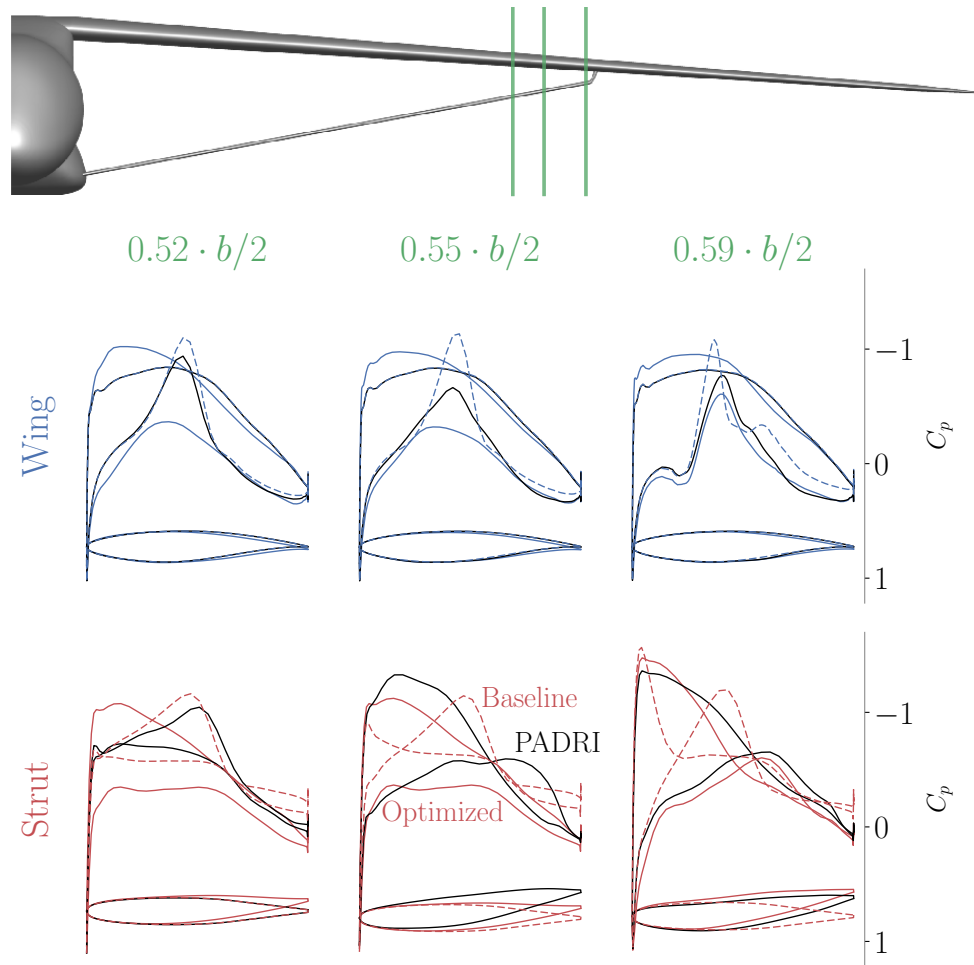


Figure 5.17: Comparison among the cross-sectional slices of the wing and strut and corresponding pressure distributions for the baseline configuration (red dashed lines), the fully optimized configuration (red solid lines), and the optimized configuration for the PADRI guidelines (black). The optimized shape for the PADRI 2017 case has more twist to compensate the fixed twist angles of the wing and the remainder of the strut.

5.4 Summary

In this chapter we demonstrate the feasibility of RANS-based aerodynamic shape optimization of a transonic SBW geometry using overset meshes and component-based geometry manipulation. These methodologies avoid the mesh deformation issues seen in previous work and gave better shape control of the SBW components and their junctions.

The optimized design achieves a 14.7% drag reduction compared to the baseline configuration for the same lift. Studies of the optimized configuration show that a downward-lifting strut avoids suction peaks in surfaces facing the wing-strut gap, eliminating the shock wave seen in the baseline configuration. The inboard section of the wing has to increase its load to compensate the negative lift of the strut and achieve an elliptical overall lift distribution. Adjustments to the wing-strut intersection curve and the addition of a bump on the strut trailing edge change the pressure recovery and reduce the separation. The optimization also demonstrates that changes to the strut shape are more effective for drag reduction rather than modifications to the wing shape.

The wing-strut junction optimization following the PADRI 2017 guidelines reduces the drag coefficient by 6.4%. Changes on the strut shape follow the same trends of the previous optimization, in which the entire wing and strut are redesigned. However, the magnitude of these changes become more significant to compensate the smaller number of design variables. This highlights the fact that better performance margins are achieved when issues are addressed earlier on during the design processes, when the designer has more freedom to modify the overall shape of the aircraft. The use of high-fidelity analysis and optimization during the conceptual design facilitates the detection and solution of these potential issues.

CHAPTER 6

Concluding remarks

In this chapter the main conclusions of the work presented in this thesis are summarized. The main contributions are also highlighted and directions for future studies are suggested.

6.1 Conclusions

The aerospace research community is actively pursuing new geometric and mesh parametrization methods to add more flexibility to CFD-based optimization frameworks. Chapter 1 described that one of the main challenges of the current geometry manipulation modules is the need to track changes in intersections among different components during the optimization, while allowing the computation of derivatives for gradient-based optimization. This prevents the use of Euler- and RANS-based optimization frameworks to design configurations with multiple components and junctions, such as the SBW.

To address this need, a component-based parametrization technique that uses triangulated surfaces to compute intersections and generate collar meshes is introduced, as shown in Chapter 2. These triangulated surfaces are embedded in the same geometry manipulation module (in our case FFD) as the structured surface meshes, so that both surface representations have consistent displacements. The intersection lines defined by the triangulated surfaces are used to grow collar meshes using hyperbolic mesh marching algorithms. The hyperbolic mesh marching method is modified to improve its robustness and the quality of the surface mesh, especially near sharp edges. These tasks are implemented in the form of a Python module called *pySurf*.

This module is differentiated using reverse AD to fit directly into the reverse chain of the optimization framework used in the hybrid adjoint method and avoid the use of finite differences. The source code transformation is selectively applied to efficiently reuse information from the forward execution of the code, thus avoiding the repetition of iterative solutions and tree searches during the reverse propagation of derivatives.

The next step was the inclusion of the pySurf module into the MACH optimization framework, as explained in Chapter 3. The interactions among the multiple framework modules are detailed, both in the original and reverse executions of the code. This framework is first tested in a univariate optimization problem to address possible outcomes of component-based parametrization and variable overset connectivities. The study of the design space of this problem shows that changes in overset connectivity due to relative mesh displacement generate noise in integrated CFD quantities, such as C_L and C_D . This noise becomes smaller as the mesh is refined, and its amplitude is on the order of hundredths of drag counts for mesh sizes between one and six million cells, which are usually used in ASO problems. Nevertheless, the gradients consistently point towards the overall decrease direction, what still enables the use of gradient-based optimizers. Noise becomes an issue at the later stages of the optimization, when variations caused by the noise exceed the shallow slopes of the underlying function. Nevertheless, a significant improvement with respect to the baseline configuration has already been achieved at that point, and the distance to the true optimum falls within the uncertainties of the numerical models.

The analysis of the univariate problem gave us confidence to study more complex cases. The next two chapters refer to ASO applications that became possible due to the new geometry and mesh manipulation approach.

The MACH optimization framework is used, now with the pySurf module, to optimize the wing-body junction of a conventional aircraft configuration in Chapter 4. The optimizer managed to remove the flow separation at the junction trailing edge by using the fairing design variables, achieving a 1.8% reduction in drag. The simultaneous optimization of the wing and fairing reduced drag by 5.4%. The optimum fairing size gets smaller as wing design variables are included since the optimizer only uses the fairing design variables to locally improve the junction flow.

Next, an application of the optimization framework in the design of an unconventional aircraft configuration is shown. The wing and the strut of a SBW flying at transonic conditions is optimized in Chapter 5. The optimization reduces drag by 14.7% for a fixed lift constraint. The wing and strut design variables remove shock waves and flow separation at the wing-strut junction. The strut generates negative lift to reduce the flow speed in the wing-strut gap. The wing increases its midspan twist to compensate for the negative lift of the strut, so that the overall lift distribution becomes elliptical.

Despite the noise in the functions of interest, the optimizer manages to reduce drag in all cases, showing that ASO with variable overset connectivities is feasible. Analyses of the optimized designs provide insights for future aircraft configuration. These ASO cases demonstrate that the new geometry manipulation methodology expands the envelope of

possible applications for CFD-based optimization frameworks and also gives better shape control for the designer. In addition, this methodology contributes to the application of ASO beyond the conceptual aircraft design.

6.2 Contributions

The contributions of this thesis are the following:

1. Development of a component-based manipulation technique for gradient-based optimization that is fully integrated in the reverse AD chain used to compute derivatives in the hybrid adjoint method [118, 123]. This made possible the integration of intersection computation, automatic mesh generation, and overset connectivity variation in gradient-based optimization frameworks. I also modified the hyperbolic mesh marching to improve the robustness of the automatic collar mesh generation procedure.
2. Study of the impacts caused by changes in overset connectivities during the optimization iterations. The noise caused by connectivity changes does not hinder the use of gradient-based optimizers provided the mesh is sufficiently fine. The use of reverse AD for gradient computation is key, since gradients obtained via finite differences will be affected by the noise within small step sizes.
3. Simultaneous ASO of the wing and fairing of a conventional aircraft configuration [118, 123]. It was shown that the optimizer may ineffectively use the fairing design variables to address overall problems of the configuration in a fairing-alone optimization, while the simultaneous wing-fairing optimization leads to more efficient results.
4. First RANS-based ASO of a SBW configuration flying at a transonic condition [124, 125]. The component-based design variables successfully removed undesired flow features in the junction region. The new approach also prevented the mesh-related problems faced in previous optimization attempts.

Figure 6.1 highlights the main modifications done to the MACH optimization framework to achieve the result shown in this thesis.

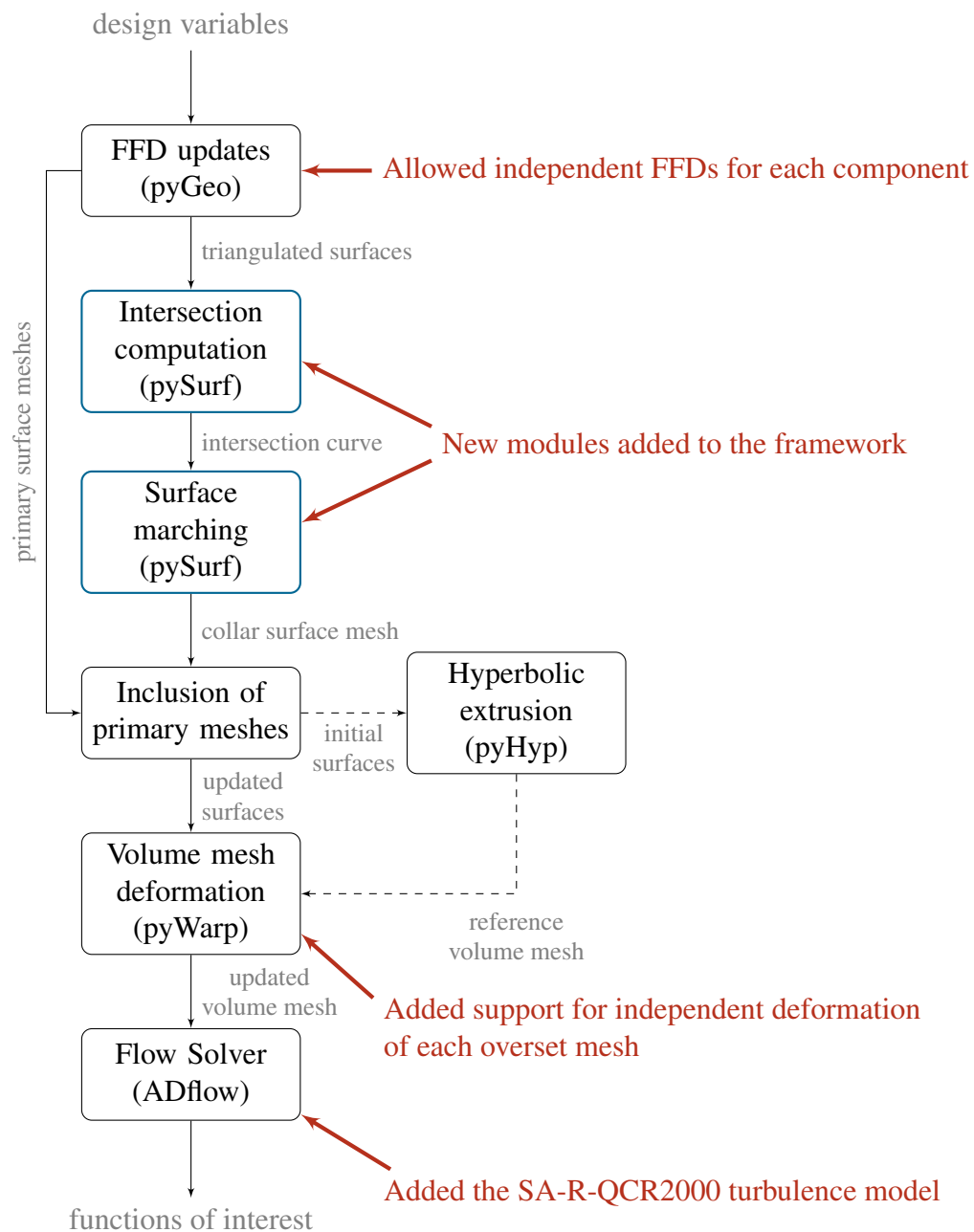


Figure 6.1: Execution order of the MACH framework modules during each optimization iteration (black lines) and summary of the modifications done to these modules as part of this thesis (in red). The dashed lines indicate processes that are only executed in the initialization step of the optimization.

6.3 Recommendations for future work

Possible research topics were identified during the execution of the current work. These suggestions involve not only new applications cases but also the improvement of the current ASO techniques:

1. *Hybrid mesh deformation approach.*

It is important to develop techniques to reduce the noise associated with the overset connectivity changes and facilitate the optimization convergence to more strict optimality levels.

A two-step optimization using different mesh deformation procedures may avoid the noise issue. First, the optimization would use the component-based geometry and mesh manipulation techniques introduced in this thesis to handle the more expressive changes in the design, while allowing changes in the overset connectivities. This optimization would be carried out until the optimizer quits due to numerical difficulties caused by the noise. Next, one would take the final design and start a new optimization using fixed overset connectivities and a global geometry and mesh deformation technique.

Since all components are deformed simultaneously, the relative position of the interpolated cells and their respective interpolation stencils are preserved, thus reducing the noise level. The global geometry manipulation approach would be feasible in this situation since only small changes in design variables are expected.

2. *Aerostructural optimization of the SBW configuration.*

The SBW optimization shown in this work did not consider the spanwise position of the wing strut-junction as a design variable since structural considerations are necessary to model the trade-offs associated with this design variable.

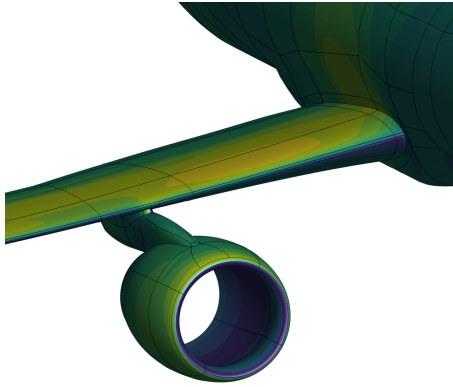
3. *Multipoint ASO of the wing-body junction.*

The consideration of multiple flight conditions in the ASO of the wing-body junction may lead to more realistic designs. This can also tell how robust the multipoint design case compared to the single point design.

4. *Other novel applications cases.*

The techniques developed in this work are applicable to other ASO problems that currently are not fully explored (Fig. 6.2), such as the nacelle-pylon-wing integration and engine placement, the design of aft propulsor fans with buried vertical tail, and

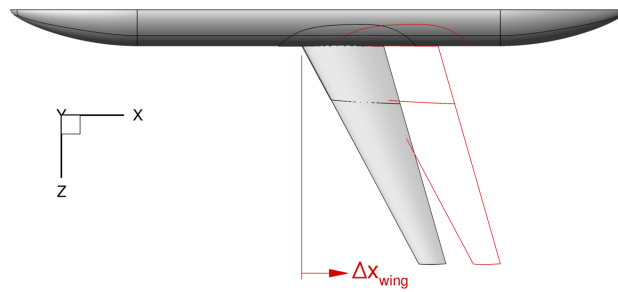
the optimization the longitudinal position of the wing with respect to the fuselage while accounting for stability and controllability considerations.



(a) Nacelle integration.



(b) Aft propulsor.



(c) Wing longitudinal position.

Figure 6.2: Possible applications for the component-based parametrization technique developed in this thesis.

APPENDIX A

Surface mesh generation

In this chapter we provide an overview of the hyperbolic equation used to generate the surface meshes. First, we provide an alternative derivations of the mesh generation equations proposed by [37] in Sec. A.1. Then, we discuss the modifications introduced in this thesis to improve the robustness of the automatic mesh generation in the remaining sections.

A.1 Hyperbolic equations for surface generation

Let $\mathbf{r} = (x, y, z)^T$ represent physical coordinates of mesh nodes in a tridimensional space while $\xi(x, y, z)$ and $\eta(x, y, z)$ represent bidimensional parametric coordinates of a tridimensional surface. Then, $\mathbf{r}(\xi, \eta)$ is the mapping of this surface from the bidimensional parametric space to the tridimensional physical space.

To generate a surface mesh with n_i by n_j nodes, we first define a regular grid in the parametric space with the same number of nodes spaced by $\Delta\xi = 1$ and $\Delta\eta = 1$. Therefore, the parametric mesh is within the bounds defined by $[1, n_i] \times [1, n_j]$. The initial curve used to propagate the surface mesh is defined by $\mathbf{r}(\xi, 1)$ for $1 \leq \xi \leq n_i$, and the mesh is marched in the η direction.

The hyperbolic mesh marching equations is a set of partial differential equations that defines the $\mathbf{r}(\xi, \eta)$ mapping from parametric to physical coordinates. The sampling of this function on the regular grid we introduced in the previous paragraph will determine the physical location of the surface mesh nodes. The hyperbolic mesh marching equations are:

$$\mathbf{r}_\xi \cdot \mathbf{r}_\eta = 0, \tag{A.1a}$$

$$\mathbf{n} \cdot (\mathbf{r}_\xi \times \mathbf{r}_\eta) = \Delta S, \tag{A.1b}$$

$$\mathbf{n} \cdot \mathbf{r}_\eta = 0, \tag{A.1c}$$

where \mathbf{r}_ξ is the partial derivative with respect to ξ , \mathbf{r}_η is the partial derivative with respect to η . The vector $\mathbf{n} = (n_1, n_2, n_3)^T$ is the local normal vector (in physical coordinates) from the reference surface onto which the mesh will be generated. ΔS is a factor that controls the area of the cells in physical coordinates, and its value may vary within the bounds of the parametric space to control mesh resolution. Equation (A.1a) enforces cell orthogonality, Eq. (A.1b) controls the cell density, and Eq. (A.1c) indicates that the marching direction should be parallel to the surface.

The PDE are hyperbolic with respect to the marching direction η . Therefore, we can generate the mesh layer by layer in this direction, from layer $\eta = j - 1$ to layer $\eta = j$. This nonlinear partial differential equations (PDE) can be linearized around a point $\mathbf{r}^0 = (x^0, y^0, z^0) = \mathbf{r}_{i,j-1}$ that belongs to the layer $\eta = j - 1$. Since, the PDE should be valid for this point we have:

$$\mathbf{r}_\xi^0 \cdot \mathbf{r}_\eta^0 = 0 \quad (\text{A.2a})$$

$$\mathbf{n}^0 \cdot (\mathbf{r}_\xi^0 \times \mathbf{r}_\eta^0) = \Delta S^0 \quad (\text{A.2b})$$

$$\mathbf{n}^0 \cdot \mathbf{r}_\eta^0 = 0 \quad (\text{A.2c})$$

Now, by setting $\mathbf{r} = \mathbf{r}^0 + \Delta \mathbf{r}$ in Eq. (A.1) we obtain the linearized equations:

$$\mathbf{r}_\eta^0 \cdot \Delta \mathbf{r}_\xi + \mathbf{r}_\xi^0 \cdot \Delta \mathbf{r}_\eta = 0 \quad (\text{A.3a})$$

$$(\mathbf{n}^0 \times \mathbf{r}_\eta^0) \cdot \Delta \mathbf{r}_\xi + (\mathbf{n}^0 \times \mathbf{r}_\xi^0) \cdot \Delta \mathbf{r}_\eta = \Delta S - \Delta S^0 \quad (\text{A.3b})$$

$$\mathbf{n}^0 \cdot \Delta \mathbf{r}_\eta = 0, \quad (\text{A.3c})$$

We can also cast Eq. (A.3) using the individual physical coordinates in matrix form as:

$$\mathbf{A}^0 \cdot \Delta \mathbf{r}_\xi + \mathbf{B}^0 \cdot \Delta \mathbf{r}_\eta = \mathbf{g} - \mathbf{g}^0, \quad (\text{A.4})$$

where:

$$\mathbf{A}^0 = \begin{bmatrix} x_\eta^0 & y_\eta^0 & z_\eta^0 \\ n_2^0 z_\eta^0 - n_3^0 y_\eta^0 & n_3^0 x_\eta^0 - n_1^0 z_\eta^0 & n_1^0 y_\eta^0 - n_2^0 x_\eta^0 \\ 0 & 0 & 0 \end{bmatrix}, \quad (\text{A.5})$$

$$\mathbf{B}^0 = \begin{bmatrix} x_\xi^0 & y_\xi^0 & z_\xi^0 \\ n_2^0 z_\xi^0 - n_3^0 y_\xi^0 & n_3^0 x_\xi^0 - n_1^0 z_\xi^0 & n_1^0 y_\xi^0 - n_2^0 x_\xi^0 \\ n_1 & n_2 & n_3 \end{bmatrix}, \quad (\text{A.6})$$

$$\mathbf{g} = \begin{bmatrix} 0 \\ \Delta S \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{g}^0 = \begin{bmatrix} 0 \\ \Delta S^0 \\ 0 \end{bmatrix}. \quad (\text{A.7})$$

We need to compute partial derivatives with respect to ξ and η at the reference point \mathbf{r}^0 to populate these matrices. Since we generate the surface mesh layer by layer, we know the position of all other nodes of the previous layer ($\eta = j - 1$). Therefore, we can compute the ξ derivatives for the node $(i, j - 1)$ via central difference (using $\Delta\xi = 1$):

$$\mathbf{r}_\xi^0 = \frac{\mathbf{r}_{i+1,j-1} - \mathbf{r}_{i-1,j-1}}{2} \quad (\text{A.8})$$

Once we have the ξ derivatives, we can obtain the η derivatives via Eq. (A.2). This equation can be modified to use the same \mathbf{B}^0 matrix from Eq. (A.6):

$$\begin{aligned} \mathbf{r}_\xi^0 \cdot \mathbf{r}_\eta^0 &= 0 \\ (\mathbf{n}^0 \times \mathbf{r}_\xi^0) \cdot \mathbf{r}_\eta^0 &= \Delta S^0 \\ \mathbf{n}^0 \cdot \mathbf{r}_\eta^0 &= 0 \end{aligned} \quad \Longrightarrow \quad \mathbf{B}^0 \cdot \mathbf{r}_\eta^0 = \begin{bmatrix} 0 \\ \Delta S^0 \\ 0 \end{bmatrix} \quad (\text{A.9})$$

Since we already compute the inverse of \mathbf{B}^0 to solve Eq. (A.9), we can modify Eq. (A.4) to use the same matrix:

$$\Delta \mathbf{r}_\eta + (\mathbf{B}^0)^{-1} \cdot \mathbf{A}^0 \cdot \Delta \mathbf{r}_\xi = (\mathbf{B}^0)^{-1} \cdot (\mathbf{g} - \mathbf{g}^0). \quad (\text{A.10})$$

We can rewrite this equation as:

$$\Delta \mathbf{r}_\eta + \mathbf{C}^0 \cdot \Delta \mathbf{r}_\xi = (\mathbf{B}^0)^{-1} \cdot (\mathbf{g} - \mathbf{g}^0), \quad (\text{A.11})$$

where:

$$\mathbf{C}^0 = (\mathbf{B}^0)^{-1} \cdot \mathbf{A}^0 \quad (\text{A.12})$$

Chan and Buning [37] suggests the addition of second-order dissipation terms to stabilize the hyperbolic marching. This is achieved by adding an implicit dissipation term to the left-hand side of the equation and an explicit dissipation term to the right-hand side:

$$\Delta \mathbf{r}_\eta + \mathbf{C}^0 \cdot \Delta \mathbf{r}_\xi - \epsilon_i \Delta \mathbf{r}_{\xi\xi} = (\mathbf{B}^0)^{-1} \cdot (\mathbf{g} - \mathbf{g}^0) + \epsilon_e \mathbf{r}_{\xi\xi}^0 \quad (\text{A.13})$$

Where ϵ_i is the implicit dissipation factor and ϵ_e is the explicit dissipation factor. We reversed the sign of the explicit dissipation term compared to the cited reference since we verified that the original equations were causing instabilities during the mesh generation.

We can apply backward difference with respect to the $\eta = j$ layer for η derivatives and central difference for ξ derivatives to write:

$$\begin{aligned} \Delta \mathbf{r}_\eta &\approx \Delta \mathbf{r}_{i,j} - \Delta \mathbf{r}_{i,j-1} \\ \Delta \mathbf{r}_\xi &\approx \frac{\Delta \mathbf{r}_{i+1,j} - \Delta \mathbf{r}_{i-1,j}}{2} \end{aligned} \quad (\text{A.14})$$

$$\Delta \mathbf{r}_{\xi\xi} \approx \Delta \mathbf{r}_{i+1,j} - 2\Delta \mathbf{r}_{i,j} + \Delta \mathbf{r}_{i-1,j}$$

The second order partial derivative of the explicit dissipation term can be computed with:

$$\mathbf{r}_{\xi\xi}^0 = \mathbf{r}_{i+1,j-1} - 2\mathbf{r}_{i,j-1} + \mathbf{r}_{i-1,j-1} \quad (\text{A.15})$$

Using Eq. (A.14) into Eq. (A.13) gives:

$$\begin{aligned} \left(-\frac{\mathbf{C}^0}{2} - \epsilon_i \mathbf{I} \right) \cdot \Delta \mathbf{r}_{i-1,j} + (1 + 2\epsilon_i) \mathbf{I} \cdot \Delta \mathbf{r}_{i,j} + \left(\frac{\mathbf{C}^0}{2} - \epsilon_i \mathbf{I} \right) \cdot \Delta \mathbf{r}_{i+1,j} = \\ = (\mathbf{B}^0)^{-1} \cdot (\mathbf{g} - \mathbf{g}^0) + \epsilon_e \mathbf{r}_{\xi\xi}^0 + \Delta \mathbf{r}_{i,j-1} \end{aligned} \quad (\text{A.16})$$

where \mathbf{I} represents a 3×3 identity matrix. If we use the backward difference approximation and Eq. (A.9) we can write:

$$\Delta \mathbf{r}_{i,j-1} = \mathbf{r}_{i,j-1} - \mathbf{r}_{i,j-2} \approx \mathbf{r}_\eta^0 = (\mathbf{B}^0)^{-1} \cdot \mathbf{g}^0 \quad (\text{A.17})$$

This lets us rewrite Eq. (A.16) as:

$$\mathbf{L}_{i,j} \cdot \Delta \mathbf{r}_{i-1,j} + \mathbf{M}_{i,j} \cdot \Delta \mathbf{r}_{i,j} + \mathbf{N}_{i,j} \cdot \Delta \mathbf{r}_{i+1,j} = \mathbf{f}_{i,j} \quad (\text{A.18})$$

where:

$$\mathbf{L}_{i,j} = -\frac{\mathbf{C}_{i,j}^0}{2} - \epsilon_i \mathbf{I} \quad (\text{A.19})$$

$$\mathbf{M}_{i,j} = (1 + 2\epsilon_i)\mathbf{I} \quad (\text{A.20})$$

$$\mathbf{N}_{i,j} = \frac{\mathbf{C}_{i,j}^0}{2} - \epsilon_i\mathbf{I} \quad (\text{A.21})$$

$$\mathbf{f}_{i,j} = (\mathbf{B}_{i,j}^0)^{-1} \cdot \mathbf{g}_{i,j} + \epsilon_e \mathbf{r}_{\xi\xi}^0 \quad (\text{A.22})$$

We added the i, j indices to highlight that these matrices and vectors are different for each node. If we apply these equation to all nodes of the current curve we get a block tridiagonal matrix that will implicitly give $\Delta \mathbf{r}_{i,j}$ values to determine the nodes of the next mesh layer:

$$\begin{bmatrix} \mathbf{M}_{1,j} & \mathbf{N}_{1,j} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{L}_{2,j} & \mathbf{M}_{2,j} & \mathbf{N}_{2,j} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_{3,j} & \mathbf{M}_{3,j} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{M}_{n_i,j} \end{bmatrix} \cdot \begin{bmatrix} \Delta \mathbf{r}_{1,j} \\ \Delta \mathbf{r}_{2,j} \\ \Delta \mathbf{r}_{3,j} \\ \vdots \\ \Delta \mathbf{r}_{n_i,j} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{1,j} \\ \mathbf{f}_{2,j} \\ \mathbf{f}_{3,j} \\ \vdots \\ \mathbf{f}_{n_i,j} \end{bmatrix} \quad (\text{A.23})$$

This system can also be compactly expressed as:

$$\mathbf{K}_j \cdot \Delta \mathbf{R}_j = \mathbf{F}_j \quad (\text{A.24})$$

Once we solve this linear system we compute the position of the j -th layer nodes with:

$$\mathbf{r}_{i,j}^* = \mathbf{r}_{i,j-1} + \Delta \mathbf{r}_{i,j} \quad (\text{A.25})$$

The new nodes do not necessarily lie on the reference surface. Therefore, we still need to project these nodes to the surface to determine the correct position of the next layer of points. This procedure is represented by the projection operator F_{proj} :

$$\mathbf{r}_{i,j} = F_{\text{proj}}(\mathbf{r}_{i,j}^*) \quad (\text{A.26})$$

Now we can use $\mathbf{r}_{i,j}$ as the starting curve for the computation of the next layer by repeating the process, from $j = 2$ until $j = n_j$.

A.2 Marching distance

We need to provide a desired cell area ΔS to control the mesh growth. In this section we outline the steps used to compute ΔS that we developed for the pySurf module.

Let Δd_j be the average distance between layer $j + 1$ and j , which is also the average height of the cells between these layers. The user initially specifies the number of layers n_j , the marching distance of the first layer Δd_1 , and the desired marching distance d for the entire surface mesh. Assuming a geometric progression of cell sizes, we can relate the average cell heights of two consecutive layers as:

$$\Delta d_j = q \cdot \Delta d_{j-1} \quad (\text{A.27})$$

Where $q > 1$ is a constant growth factor. The sum of all average cell distances between layers should be equal to the overall marching distance of the mesh:

$$d = \sum_{j=1}^{n_j-1} \Delta d_j = \Delta d_1 \frac{1 - q^{n_j-1}}{1 - q} \quad (\text{A.28})$$

This can be rearranged as the following relationship:

$$\Delta d_1(1 - q^{n_j-1}) - d(1 - q) = 0 \quad (\text{A.29})$$

We solve Eq. (A.29) for q using Newton's method. Once we have q , we can determine the average cell height Δd_j of every layer with Eq. (A.27). However, we still need to convert this distance into an area metric to be used as ΔS . We do this by multiplying the average cell height by an *average* cell width corresponding to each node of the curve. We defined this average cell width $w_{i,j}$ as the average of the distance of a given node to its two neighbors:

$$w_{i,j} = \frac{|\mathbf{r}_{i+1,j} - \mathbf{r}_{i,j}| + |\mathbf{r}_{i-1,j} - \mathbf{r}_{i,j}|}{2} \quad (\text{A.30})$$

The boundary nodes use only the distance to their single neighbor. Then we can compute the area factor with:

$$\Delta S_{i,j} = w_{i,j-1} \Delta d_{j-1} \quad (\text{A.31})$$

When marching the j -th layer, we use $\Delta S_{i,j}$ to compute $\mathbf{g}_{i,j}$ in Eq. (A.22) and $\Delta S_{i,j-1}$ to obtain \mathbf{r}_η^0 in Eq. (A.9).

A.3 Dissipation coefficients

The dissipation coefficients ϵ_i and ϵ_e are important to stabilize the hyperbolic marching scheme. Large values of dissipation add stability to the mesh generation process while sacrificing cell orthogonality. Ideally, one should use the smallest dissipation value that ensures a stable solution. Consequently, we need methods that automatically vary these coefficients to avoid excessive dissipation in regions where it is not necessary. We adapted the dissipation coefficients used for volume mesh generation [38] to make them suitable for surface mesh generation.

The dissipation coefficients are computed in pySurf as:

$$\epsilon_e = \epsilon_0 \cdot N \cdot R \quad \text{and} \quad \epsilon_i = 2\epsilon_e, \quad (\text{A.32})$$

where ϵ_0 is a user provided factor (ranging from 4.0 to 15.0). N is an approximation of the $\| [C^0] \|$ matrix norm to keep the same relative scale the dissipation terms to the other terms of Eq. (A.13):

$$N = \frac{|\mathbf{r}_\eta|}{|\mathbf{r}_\xi|} \quad (\text{A.33})$$

The R coefficient is composed by three factors that control the dissipation:

$$R = s_j \cdot a \cdot \bar{d} \quad (\text{A.34})$$

It is useful to use low dissipation factors for the first layers of the mesh to guarantee orthogonality with respect to the baseline curve. On the other, high dissipation values in the outer layers are important to prevent crossing of grid lines. The s_j factor steadily increases the dissipation along the marching direction:

$$s_j = \begin{cases} \sqrt{\frac{j-1}{n_j-1}} & \text{for } 2 \leq j \leq n_{\text{trans}} \\ \sqrt{\frac{n_{\text{trans}}-1}{n_j-1}} & \text{for } n_{\text{trans}} + 1 \leq j \leq n_j \end{cases} \quad (\text{A.35})$$

where $n_{\text{trans}} = 3n_j/4$.

The a factor increases dissipation in concave segments of the layer to spread the grid lines and avoid invalid cells, while there is no need to increase the dissipation if the layer is locally convex:

$$a = \begin{cases} 1 & \text{if } \alpha \leq \pi \\ \frac{1}{1 - \cos^2(\alpha/2)} & \text{if } \alpha > \pi \end{cases} \quad (\text{A.36})$$

α is the angle between the two vectors connecting a node and its neighbors. For the i -th node of the layer we define:

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{r}_i - \mathbf{r}_{i-1} \\ \mathbf{v}_2 &= \mathbf{r}_{i+1} - \mathbf{r}_i \end{aligned} \quad (\text{A.37})$$

Then we compute the acute angle between the vectors' orientation:

$$\alpha^* = \arccos\left(\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{|\mathbf{v}_1| \cdot |\mathbf{v}_2|}\right) \quad (\text{A.38})$$

Then we detect if we have a concave ($\alpha < \pi$) or convex ($\alpha \geq \pi$) corner with:

$$\alpha = \begin{cases} \pi - \alpha^* & \text{if } \mathbf{n}_{i,j} \cdot (\mathbf{v}_1 \times \mathbf{v}_2) < 0 \\ \pi + \alpha^* & \text{if } \mathbf{n}_{i,j} \cdot (\mathbf{v}_1 \times \mathbf{v}_2) \geq 0 \end{cases} \quad (\text{A.39})$$

where \mathbf{n}_i is the reference surface normal vector at the point \mathbf{r}_i .

The last factor of the dissipation coefficient is \bar{d} , which increases dissipation when nodes are getting closer for consecutive layers. First, we compute the node distribution sensor $b_{i,j}$ as:

$$b_{i,j} = \frac{|\mathbf{r}_{i+1,j-1} - \mathbf{r}_{i,j-1}| + |\mathbf{r}_{i-1,j-1} - \mathbf{r}_{i,j-1}|}{|\mathbf{r}_{i+1,j} - \mathbf{r}_{i,j}| + |\mathbf{r}_{i-1,j} - \mathbf{r}_{i,j}|} \quad (\text{A.40})$$

Then we compute \bar{d} as:

$$\bar{d} = \max\left(0.1, (b_{i,j})^{2/s_j}\right) \quad (\text{A.41})$$

A.4 Boundary conditions and special cases

This section discusses about how the $\mathbf{L}_{i,j}$, $\mathbf{M}_{i,j}$, and $\mathbf{N}_{i,j}$ matrices from Eq. (A.18) can be modified for special nodes, such as the boundary nodes ($i=1$ or $i = n_i$).

A.4.1 Periodic boundary conditions

If we start the marching process with a periodic curve, we may use periodic boundary conditions. In this case, we use the stencil $(n_i, j - 1)$, $(1, j - 1)$, and $(2, j - 1)$ to compute

the \mathbf{A}^0 and \mathbf{B}^0 matrices for node $(1, j)$. We also enforce the last node to have the same displacement as the first one with the following equation:

$$\mathbf{I} \cdot \Delta \mathbf{r}_{1,j} - \mathbf{I} \cdot \Delta \mathbf{r}_{n_i,j} = 0 \quad (\text{A.42})$$

The identity matrices replace the \mathbf{L} , \mathbf{M} , and \mathbf{N} matrices in the last row of Eq. (A.23).

A.4.2 Free nodes

If the curve is not periodic, we cannot use central differences at the extremities of the baseline curve. So we modify the marching equations of these nodes to use one-sided second-order differences.

- For node $i = 1$:

Eq. (A.8) becomes:

$$\mathbf{r}_\xi^0 = \frac{-3 \cdot \mathbf{r}_{1,j-1} + 4 \cdot \mathbf{r}_{2,j-1} - \mathbf{r}_{3,j-1}}{2} \quad (\text{A.43})$$

And Eq. A.18 is replaced by:

$$\mathbf{L}_{1,j} \cdot \Delta \mathbf{r}_{1,j} + \mathbf{M}_{1,j} \cdot \Delta \mathbf{r}_{2,j} + \mathbf{N}_{1,j} \cdot \Delta \mathbf{r}_{3,j} = \mathbf{f}_{1,j} \quad (\text{A.44})$$

where:

$$\mathbf{L}_{1,j} = -\frac{3}{2} \mathbf{C}_{1,j}^0 + (1 - \epsilon_i) \mathbf{I} \quad (\text{A.45})$$

$$\mathbf{M}_{1,j} = -2 \mathbf{C}_{1,j}^0 + 2 \epsilon_i \mathbf{I} \quad (\text{A.46})$$

$$\mathbf{N}_{1,j} = -\frac{1}{2} \mathbf{C}_{1,j}^0 - \epsilon_i \mathbf{I} \quad (\text{A.47})$$

- For node $i = n_i$:

Eq. (A.8) becomes:

$$\mathbf{r}_\xi^0 = \frac{\mathbf{r}_{n_i-2,j-1} - 4 \cdot \mathbf{r}_{n_i-1,j-1} + 3 \cdot \mathbf{r}_{n_i,j-1}}{2} \quad (\text{A.48})$$

And Eq. A.18 is replaced by:

$$\mathbf{L}_{n_i,j} \cdot \Delta \mathbf{r}_{n_i-2,j} + \mathbf{M}_{n_i,j} \cdot \Delta \mathbf{r}_{n_i-1,j} + \mathbf{N}_{n_i,j} \cdot \Delta \mathbf{r}_{n_i,j} = \mathbf{f}_{n_i,j} \quad (\text{A.49})$$

where:

$$\mathbf{L}_{n_i,j} = \frac{1}{2} \mathbf{C}_{n_i,j}^0 - \epsilon_i \mathbf{I} \quad (\text{A.50})$$

$$\mathbf{M}_{n_i,j} = -2\mathbf{C}_{n_i,j}^0 + 2\epsilon_i \mathbf{I} \quad (\text{A.51})$$

$$\mathbf{N}_{n_i,j} = \frac{3}{2} \mathbf{C}_{n_i,j}^0 + (1 - \epsilon_i) \mathbf{I} \quad (\text{A.52})$$

We can control the splay of the mesh boundaries by changing the marching distance of the end nodes compared to the interior ones. This can be achieved by modifying Eq. (A.31) as follows:

$$\Delta S_{1,j} = w_{1,j-1} \Delta d_{j-1} (1 - \sigma) \quad (\text{A.53})$$

$$\Delta S_{n_i,j} = w_{n_i,j-1} \Delta d_{j-1} (1 - \sigma) \quad (\text{A.54})$$

where σ is the splay factor. If the marching distance of the end nodes is smaller than the interior nodes ($\sigma > 0$), the mesh splays out, while if the end nodes march farther than the interior ones ($\sigma < 0$), the mesh splays in, as shown in Fig. A.1.

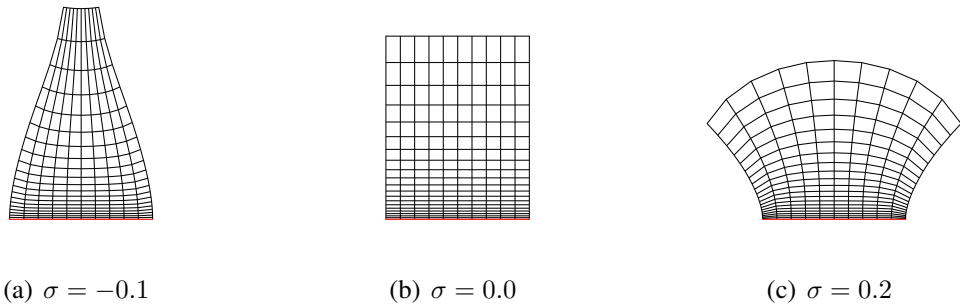


Figure A.1: Meshes marched with different splay factors. The red line is the baseline curve for the marching process.

A.4.3 Sharp angles

If the current layer of nodes has a sharp angle, we determine the marching direction for the corner node as the average of the distance marched by its neighbors, since this is more robust [38]. We classify a node as a corner if $\alpha < 70$ degrees, where α is the local layer angle computed with Eq. (A.39).

The contribution of a corner node (i, j) to Eq. A.23 becomes:

$$-\mathbf{I} \cdot \Delta \mathbf{r}_{i-1,j} + 2\mathbf{I} \cdot \Delta \mathbf{r}_{i,j} - \mathbf{I} \cdot \Delta \mathbf{r}_{i+1,j} = 0 \quad (\text{A.55})$$

A.4.4 Guide curves

The user can specify guide curves to control the marching direction of certain nodes and ensure that relevant surface features, such as sharp corners, are represented by the mesh (Fig. 2.2). These guide curves are represented by a set of points connected by the linear segments. At the beginning of the marching process, we detect which nodes from the baseline curve are the closest ones to each guide curve. Once these nodes are assigned to their corresponding guide curves, their marching direction is dictated by the curve tangent vector rather than the reference surface normal. Let $\mathbf{t}_{g,j-1}$ be the curve tangent vector at the projection of node $\mathbf{r}_{g,j-1}$ on this curve. The contribution from node (g, j) to Eq. (A.23) becomes:

$$\mathbf{I} \cdot \Delta \mathbf{r}_{g,j} = \Delta d_{j-1} \mathbf{t}_{g,j-1} \quad (\text{A.56})$$

After the solution of Eq. (A.23), we project the node (g, j) back to the guide curve rather than the reference surface before marching the next layer.

A.5 Guide curve blending

If a guide curve is oblique to the marching direction of the mesh, the hyperbolic equations will generate highly skewed or even invalid cells due to the conflicting marching directions. This happens because only the guided node uses the guide curve tangent to compute its displacement (Eq. (A.56)). We developed a blending procedure that extends the influence to the guide curve tangent to other nodes, preventing crossed grid lines.

Let the node (g, j) be the node guided by a curve while (i, j) is the node where we apply the blending. We define an user-provided blending factor ν that combines the displacement of the guided node and the displacement originally predicted by the hyperbolic marching

equations $\Delta \mathbf{r}_{i,j}$:

$$\Delta \mathbf{r}_{i,j}^b = \nu \Delta \mathbf{r}_{g,j} + (1 - \nu) \Delta \mathbf{r}_{i,j} \quad (\text{A.57})$$

where $\Delta \mathbf{r}_{i,j}^b$ is the blended displacement. If $\nu = 1$, node (i, j) will march parallel to the guide node, whereas no blending occurs for $\nu = 0$. If we substitute $\Delta \mathbf{r}_{i,j}$ from Eq. (A.57) into Eq. (A.18) we get the blended marching equation:

$$\mathbf{L}_{i,j} \cdot \Delta \mathbf{r}_{i-1,j} + \mathbf{M}_{i,j}^b \cdot \Delta \mathbf{r}_{i,j} + \mathbf{N}_{i,j} \cdot \Delta \mathbf{r}_{i+1,j} = \mathbf{f}_{i,j}^b \quad (\text{A.58})$$

where we only need to redefine:

$$\mathbf{M}_{i,j}^b = \frac{1}{1 - \nu} \mathbf{M}_{i,j} \quad (\text{A.59})$$

$$\mathbf{f}_{i,j}^b = \mathbf{f}_{i,j} + \frac{\nu}{1 - \nu} \mathbf{M}_{i,j} \cdot (\Delta d_{j-1} \mathbf{t}_{g,j-1}) \quad (\text{A.60})$$

We apply the blending to the four nearest neighbors of the guided node. The blending factor is halved for the last two neighbors. Figure A.2 shows a blending application for a blunt trailing edge mesh that is marched from an oblique intersection.

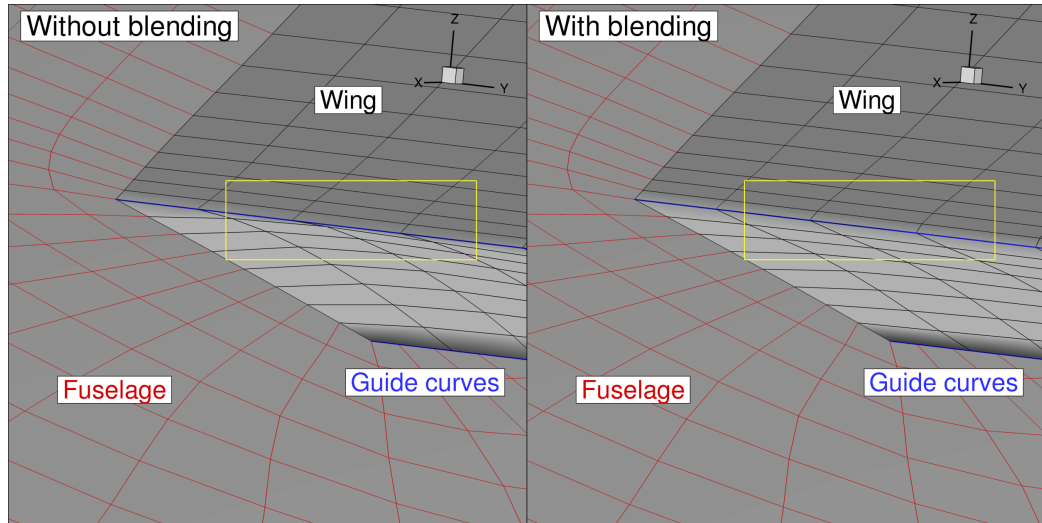


Figure A.2: The use of blending factor ($\nu = 0.5$) avoids highly skewed cells where guide curves are oblique to the intersection line.

A.6 Pseudo marching steps

Large marching distances may lead to instabilities in the mesh generation process. We solve this by using pseudo marching steps. In other words, instead of marching directly to the desired marching distance, we execute many intermediate steps until reaching this distance. The nodes of these intermediate layers are not included in the final surface mesh. They are stored nonetheless for the propagation of derivatives with reverse AD.

The number of intermediate steps to march from layer $j - 1$ to layer j is based on the aspect ratio of the cells ($c_{i,j}$), which we define as the ratio of their height Δd_{j-1} , given by Eq. (A.27), to their width $w_{i,j-1}$, defined by Eq. (A.30):

$$c_{i,j} = \frac{\Delta d_{j-1}}{w_{i,j-1}} \quad (\text{A.61})$$

The number of pseudo marching steps n^* is given by:

$$n^* = \left\lceil \frac{\max_i c_{i,j}}{c_{\max}} \right\rceil \quad (\text{A.62})$$

Where c_{\max} is an user-provided value (usually between 5.0 and 10.0). Then we compute the pseudo marching distance Δd_p^* as:

$$\Delta d_{j-1}^* = \Delta d_{j-1} / n^* \quad (\text{A.63})$$

Thus, instead of directly marching to a distance Δd_{j-1} , we execute n^* marching steps of size Δd_{j-1}^* .

BIBLIOGRAPHY

- [1] Anderson Jr, J. D., *Fundamentals of Aerodynamics*, Tata McGraw-Hill Education, 2010.
- [2] Nemec, M., and Zingg, D. W., “Newton-Krylov algorithm for aerodynamic design using the Navier-Stokes equations,” *AIAA Journal*, Vol. 40, No. 6, 2002, pp. 1146–1154.
- [3] Poole, D., Allen, C., and Rendall, T., “High-fidelity aerodynamic shape optimization using efficient orthogonal modal design variables with a constrained global optimizer,” *Computers & Fluids*, Vol. 143, 2017, pp. 1–15.
- [4] Masters, D., Taylor, N., Rendall, T., and Allen, C., “Multilevel Subdivision Parameterization Scheme for Aerodynamic Shape Optimization,” *AIAA Journal*, 2017, pp. 1–16.
- [5] Hicken, J. E., and Zingg, D. W., “Induced-drag minimization of nonplanar geometries based on the Euler equations,” *AIAA Journal*, Vol. 48, No. 11, 2010, pp. 2564–2575.
- [6] Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., “Aerodynamic Shape Optimization Investigations of the Common Research Model Wing Benchmark,” *AIAA Journal*, Vol. 53, No. 4, 2015, pp. 968–985. doi:[10.2514/1.J053318](https://doi.org/10.2514/1.J053318).
- [7] Kenway, G. K. W., and Martins, J. R. R. A., “Multipoint Aerodynamic Shape Optimization Investigations of the Common Research Model Wing,” *AIAA Journal*, Vol. 54, No. 1, 2016, pp. 113–128. doi:[10.2514/1.J054154](https://doi.org/10.2514/1.J054154).
- [8] Dumont, A., and Méheut, M., “Gradient-Based Optimization of CRM Wing-alone and Wing-body-tail Configurations by RANS Adjoint Technique,” *54th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics (AIAA), 2016. doi:[10.2514/6.2016-1293](https://doi.org/10.2514/6.2016-1293).
- [9] Chen, S., Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., “Aerodynamic Shape Optimization of the Common Research Model Wing-Body-Tail Configuration,” *Journal of Aircraft*, Vol. 53, No. 1, 2016, pp. 276–293. doi:[10.2514/1.C033328](https://doi.org/10.2514/1.C033328).
- [10] Merle, A., Stueck, A., and Rempke, A., “An Adjoint-based Aerodynamic Shape Optimization Strategy for Trimmed Aircraft with Active Engines,” *35th AIAA Applied Aerodynamics Conference*, 2017. doi:[10.2514/6.2017-3754](https://doi.org/10.2514/6.2017-3754), AIAA 2017-3754.

- [11] Ivaldi, D., Secco, N. R., Chen, S., Hwang, J. T., and Martins, J. R. R. A., “Aerodynamic Shape Optimization of a Truss-Braced-Wing Aircraft,” *Proceedings of the 16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Dallas, TX, 2015. doi:[10.2514/6.2015-3436](https://doi.org/10.2514/6.2015-3436).
- [12] Gray, J. S., Mader, C. A., Kenway, G. K., and Martins, J. R., “Modeling Boundary Layer Ingestion Using a Coupled Aeropropulsive Analysis,” *Journal of Aircraft*, 2017, pp. 1–9.
- [13] Kenway, G. K., and Kiris, C. C., “Aerodynamic shape optimization of the STARC-ABL concept for minimal inlet distortion,” *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018. doi:[10.2514/6.2018-1912](https://doi.org/10.2514/6.2018-1912), AIAA 2018-1912.
- [14] Gray, J., Mader, C. A., Kenway, G. K. W., and Martins, J. R. R. A., “Modeling Boundary Layer Ingestion Using a Coupled Aeropropulsive Analysis,” *Journal of Aircraft*, 2018. doi:[10.2514/1.C034601](https://doi.org/10.2514/1.C034601), (In press).
- [15] Kroll, N., Abu-Zurayk, M., Dimitrov, D., Franz, T., Führer, T., Gerhold, T., Görtz, S., Heinrich, R., Ilic, C., Jepsen, J., et al., “DLR project Digital-X: towards virtual aircraft design and flight testing based on high-fidelity methods,” *CEAS Aeronautical Journal*, Vol. 7, No. 1, 2016, pp. 3–27.
- [16] Zingg, D. W., Nemec, M., and Pulliam, T. H., “A Comparative Evaluation of Genetic and Gradient-Based Algorithms Applied to Aerodynamic Optimization,” *European Journal of Computational Mechanics*, Vol. 17, No. 1–2, 2008, pp. 103–126. doi:[10.3166/remn.17.103-126](https://doi.org/10.3166/remn.17.103-126).
- [17] Yu, Y., Lyu, Z., Xu, Z., and Martins, J. R. R. A., “On the Influence of Optimization Algorithm and Starting Design on Wing Aerodynamic Shape Optimization,” *Aerospace Science and Technology*, Vol. 75, 2018, pp. 183–199. doi:[10.1016/j.ast.2018.01.016](https://doi.org/10.1016/j.ast.2018.01.016).
- [18] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE transactions on evolutionary computation*, Vol. 6, No. 2, 2002, pp. 182–197.
- [19] Jansen, P. W., and Perez, R. E., “Constrained structural design optimization via a parallel augmented Lagrangian particle swarm optimization approach,” *Computers & Structures*, Vol. 89, No. 13-14, 2011, pp. 1352–1366.
- [20] Nelder, J. A., and Mead, R., “A simplex method for function minimization,” *The Computer Journal*, Vol. 7, No. 4, 1965, pp. 308–313.
- [21] Lyu, Z., Xu, Z., and Martins, J. R. R. A., “Benchmarking Optimization Algorithms for Wing Aerodynamic Design Optimization,” *Proceedings of the 8th International Conference on Computational Fluid Dynamics*, Chengdu, Sichuan, China, 2014. ICCFD8-2014-0203.

- [22] Nocedal, J., and Wright, S. J., *Numerical Optimization*, 2nd ed., Springer-Verlag, 2006.
- [23] Peter, J. E. V., and Dwight, R. P., “Numerical Sensitivity Analysis for Aerodynamic Optimization: A Survey of Approaches,” *Computers and Fluids*, Vol. 39, No. 3, 2010, pp. 373–391. doi:[10.1016/j.compfluid.2009.09.013](https://doi.org/10.1016/j.compfluid.2009.09.013).
- [24] Griewank, A., and Walther, A., *Evaluating derivatives: principles and techniques of algorithmic differentiation*, Siam, 2008.
- [25] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., “The Complex-Step Derivative Approximation,” *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262. doi:[10.1145/838250.838251](https://doi.org/10.1145/838250.838251), september.
- [26] Martins, J. R. R. A., and Hwang, J. T., “Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models,” *AIAA Journal*, Vol. 51, No. 11, 2013, pp. 2582–2599. doi:[10.2514/1.J052184](https://doi.org/10.2514/1.J052184).
- [27] Mader, C. A., Martins, J. R. R. A., Alonso, J. J., and van der Weide, E., “ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers,” *AIAA Journal*, Vol. 46, No. 4, 2008, pp. 863–873. doi:[10.2514/1.29123](https://doi.org/10.2514/1.29123).
- [28] Elham, A., and van Tooren, M. J., “Coupled adjoint aerostructural wing optimization using quasi-three-dimensional aerodynamic analysis,” *Structural and Multidisciplinary Optimization*, Vol. 54, No. 4, 2016, pp. 889–906.
- [29] Brezillon, J., and Dwight, R. P., “Applications of a Discrete Viscous Adjoint Method for Aerodynamic Shape Optimisation of 3D Configurations,” *CEAS Aeronautical Journal*, Vol. 3, No. 1, 2012, pp. 25–34.
- [30] Lyu, Z., Kenway, G. K., Paige, C., and Martins, J. R. R. A., “Automatic Differentiation Adjoint of the Reynolds-Averaged Navier–Stokes Equations with a Turbulence Model,” *21st AIAA Computational Fluid Dynamics Conference*, San Diego, CA, 2013. doi:[10.2514/6.2013-2581](https://doi.org/10.2514/6.2013-2581).
- [31] Burdette, D. A., “High-Fidelity Aerostructural Design Optimization of Transport Aircraft with Continuous Morphing Trailing Edge Technology,” Ph.D. thesis, University of Michigan, 2017.
- [32] Thompson, J. F., Soni, B. K., and Weatherill, N. P., *Handbook of grid generation*, CRC press, 1999.
- [33] Chawner, J. R., Michal, T. R., Slotnick, J. P., and Rumsey, C. L., “Summary of the 1st AIAA Geometry and Mesh Generation Workshop (GMGW-1) and Future Plans,” *2018 AIAA Aerospace Sciences Meeting*, 2018. doi:[10.2514/6.2018-0128](https://doi.org/10.2514/6.2018-0128), AIAA 2018-0128.

- [34] Mavriplis, D. J., Vassberg, J. C., Tinoco, E. N., Mani, M., Brodersen, O. P., Einfeld, B., Wahls, R. A., Morrison, J. H., Zickuhr, T., Levy, D., et al., “Grid quality and resolution issues from the drag prediction workshop series,” *Journal of Aircraft*, Vol. 46, No. 3, 2009, pp. 935–950.
- [35] Diskin, B., Thomas, J., Rumsey, C. L., and Schwöppe, A., “Grid convergence for turbulent flows,” *53rd AIAA Aerospace Sciences Meeting*, 2015. doi:[10.2514/6.2015-1746](https://doi.org/10.2514/6.2015-1746), AIAA 2015-1746.
- [36] Steger, J. L., Dougherty, F. C., and Benek, J. A., “A chimera grid scheme.[multiple overset body-conforming mesh system for finite difference adaptation to complex aircraft configurations],” *Advances in grid generation; Proceedings of the Applied Mechanics, Bioengineering, and Fluids Engineering Conference*, American Society of Mechanical Engineers, Houston, TX, 1983, pp. 59–69. A84-11576 02-64.
- [37] Chan, W. M., and Buning, P. G., “Surface grid generation methods for overset grids,” *Computers & fluids*, Vol. 24, No. 5, 1995, pp. 509–522. doi:[10.1016/0045-7930\(95\)00003-U](https://doi.org/10.1016/0045-7930(95)00003-U).
- [38] Chan, W. M., and Steger, J. L., “Enhancements of a three-dimensional hyperbolic grid generation scheme,” *Applied Mathematics and Computation*, Vol. 51, No. 2, 1992, pp. 181–205. doi:[10.1016/0096-3003\(92\)90073-A](https://doi.org/10.1016/0096-3003(92)90073-A).
- [39] Lee, Y., and Baeder, J. D., “Implicit hole cutting - a new approach to overset grid connectivity,” *16th AIAA Computational Fluid Dynamics Conference, Fluid Dynamics and Co-located Conferences*, 2003. doi:[10.2514/6.2003-4128](https://doi.org/10.2514/6.2003-4128), AIAA 2003-4128.
- [40] Chan, W., Gomez, R., Rogers, S., and Buning, P., “Best practices in overset grid generation,” *32nd AIAA Fluid Dynamics Conference and Exhibit*, 2002. doi:[10.2514/6.2002-3191](https://doi.org/10.2514/6.2002-3191), AIAA 2002-3191.
- [41] Liao, W., and Tsai, H. M., “Aerodynamic Shape Optimization on Overset Grids Using the Adjoint Method,” *International Journal for Numerical Methods in Fluids*, Vol. 62, No. 12, 2010, pp. 1332–1356. doi:[10.1002/flid.2070](https://doi.org/10.1002/flid.2070).
- [42] Lee, B. J., and Kim, C., “Aerodynamic redesign using discrete adjoint approach on overset mesh system,” *Journal of Aircraft*, Vol. 45, No. 5, 2008, pp. 1643–1653. doi:[10.2514/1.34112](https://doi.org/10.2514/1.34112).
- [43] Lee, B. J., Liou, M.-S., and Kim, C., “Optimizing a boundary-layer-ingestion offset inlet by discrete adjoint approach,” *AIAA Journal*, Vol. 48, No. 9, 2010, pp. 2008–2016. doi:[10.2514/1.J050222](https://doi.org/10.2514/1.J050222).
- [44] Kenway, G. K. W., Secco, N., Martins, J. R. R. A., Mishra, A., and Duraisamy, K., “An Efficient Parallel Overset Method for Aerodynamic Shape Optimization,” *Proceedings of the 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum*, Grapevine, TX, 2017. doi:[10.2514/6.2017-0357](https://doi.org/10.2514/6.2017-0357).

- [45] Li, Y., Paik, K.-J., Xing, T., and Carrica, P. M., “Dynamic overset CFD simulations of wind turbine aerodynamics,” *Renewable Energy*, Vol. 37, No. 1, 2012, pp. 285–298.
- [46] Carrica, P. M., Wilson, R. V., Noack, R. W., and Stern, F., “Ship motions using single-phase level set with dynamic overset grids,” *Computers & fluids*, Vol. 36, No. 9, 2007, pp. 1415–1433.
- [47] Carrica, P. M., Ismail, F., Hyman, M., Bhushan, S., and Stern, F., “Turn and zigzag maneuvers of a surface combatant using a URANS approach with dynamic overset grids,” *Journal of Marine Science and Technology*, Vol. 18, No. 2, 2013, pp. 166–181.
- [48] Samareh, J. A., “Survey of Shape Parameterization Techniques for High-Fidelity Multidisciplinary Shape Optimization,” *AIAA Journal*, Vol. 39, No. 5, 2001, pp. 877–884.
- [49] Bobrowski, K., Ferrer, E., Valero, E., and Barnewitz, H., “Aerodynamic Shape Optimization Using Geometry Surrogates and Adjoint Method,” *AIAA Journal*, Vol. 55, No. 10, 2017, pp. 3304–3317. doi:[10.2514/1.J055766](https://doi.org/10.2514/1.J055766).
- [50] Haimes, R., and Drela, M., “On the Construction of Aircraft Conceptual Geometry for High-fidelity Analysis and Design,” *50th AIAA Aerospace sciences meeting including the new horizons forum and aerospace exposition*, 2012. doi:[10.2514/6.2012-683](https://doi.org/10.2514/6.2012-683), AIAA 2012-0683.
- [51] Taylor, N. J., “Industrial Perspectives on Geometry Handling for Aerodynamics,” *22nd AIAA Computational Fluid Dynamics Conference*, 2015. doi:[10.2514/6.2015-3408](https://doi.org/10.2514/6.2015-3408), AIAA 2015-3408.
- [52] Hahn, A., “Vehicle Sketch Pad: a Parametric Geometry Modeler for Conceptual Aircraft Design,” *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010. doi:[10.2514/6.2010-657](https://doi.org/10.2514/6.2010-657), AIAA 2010-657.
- [53] Hwang, J. T., and Martins, J. R. R. A., “GeoMACH: Geometry-centric MDAO of Aircraft Configurations with High Fidelity,” *Proceedings of the 14th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Indianapolis, IN, 2012. doi:[10.2514/6.2012-5605](https://doi.org/10.2514/6.2012-5605), AIAA 2012-5605.
- [54] McDonald, R. A., “Interactive Reconstruction of 3D Models in the Open-VSP Parametric Geometry Tool,” *53rd AIAA Aerospace Sciences Meeting*, 2015. doi:[10.2514/6.2015-1014](https://doi.org/10.2514/6.2015-1014), AIAA 2015-1014.
- [55] Kenway, G. K., Kennedy, G. J., and Martins, J. R. R. A., “A CAD-Free Approach to High-Fidelity Aerostructural Optimization,” *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Fort Worth, TX, 2010. doi:[10.2514/6.2010-9231](https://doi.org/10.2514/6.2010-9231).

- [56] Koc, S., Kim, H.-J., and Nakahashi, K., “Aerodynamic design of complex configurations with junctions,” *Journal of aircraft*, Vol. 43, No. 6, 2006, pp. 1838–1844.
- [57] Kim, H.-J., and Nakahashi, K., “Surface mesh movement for aerodynamic design of body-installation junction,” *AIAA Journal*, Vol. 45, No. 5, 2007, pp. 1138–1142.
- [58] Xu, S., Timme, S., Mykhaskiv, O., and Müller, J.-D., “Wing-body Junction Optimisation with CAD-based Parametrisation Including a Moving Intersection,” *Aerospace Science and Technology*, Vol. 68, 2017, pp. 543–551.
- [59] Mykhaskiv, O., Mohanamuraly, P., Mueller, J.-D., Xu, S., and Timme, S., “CAD-based shape optimisation of the NASA CRM wing-body intersection using differentiated CAD-kernel,” *35th AIAA Applied Aerodynamics Conference*, 2017. doi:[10.2514/6.2017-4080](https://doi.org/10.2514/6.2017-4080), AIAA 2017-4080.
- [60] Walther, A., and Griewank, A., “Getting Started with ADOL-C.” *Combinatorial scientific computing*, 2009, pp. 181–202.
- [61] Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., “Scalable Parallel Approach for High-Fidelity Steady-State Aeroelastic Analysis and Derivative Computations,” *AIAA Journal*, Vol. 52, No. 5, 2014, pp. 935–951. doi:[10.2514/1.J052255](https://doi.org/10.2514/1.J052255).
- [62] Bonet, J., and Peraire, J., “An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems,” *International Journal for Numerical Methods in Engineering*, Vol. 31, No. 1, 1991, pp. 1–17. doi:[10.1002/nme.1620310102](https://doi.org/10.1002/nme.1620310102).
- [63] Möller, T., “A fast triangle-triangle intersection test,” *Journal of Graphics Tools*, Vol. 2, No. 2, 1997, pp. 25–30. doi:[10.1080/10867651.1997.10487472](https://doi.org/10.1080/10867651.1997.10487472).
- [64] Eriksson, L., “Generation of Boundary-conforming Grids Around Wing-body Configurations Using Transfinite Interpolation,” *AIAA Journal*, Vol. 20, No. 10, 1982, pp. 1313–1320.
- [65] Hascoët, L., and Pascual, V., “The Tapenade Automatic Differentiation tool: Principles, Model, and Specification,” *ACM Transactions On Mathematical Software*, Vol. 39, No. 3, 2013. URL <http://dx.doi.org/10.1145/2450153.2450158>.
- [66] Anderson, E., Bai, Z., Dongarra, J., Greenbaum, A., McKenney, A., Du Croz, J., Hammarling, S., Demmel, J., Bischof, C., and Sorensen, D., “LAPACK: A Portable Linear Algebra Library for High-performance Computers,” *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1990, pp. 2–11.
- [67] Giles, M. B., “Collected Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation,” *Advances in Automatic Differentiation*, Springer, 2008, pp. 35–44.

- [68] Hascoët, L., Vázquez, M., and Dervieux, A., “Automatic Differentiation for Optimum Design, Applied to Sonic Boom Reduction,” *Computational Science and Its Applications - ICCSA 2003*, 2003, pp. 976–976.
- [69] Vassberg, J. C., DeHaan, M. A., Rivers, S. M., and Wahls, R. A., “Development of a Common Research Model for Applied CFD Validation Studies,” *26th AIAA Applied Aerodynamics Conference, Guidance, Navigation, and Control and Co-located Conferences*, 2008. doi:[10.2514/6.2008-6919](https://doi.org/10.2514/6.2008-6919), AIAA 2008-6919.
- [70] Chan, W. M., “Best Practices on Overset Structured Mesh Generation for the High-Lift CRM Geometry,” *55th AIAA Aerospace Sciences Meeting*, 2017. doi:[10.2514/6.2017-0362](https://doi.org/10.2514/6.2017-0362), AIAA 2017-0362.
- [71] Lambe, A. B., and Martins, J. R. R. A., “Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes,” *Structural and Multidisciplinary Optimization*, Vol. 46, 2012, pp. 273–284. doi:[10.1007/s00158-012-0763-y](https://doi.org/10.1007/s00158-012-0763-y).
- [72] Kenway, G. K. W., and Martins, J. R. R. A., “Buffet Onset Constraint Formulation for Aerodynamic Shape Optimization,” *AIAA Journal*, Vol. 55, No. 6, 2017, pp. 1930–1947. doi:[10.2514/1.J055172](https://doi.org/10.2514/1.J055172).
- [73] Lyu, Z., and Martins, J. R. R. A., “Aerodynamic Design Optimization Studies of a Blended-Wing-Body Aircraft,” *Journal of Aircraft*, Vol. 51, No. 5, 2014, pp. 1604–1617. doi:[10.2514/1.C032491](https://doi.org/10.2514/1.C032491).
- [74] Kenway, G. K. W., and Martins, J. R. R. A., “Multipoint High-Fidelity Aerostructural Optimization of a Transport Aircraft Configuration,” *Journal of Aircraft*, Vol. 51, No. 1, 2014, pp. 144–160. doi:[10.2514/1.C032150](https://doi.org/10.2514/1.C032150).
- [75] Brooks, T. R., Kennedy, G. J., and Martins, J. R. R. A., “High-fidelity Multipoint Aerostructural Optimization of a High Aspect Ratio Tow-steered Composite Wing,” *Proceedings of the 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum*, Grapevine, TX, 2017. doi:[10.2514/6.2017-1350](https://doi.org/10.2514/6.2017-1350).
- [76] Vassberg, J., Tinoco, E., Mani, M., Brodersen, O., Eisfeld, B., Wahls, R., Morrison, J., Zickuhr, T., Laflin, K., and Mavriplis, D., “Summary of the third AIAA CFD drag prediction workshop,” *45th AIAA Aerospace Sciences Meeting and Exhibit*, 2007. doi:[10.2514/6.2007-260](https://doi.org/10.2514/6.2007-260), AIAA 2007-260.
- [77] Luke, E., Collins, E., and Blades, E., “A Fast Mesh Deformation Method Using Explicit Interpolation,” *Journal of Computational Physics*, Vol. 231, No. 2, 2012, pp. 586–601. doi:[10.1016/j.jcp.2011.09.021](https://doi.org/10.1016/j.jcp.2011.09.021).
- [78] van der Weide, E., Kalitzin, G., Schluter, J., and Alonso, J. J., “Unsteady Turbomachinery Computations Using Massively Parallel Platforms,” *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2006. AIAA 2006-0421.

- [79] Jameson, A., Schmidt, W., and Turkel, E., “Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge–Kutta Time Stepping Schemes,” Tech. Rep. AIAA 1981-1259, 1981.
- [80] Klopfer, G., Hung, C., Van der Wijngaart, R., and Onufer, J., “A Diagonalized Diagonal Dominant Alternating Direction Implicit (D3ADI) Scheme and Subiteration Correction,” *29th AIAA, Fluid Dynamics Conference*, 1998. doi:[10.2514/6.1998-2824](https://doi.org/10.2514/6.1998-2824), AIAA 1998-2824.
- [81] Jameson, A., “Analysis and Design of Numerical Schemes for Gas Dynamics, 1: Artificial Diffusion, Upwind Biasing, Limiters and Their Effect on Accuracy and Multigrid Convergence,” *International Journal of Computational Fluid Dynamics*, Vol. 4, No. 3-4, 1995, pp. 171–218. doi:[10.1080/10618569508904524](https://doi.org/10.1080/10618569508904524), URL <https://doi.org/10.1080/10618569508904524>.
- [82] Knoll, D. A., and Keyes, D. E., “Jacobian-free Newton–Krylov methods: a survey of approaches and applications,” *Journal of Computational Physics*, Vol. 193, No. 2, 2004, pp. 357–397.
- [83] Landmann, B., and Montagnac, M., “A highly automated parallel Chimera method for overset grids based on the implicit hole cutting technique,” *International Journal for Numerical Methods in Fluids*, Vol. 66, No. 6, 2011, pp. 778–804. doi:[10.1002/fld.2292](https://doi.org/10.1002/fld.2292).
- [84] Chan, W. M., “Enhancements to the Hybrid Mesh Approach to Surface Loads Integration on Overset Structured Grids,” *19th AIAA Computational Fluid Dynamics, Fluid Dynamics and Co-located Conferences*, 2009. doi:[10.2514/6.2009-3990](https://doi.org/10.2514/6.2009-3990), AIAA 2009-3990.
- [85] Tinoco, E. N., Brodersen, O., Keye, S., and Laffin, K., “Summary of Data from the Sixth AIAA CFD Drag Prediction Workshop: CRM Cases 2 to 5,” *55th AIAA Aerospace Sciences Meeting*, 2017. doi:[10.2514/6.2017-1208](https://doi.org/10.2514/6.2017-1208), AIAA 2017-1208.
- [86] Spalart, P. R., and Allmaras, S. R., “A one-equation turbulence model for aerodynamic flows,” *30th Aerospace Sciences Meeting and Exhibit*, 1992. doi:[10.2514/6.1992-439](https://doi.org/10.2514/6.1992-439), AIAA 1992-439.
- [87] Coder, J. G., Pulliam, T. H., Hue, D., Kenway, G. K., and Sclafani, A. J., “Contributions to the 6th AIAA CFD Drag Prediction Workshop Using Structured Grid Methods,” *AIAA SciTech Forum*, American Institute of Aeronautics and Astronautics, 2017. doi:[10.2514/6.2017-0960](https://doi.org/10.2514/6.2017-0960), URL <http://dx.doi.org/10.2514/6.2017-0960>.
- [88] Dacles-Mariani, J., Zilliac, G. G., Chow, J. S., and Bradshaw, P., “Numerical/experimental study of a wingtip vortex in the near field,” *AIAA Journal*, Vol. 33, No. 9, 1995, pp. 1561–1568. doi:[10.2514/3.12826](https://doi.org/10.2514/3.12826).

- [89] Spalart, P. R., “Strategies for turbulence modelling and simulations,” *International Journal of Heat and Fluid Flow*, Vol. 21, No. 3, 2000, pp. 252–263. doi:[10.1016/S0142-727X\(00\)00007-2](https://doi.org/10.1016/S0142-727X(00)00007-2).
- [90] Gill, P., Murray, W., and Saunders, M., “SNOPT: An SQP Algorithm for Large-scale Constrained Optimization,” *SIAM Journal on Optimization*, Vol. 12, No. 4, 2002, pp. 979–1006. doi:[10.1137/S1052623499350013](https://doi.org/10.1137/S1052623499350013), URL <http://dx.doi.org/10.1137/S1052623499350013>.
- [91] Saad, Y., and Schultz, M. H., “GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems,” *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, 1986, pp. 856–869. doi:[10.1137/0907058](https://doi.org/10.1137/0907058).
- [92] Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F., “Efficient Management of Parallelism in Object Oriented Numerical Software Libraries,” *Modern Software Tools in Scientific Computing*, edited by E. Arge, A. M. Bruaset, and H. P. Langtangen, Birkhäuser Press, 1997, pp. 163–202.
- [93] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., “PETSc Web page,” <http://www.mcs.anl.gov/petsc>, 2016. URL <http://www.mcs.anl.gov/petsc>.
- [94] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., “PETSc Users Manual,” Tech. Rep. ANL-95/11 - Revision 3.7, Argonne National Laboratory, 2016. URL <http://www.mcs.anl.gov/petsc>.
- [95] Brodersen, O., “Drag Prediction of Engine-airframe Interference Effects Using Unstructured Navier-Stokes Calculations,” *Journal of Aircraft*, Vol. 39, No. 6, 2002, pp. 927–935.
- [96] Laffin, K. R., Klausmeyer, S. M., Zickuhr, T., Vassberg, J. C., Wahls, R. A., Morrison, J. H., Brodersen, O. P., Rakowitz, M. E., Tinoco, E. N., and Godard, J.-L., “Data Summary from Second AIAA Computational Fluid Dynamics Drag Prediction Workshop,” *Journal of Aircraft*, Vol. 42, No. 5, 2005, pp. 1165–1178.
- [97] Lee, B., and Kim, C., “Aerodynamic Shape Optimization Using Discrete Adjoint Formulation Based on Overset Mesh Technique,” *European Conference on Computational Fluid Dynamics*, 2006.
- [98] Vassberg, J., Sclafani, A., and DeHaan, M., “A Wing-body Fairing Design for the DLR-F6 Model: a DPW-III Case Study,” *23rd AIAA Applied Aerodynamics Conference*, 2005. doi:[10.2514/6.2005-4730](https://doi.org/10.2514/6.2005-4730), AIAA 2005-4730.

- [99] Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., “RANS-based Aerodynamic Shape Optimization Investigations of the Common Research Model Wing,” *Proceedings of the AIAA Science and Technology Forum and Exposition (SciTech)*, National Harbor, MD, 2014. doi:[10.2514/6.2014-0567](https://doi.org/10.2514/6.2014-0567), aIAA 2014-0567.
- [100] Sederberg, T. W., and Parry, S. R., “Free-form Deformation of Solid Geometric Models,” *SIGGRAPH Comput. Graph.*, Vol. 20, No. 4, 1986, pp. 151–160. doi:[10.1145/15886.15903](https://doi.org/10.1145/15886.15903).
- [101] ANSYS ICEM CFD 14.0, ANSYS, Inc., 275 Technology Drive, Canonsburg, PA, October 2011.
- [102] Yamamoto, K., Tanaka, K., and Murayama, M., “Effect of a nonlinear constitutive relation for turbulence modeling on predicting flow separation at wing-body juncture of transonic commercial aircraft,” *30th AIAA Applied Aerodynamics Conference*, 2012. doi:[10.2514/6.2012-2895](https://doi.org/10.2514/6.2012-2895), AIAA 2012-2895.
- [103] Vassberg, J., Brodersen, O., Wahls, R., Zickuhr, T., Mavriplis, D., Tinoco, E., Mani, M., Levy, D., and Morrison, J., “Comparison of NTF Experimental Data with CFD Predictions from the Third AIAA CFD Drag Prediction Workshop,” *26th AIAA Applied Aerodynamics Conference*, 2008. doi:[10.2514/6.2008-6918](https://doi.org/10.2514/6.2008-6918), AIAA 2008-6918.
- [104] Kennedy, G. J., Kenway, G. K. W., and Martins, J. R. R. A., “High Aspect Ratio Wing Design: Optimal Aerostructural Tradeoffs for the Next Generation of Materials,” *Proceedings of the AIAA Science and Technology Forum and Exposition (SciTech)*, National Harbor, MD, 2014. doi:[10.2514/6.2014-0596](https://doi.org/10.2514/6.2014-0596).
- [105] Gur, O., Bhatia, M., Schetz, J. A., Mason, W. H., Kapania, R. K., and Mavris, D. N., “Design Optimization of a Truss-Braced-Wing Transonic Transport Aircraft,” *Journal of Aircraft*, Vol. 47, No. 6, 2010. doi:[10.2514/1.47546](https://doi.org/10.2514/1.47546).
- [106] Chakraborty, I., Gross, J. R., Nam, T., Perullo, C., and Mavris, D. N., “Analysis of the Effect of Cruise Speed on Fuel Efficiency and Cost for a Truss-Braced Wing Concept,” *14th AIAA Aviation Technology, Integration, and Operations Conference*, 2014. doi:[10.2514/6.2014-2424](https://doi.org/10.2514/6.2014-2424), AIAA 2014-2424.
- [107] Bradley, M. K., Droney, C. K., and Allen, T. J., “Subsonic Ultra Green Aircraft Research. Phase II-Volume I; Truss Braced Wing Design Exploration,” *NASA Technical Report*, 2015. NASA/CR-2015-218704/VOL1.
- [108] Carrier, G., Atinault, O., Dequand, S., Hantrais-Gervois, J., Liauzun, C., Paluch, B., Rodde, A., and Toussaint, C., “Investigation of a strut-braced wing configuration for future commercial transport,” *28th Congress of the International Council of the Aeronautical Sciences (Brisbane, Australia)*, 2012.
- [109] Turriziani, R., Lovell, W., Martin, G., Price, J., Swanson, E., and Washburn, G., “Preliminary design characteristics of a subsonic business jet concept employing an aspect ratio 25 strut braced wing,” *NASA Technical Report*, 1980. NASA-CR-159361.

- [110] Grasmeyer, J., “Multidisciplinary design optimization of a transonic strut-braced wing aircraft,” *37th AIAA Aerospace Sciences Meeting and Exhibit*, 1999, pp. 11–14. doi:[10.2514/6.1999-10](https://doi.org/10.2514/6.1999-10).
- [111] Gundlach, J. F., Tétrault, P.-A., Gern, F. H., Nagshineh-Pour, A. H., Ko, A., Schetz, J. A., Mason, W. H., Kapania, R. K., et al., “Conceptual design studies of a strut-braced wing transonic transport,” *Journal of aircraft*, Vol. 37, No. 6, 2000, pp. 976–983. doi:[10.2514/2.2724](https://doi.org/10.2514/2.2724).
- [112] Grasmeyer, J., and Mason, W., “A discrete vortex method for calculating the minimum induced drag and optimum load distribution for aircraft configurations with noncoplanar surfaces,” *VPIAOE-242, AOE Department, VPI & SU, Blacksburg, Virginia*, Vol. 24061, 1997.
- [113] Tétrault, P.-A., Schetz, J. A., and Grossman, B., “Numerical Prediction of Interference Drag of Strut-surface Intersection in Transonic Flow,” *AIAA Journal*, Vol. 39, No. 5, 2001, pp. 857–864. doi:[10.2514/2.1389](https://doi.org/10.2514/2.1389).
- [114] Duggirala, R. K., Roy, C. J., and Schetz, J. A., “Analysis of Interference Drag for Strut-strut Interaction in Transonic Flow,” *AIAA Journal*, Vol. 49, No. 3, 2011, pp. 449–462. doi:[10.2514/1.45703](https://doi.org/10.2514/1.45703).
- [115] Gipson, L., “NASA Aeronautics Budget Proposes Return of X-Planes,” <http://www.nasa.gov/feature/nasa-aeronautics-budget-proposes-return-of-x-planes>, 2016-02-18. Accessed: 2016-04-22.
- [116] Gagnon, H., and Zingg, D. W., “High-fidelity Aerodynamic Shape Optimization of Unconventional Aircraft through Axial Deformation,” *52nd Aerospace Sciences Meeting*, 2014. doi:[10.2514/6.2014-0908](https://doi.org/10.2514/6.2014-0908), AIAA 2014-0908.
- [117] Gagnon, H., and Zingg, D. W., “Euler-equation-based drag minimization of unconventional aircraft configurations,” *Journal of Aircraft*, Vol. 53, No. 5, 2016, pp. 1361–1371. doi:[10.2514/1.C033591](https://doi.org/10.2514/1.C033591).
- [118] Secco, N. R., Jasa, J. P., Kenway, G. K. W., and Martins, J. R. R. A., “Component-based Geometry Manipulation for Aerodynamic Shape Optimization with Overset Meshes,” *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, American Institute of Aeronautics and Astronautics, 2017. doi:[10.2514/6.2017-3327](https://doi.org/10.2514/6.2017-3327), AIAA 2017-3327.
- [119] Bieler, H., Bier, N., Bugada, G., Periaux, J., Redondo, D., Guttila, S., and Pons, J., “A common platform for validation of aircraft drag reduction technologies,” , 2017. URL <http://congress.cimne.com/padri-2017/frontal/default.asp>.
- [120] Hwang, J. T., Kenway, G. K. W., and Martins, J. R. R. A., “Geometry and Structural Modeling for High-Fidelity Aircraft Conceptual Design Optimization,” *Proceedings*

of the 15th AIAA Multidisciplinary Analysis and Optimization Conference, Atlanta, GA, 2014. doi:[10.2514/6.2014-2041](https://doi.org/10.2514/6.2014-2041), aIAA 2014-2041.

- [121] Meadows, N. A., Schetz, J. A., Kapania, R. K., Bhatia, M., and Seber, G., “Multidisciplinary design optimization of medium-range transonic truss-braced wing transport aircraft,” *Journal of Aircraft*, Vol. 49, No. 6, 2012, pp. 1844–1856. doi:[10.2514/1.C031695](https://doi.org/10.2514/1.C031695).
- [122] Ko, A., Mason, W., and Grossman, B., “Transonic aerodynamics of a wing/pylon/strut juncture,” *21st AIAA Applied Aerodynamics Conference*, 2003. doi:[10.2514/6.2003-4062](https://doi.org/10.2514/6.2003-4062), AIAA 2003-4062.
- [123] Secco, N. R., Jasa, J. P., Kenway, G. K. W., and Martins, J. R. R. A., “Component-based Geometry Manipulation for Aerodynamic Shape Optimization with Overset Meshes,” *AIAA Journal*, 2018. (Accepted 04-2018).
- [124] Secco, N. R., and Martins, J. R. R. A., “RANS-based Aerodynamic Shape Optimization of a Strut-braced Wing with Overset Meshes,” *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, American Institute of Aeronautics and Astronautics, Kissimmee, FL, 2018. AIAA 2018–0413.
- [125] Secco, N. R., and Martins, J. R. R. A., “RANS-based Aerodynamic Shape Optimization of a Strut-braced Wing with Overset Meshes,” *Journal of Aircraft*, 2018. (Accepted 06-2018).