Efficient algorithms for future aircraft design: contributions to aerodynamic shape optimization

Doctor of Philosophy

Jason Edward Hicken

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Aerospace Science and Engineering
University of Toronto

Abstract

Efficient algorithms for future aircraft design: contributions to aerodynamic shape optimization

Jason Edward Hicken

<jason.hicken@gmail.com>

Doctor of Philosophy

Graduate Department of Aerospace Science and Engineering

University of Toronto

2009

Advances in numerical optimization have raised the possibility that efficient and novel aircraft configurations may be "discovered" by an algorithm. To begin exploring this possibility, a fast and robust set of tools for aerodynamic shape optimization is developed. Parameterization and mesh-movement are integrated to accommodate large changes in the geometry. This integrated approach uses a coarse B-spline control grid to represent the geometry and move the computational mesh; consequently, the mesh-movement algorithm is two to three orders faster than a node-based linear elasticity approach, without compromising mesh quality. Aerodynamic analysis is performed using a flow solver for the Euler equations. The governing equations are discretized using summation-by-parts finite-difference operators and simultaneous approximation terms, which permit C^0 mesh continuity at block interfaces. The discretization results in a set of nonlinear algebraic equations, which are solved using an efficient parallel Newton-Krylov-Schur strategy. A gradient-based optimization algorithm is adopted. The gradient is evaluated using adjoint variables for the flow and mesh equations in a sequential approach. The flow adjoint equations are solved using a novel variant of the Krylov solver GCROT. This variant of GCROT is flexible to take advantage of non-stationary preconditioners and is shown to outperform restarted flexible GMRES. The aerodynamic optimizer is applied to several studies of induced-drag minimiza-

tion. An elliptical lift distribution is recovered by varying spanwise twist, thereby validating the algorithm. Planform optimization based on the Euler equations produces a nonelliptical lift distribution, in contrast with the predictions of lifting-line theory. A study of spanwise vertical shape optimization confirms that a winglet-up configuration is more efficient than a winglet-down configuration. A split-tip geometry is used to explore nonlinear wake-wing interactions: the optimized split-tip demonstrates a significant reduction in induced drag relative to a single-tip wing. Finally, the optimal spanwise loading for a box-wing configuration is investigated.

Acknowledgements

I remember sitting down with Dr. David Zingg, about five years ago, to discuss possible doctoral projects. During this meeting, he asked the questions that set the ground for this thesis: what is the optimal aerodynamic shape for an aircraft, and can a numerical algorithm find it? Beyond motivating this work, Dr. Zingg has been an exceptional supervisor and mentor. It has been an honour and pleasure to work with him.

I thank my doctoral committee members, Drs. Joaquim Martins and Jorn Hansen, for their insights and questions. Dr. Hansen provided an important outside perspective that improved the quality and rigour of the thesis. Dr. Martins' knowledge of, and enthusiasm for, aircraft design and optimization continues to inspire me.

I gratefully acknowledge Drs. Juan Alonso, Markus Bussmann, and Philippe Lavoie for their involvement in my final oral examination, and Dr. Alonso, in particular, for writing the appraisal letter.

My thanks to Clara, Gail, Joan, Nora, Peter, and Rosanna for making my stay at UTIAS a smooth and enjoyable one. Their efforts kept me free to focus on research rather than the bureaucratic intricacies of the University of Toronto.

My colleagues at UTIAS, both past and present, have my gratitude and appreciation for providing a world-class academic environment. Many thanks to my fellow students in the CFD lab for sharing their knowledge and expertise. In particular, Scott Northrup, for administrating the network and answering innumerable computer questions; James M^{c}Donald, for teaching me to be watchful of my generalizations; Mohammad Tabesh, for providing perspective and humour; Jonathan Driver, for his contagious scientific curiosity; and Graeme Kennedy, for his insights into solid mechanics and finite-element methods.

I am grateful for the patience of my family and extended family, who continue to support me, despite the often esoteric nature of my work.

Meaghan, thank you for your encouragement, love, and friendship. *You continue to be my beautiful reminder of what is important in life.*

Jason Edward Hicken

University of Toronto Institute for Aerospace Studies
April, 2009

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

Alphanumeric

$\mathcal{L}$	Lagrangian of a constrained optimization problem
$\mathcal{N}_i^{(p)}$	pth-order B-spline basis function (fourth if no superscript)
$\mathcal{J}$	objective function
A	inviscid flux Jacobian matrix
a	speed of sound
b	wing span
D, C_D	drag and coefficient of drag
E	Young's modulus
e	span efficiency factor
J	Jacobian of the mapping between $\boldsymbol{\xi}$ and $\mathbf{x}$
L, C_L	lift and coefficient of lift
p	pressure
q_∞	free-stream dynamic pressure
$R_d^{(n)}$	norm of Euler residual at Newton iteration n
S	reference area
A	Jacobian matrix of Euler residual, or a generic matrix
A_1	first-order Jacobian matrix
H	matrix norm for summation-by-parts operator
K	mesh-movement stiffness matrix
L_i, U_i	lower and upper matrices in an incomplete matrix factorization
Q	difference matrix for summation-by-parts operator
S, S_i	global and local Schur-complement matrices
$\mathsf{S}^{\pm}$	matrix operator for isolating boundary variables
T	diagonal matrix of inverse time steps
Z	non-linear dissipation matrix operator
$\tilde{\mathsf{A}}$	approximate Jacobian matrix

$\boldsymbol{\mathcal{M}}$	B-spline mesh-movement residual
$\boldsymbol{\mathcal{R}}$	discretized Euler equation residual (vector)
$\hat{\mathbf{F}}_i$	vector of inviscid fluxes in direction ξ_i
$\hat{\mathbf{Q}}$	Jacobian-scaled conservative variables at a point
$\mathbf{F}_i$	vector of inviscid fluxes at a point, in direction x_i
$\mathbf{Q}$	vector of conservative variables at a point
$\boldsymbol{\lambda}^{(i)}$	vector of mesh-adjoint variables for increment i
$\mathbf{b}$	vector of B-spline control-point coordinates, or a generic RHS vector
$\mathbf{c}$	vector of constraint equations for the intermediate variables
$\mathbf{f}$	vector of inviscid fluxes on a mesh
$\mathbf{q}$	vector of flow variables on a mesh
$\mathbf{v}$	vector of design variables
$\mathbf{w}$	vector of intermediate variables

Greek

α	angle of attack
$\boldsymbol{\Lambda}$	block vector of Lagrange multipliers
$\boldsymbol{\lambda}$	vector of Lagrange multipliers
$\boldsymbol{\psi}$	vector of flow adjoint variables on a mesh
$\boldsymbol{\Sigma}$	vector of SAT penalties
η	inexact-Newton forcing parameter (tolerance)
κ_2, κ_4	second- and fourth-difference dissipation coefficients
ν	Poisson's ratio
ξ, η, ζ	computational-space coordinates

Abbreviations

BCR	Block Compressed Row (matrix storage scheme)
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BFILU	Block Fill Incomplete Lower Upper (factorization)
BILU	Block Incomplete Lower Upper (factorization)
CAD	Computer Aided Design

CFL	Courant-Friedrichs-Lewy number
CSR	Compressed Sparse Row (matrix storage scheme)
DG	Discontinuous Galerkin
FD	Finite Difference
FFD	Free-Form-Deformation
FGMRES	Flexible Generalized Minimal RESidual
FV	Finite Volume
GA	Genetic Algorithm
GCROT	Generalized Conjugate Residual with inner-Orthogonalization, Truncated
GMRES	Generalized Minimal RESidual
ILU	Incomplete Lower Upper (factorization)
NURBS	NonUniform Rational B-Spline
RANS	Reynolds-Averaged Navier-Stokes
RBF	Radial Basis Function
RHS	Right Hand Side
SA	Simulated Annealing
SAT	Simultaneous-Approximation Term
SBP	Summation-By-Parts
SOA	Soft-Object Animation
SQP	Sequential Quadratic Programming

Chapter 1

Introduction

1.1 Oil, Carbon, and the Future of Air Transportation

This thesis is concerned with the efficiency of aircraft. In this regard, it is similar to many previous theses in aerospace engineering; however, this thesis is motivated by two global challenges that demand urgent action at the beginning of the twenty-first century: peak oil and global warming.

Peak oil refers to the point at which global demand for oil exceeds its recovery [31]. The idea is based on the work of M. King Hubert, who, in 1956, correctly predicted that oil extraction in the continental United States would peak around 1970 [77]. The consequences of the 1970's oil crisis were significant but temporary, since oil fields outside the United States were able to fill the supply void. In contrast, the worldwide peak in oil production will be permanent. Such a global peak in oil production is predicted to occur within the first quarter of this century [31, 50].

Global warming is an increase in average global temperature. The warming is caused by greenhouse gases, particularly carbon dioxide (CO_2), which are released into the atmosphere when fossil fuels are burned. The possibility that human actions could warm the planet was debated throughout much of the twentieth century [183]. Presently, the mounting evidence for anthropogenic climate change is compelling; indeed, the international scientific consensus is that humanity is very likely (90–99%) causing global warming [79]. This has led several countries to impose carbon taxes or carbon-based trading systems.

From an economics perspective, the impact of both peak oil and CO_2 reduction strategies will be the same: an increase in the effective cost of conventional fuels. How will civil aviation be affected by increased fuel prices? Certainly, commercial air transportation is important, but the relevant question is "at what price will individuals and businesses consider alternatives?" For example, most people can vacation close to home, trains can be used

for continental travel, and international business meetings can be conducted with video-conferencing. Given such alternatives, will air transportation, as it now exists, survive a dramatic increase in fuel prices?

1.2 An Argument for Unconventional Aircraft

If civil aviation is to continue growing, or even maintain its present level, the industry must address the challenges posed by global warming and peak oil. One solution is provided by unconventional aircraft configurations that significantly improve fuel efficiency and reduce greenhouse gas emissions: this is the solution that I will ultimately advance. However, unconventional designs may be unnecessary if an economical alternative to kerosene is developed. In light of this, I begin this section with a review of research into alternative fuels for aviation.

1.2.1 Alternative Fuels

Kerosene possesses many favourable characteristics as an aviation fuel, including its freezing point, clean combustion, energy density, and thermal stability [39]. In response to increasing fuel costs, many economists argue that market forces will introduce an alternative aircraft fuel; however, such an alternative must approximate the characteristics of kerosene, or risk introducing modifications to the existing aircraft fleet. Here, I consider the three most commonly cited aviation fuel alternatives: bio-fuels, Fischer-Tropsch-based fuels, and liquid hydrogen.

Bio-fuels are attractive, at least superficially, because any CO_2 that they release is re-absorbed in the feedstock (crop) used to produce the fuel. Unfortunately, this reasoning ignores the energy used to refine the fuel. Furthermore, we must consider how the arable land, necessary for growing the feedstock, is obtained. If this land comes from existing farms, conventional crops will be displaced and the price of food will increase, raising difficult ethical questions. If new land is cleared to grow the feedstock, the fuel must account for the CO_2 released by removing the existing biomass from that land, e.g. clear cutting rainforest to grow the feedstock. Even if we ignore these production problems, there is presently no bio-fuel that can be used as a drop-in replacement for kerosene. Ethanol and methanol

have unsatisfactory energy densities and flash points [156], and bio-diesel must be mixed with kerosene to maintain a suitable cloud point [14, 156]. In the near-term (5–10 years), a bio-diesel-kerosene mix offers limited fuel price stability and CO_2 reductions.

The Fischer-Tropsch (FT) process can be used to produce synthetic kerosene from a variety of feedstocks, including coal, natural gas, and biomass. If coal or natural gas are used to produce FT kerosene, fossil carbon is still released, so these feedstocks are not viable solutions for global climate change. This leaves biomass feedstocks, which have the same issues for FT synthetic kerosene as they do for bio-fuels. A possible exception would be FT kerosene produced from algae-based vegetable oils, since algae can produce at least 15 times more oil per area than land-based plants [14]. While algae-based kerosene is promising, the FT process remains very expensive, and many years will be needed to create the necessary infrastructure for production; therefore, synthetic FT kerosene may provide a partial solution, but only in the medium to long term (10+ years).

Hydrogen appears to be an excellent candidate from the perspective of climate change, since carbon emissions are eliminated during combustion of this fuel; however, like the other options, the production of hydrogen fuel cannot be excluded from the analysis. Hydrogen can be produced from biomass and electrolysis of water. When electrolysis is used, the electricity must be supplied by nuclear or renewable energy sources. In either case, the cost of production is estimated to be as high as synthetic kerosene [156]. In addition, compared with the other options, hydrogen is the least compatible with present aircraft designs. Indeed, for the same amount of energy, liquid hydrogen occupies 4 times the volume of kerosene [156]. While the volume penalty is somewhat offset by a weight reduction — liquid hydrogen weighs 2.6 times less than kerosene — the liquid hydrogen storage vessels will add weight and complexity to the fuel system [163]. For these reasons, liquid hydrogen should be considered a long term solution only [156].

Finally, the aviation industry could continue using conventional kerosene, with alternative fuels directed toward surface-based modes of transportation, which are much more flexible with respect to fuel types. This type of fuel management would likely require governments to subsidize the civil aviation industry, since aircraft presently compete with surface-based vehicles for the same fossil fuel. Moreover, the continued use of conventional kerosene addresses only peak oil and ignores the greenhouse gas emissions produced by the industry.

1.2.2 Unconventional Configurations and Operations

Based on the above reasoning, two outcomes seem likely. First, if conventional fuels are available for aviation, for example through government subsidies, efficiency improvements will still be necessary to reduce greenhouse gas emissions. Second, if alternative fuels are adopted by the industry, aircraft performance will again need to improve to offset the increase in fuel cost. In both scenarios the conclusion is the same: air transportation must be made significantly more efficient.

However, modern aircraft are already remarkably efficient. A person traveling from Seattle to New York on a 737-900 aircraft uses less fuel than one person traveling the same distance in a gas-electric hybrid car[1]. Indeed, the modern aircraft is so highly optimized, that the improvements demanded by peak oil and climate change are likely not possible using the conventional configuration [52]. This has motivated the investigation of unconventional aircraft and operations.

Among unconventional configurations, the blended-wing body (BWB) is a promising alternative to tube-and-wing aircraft. The BWB concept has been estimated to reduce fuel burn by 27% per seat compared with a conventional composite aircraft [101]. Additional benefits of the design include reduced noise, reduced manufacturing part count (reduced cost), and a high commonality in a BWB-family concept [101]. Although issues with the BWB remain, they appear solvable.

The BWB achieves its aerodynamic efficiency through substantially reduced viscous drag. Another potential route to increase efficiency is by reducing induced drag. Induced drag, or drag due to lift, is an inviscid effect typically comprising 40% of the total drag of a conventional aircraft in cruise flight [97]. To achieve substantial reductions in induced drag, it is necessary to increase the span or consider nonplanar systems. Roughly, a nonplanar system is one in which the wake leaving the wing has vertical structure. Winglets are one example of a nonplanar system.

The nonplanar system with the smallest induced drag, for a given wing span and height, is a closed rectangular box [178]. For example, with a height-to-span ratio of 0.2, this box-wing configuration achieves a 46% reduction in induced drag relative to an elliptically loaded

[1] Assuming that the aircraft has a 100% load factor, burns 2.4 gallons per nautical mile, and the hybrid is a Toyota Prius.

planar wing [97]. The absence of the box-wing configuration among modern aircraft designs reflects the impact of structures and viscous effects. In other words, non-planar concepts are effective at reducing induced drag, but may increase structural weight or viscous drag.

Rather than focusing on configuration changes, we can also consider active flow control. The idea of active control is to expend a small amount of energy to achieve a large gain in performance, such that the total amount of energy used is lower. One example is boundary layer suction, which is often proposed to improve aircraft efficiency; see, for example, [159]. While boundary layer suction shows promise [52], many questions remain regarding its implementation. In particular, light-weight suction systems must be developed and mitigation of ice/dirt accretion on the suction surface must be addressed [159].

Finally, improvements in aircraft efficiency can also be achieved through operational changes. Operational changes can include reducing loiter time when landing, breaking long flights into a series of short flights [51], and flying in formation [97]. An advantage of operational changes is that they can be applied to conventional aircraft. To illustrate, flight tests have shown that two aircraft flying in formation can reduce fuel burn by 18% [176]. In the case of formation flight, one challenge would be coordinating schedules between competing airlines. While operational changes may delay the problems posed by peak oil and climate change, they are not universally applicable and their benefits are ultimately limited by the performance of the aircraft.

1.3 The Evolving Role of Numerical Optimization

Clearly, a number of unconventional configurations and ideas are being pursued to improve aircraft efficiency. In general, these concepts are not new: they are old ideas that have evolved using a mix of analysis and *ad hoc* human ingenuity. Over the past three decades, numerical optimization has been introduced into this mix. Hicks, Murman, and Vanderplaats [71] were among the first to investigate numerical optimization in the context of aerodynamic design. They developed an algorithm based on the method of feasible directions together with the transonic, small-disturbance equations, and their preliminary results hinted at the potential of numerical optimization.

In their algorithm, Hicks *et al.* [71] calculated the gradient using finite-difference approximations. Resorting to finite differences limits the number of design variables that can be

considered in practice, since each component of the gradient requires at least one flow analysis. By introducing adjoint variables, Pironneau [135] showed that, remarkably, the gradient can be calculated at a cost (virtually) independent of the number of design variables. This adjoint-based gradient calculation was later pioneered by Jameson [81] within the field of computational aerodynamics.

Adjoint-based methods were further popularized by the observation that the adjoint variables could be introduced after discretizing the governing equations, rather than before [10, 43]. The resulting discrete-adjoint approach is more straightforward conceptually. In addition, the discrete-adjoint variables produce gradients that correspond exactly with the discrete objective, making it possible to use state-of-the-art gradient-based optimization algorithms.

The adjoint approach has been responsible, in part, for a significant evolution in the field of aerodynamic shape optimization. I mention only a few examples from the literature here. Jameson and Reuther [84] developed an Euler-based adjoint algorithm, which was subsequently applied by Reuther *et al.* [143] to the optimization of complete aircraft configurations; see also [144, 145]. Jameson *et al.* [83] derived the continuous adjoint equations for the compressible Navier-Stokes equations, with the viscosity, heat conduction, and turbulent viscosity treated as constants. Anderson and Bonhaus [5] addressed the variation of the turbulent viscosity by including a discrete-adjoint variable for the (Spalart-Allmaras) turbulence model. Driver and Zingg [33] incorporated laminar-turbulent transition into their gradient-based optimizer and were able to recover a high-lift airfoil with characteristics similar to those of the Liebeck LNV109A airfoil [102].

These examples illustrate that numerical optimization can be used to tackle complex problems in aerodynamic design. However, in the context of unconventional configurations, the role of high-fidelity numerical optimization has been limited to refining a particular design. Rather than using high-fidelity optimization to merely refine a design, suppose we apply the tool to the initial design problem itself. In other words, can high-fidelity optimization be used to design an aircraft? This thesis represents the first steps in a research programme directed toward this question.

1.4 Thesis Outline and Summary of Objectives

Before proceeding with a description of the thesis, I will summarize the motivation and arguments presented above. Peak oil and anthropogenic climate change will lead to substantial increases in fuel costs over the coming decades. Alternative aviation fuels will be expensive to produce and availability will be limited by infrastructure; consequently, the aviation industry cannot depend on alternative fuels to keep the cost of flying at present levels. Operational changes, while important, are also insufficient. The efficiency improvements demanded by climate change and peak oil require that we develop unconventional configurations. Numerical optimization will play an important role in the design of these configurations. To what extent can numerical optimization be used to automatically design these novel aircraft?

Using numerical optimization to design the whole aircraft is an exciting prospect, but there are significant challenges. This is a long term research programme, and the required optimization algorithm will involve multiple disciplines, many of which need complex high-fidelity analysis codes. Therefore, this thesis focuses on the aerodynamic discipline in isolation, with a view to incorporating the resulting module with other disciplines in subsequent work.

In addition, I assume that the aerodynamics are governed by the Euler equations. I have already remarked that viscous effects and structures cannot be ignored in aircraft design; however, the validity and efficiency of the final optimizer depends on the validity and efficiency of its components, so the importance of an Euler-based optimizer should not be underestimated.

Beyond the flow analysis, the aerodynamic optimizer requires an efficient geometry parameterization and a fast and robust mesh movement algorithm. An efficient parameterization is needed to capture a large range of designs using few design variables. A robust mesh-movement algorithm is important, because we want the optimizer to consider large variations in the design.

The chosen optimizer uses a gradient-based algorithm. Gradient-based optimizers are known to find local optima more efficiently than alternative algorithms, provided the gradient is evaluated in a timely and accurate manner. These conditions can easily be satisfied if the gradient is computed using adjoint variables.

I use the Euler-based optimizer to study optimal inviscid shapes. While the resulting

shapes are not practical aircraft designs, they provide interesting and important insights into the physics of induced drag. Moreover, they help to validate the Euler-based optimizer.

The objectives of this thesis, informed by the preceding remarks, are summarized below.

- Make mesh-movement algorithms more efficient by reducing the number of degrees of freedom associated with the mesh. Demonstrate this idea using B-spline volumes to parameterize multi-block grids and linear elasticity to move the B-spline control points. Take advantage of the B-spline volume parameterization of the blocks to parameterize the surface geometry.

- Develop an efficient solver for the Euler equations using a finite-difference discretization and a parallel Newton-Krylov solution algorithm. In particular, use summation-by-parts (SBP) finite-difference operators and simultaneous-approximation terms (SATs), which are well-suited to complex multi-block grids, and develop a scalable preconditioner that takes advantage of the SBP-SAT discretization.

- Develop a Krylov-iterative solver for the flow-adjoint equations that is as efficient as restarted GMRES but more robust, and uses less memory than full GMRES.

- Demonstrate that the Euler-based optimizer can be used to study induced drag. In particular, investigate

  - twist optimization as a validation of Euler-based optimization algorithms,
  - the influence of edge separation on the optimization of planform shape,
  - the optimal spanwise vertical shape of a wing,
  - whether an optimal split-tip configuration can achieve significant performance gains over an optimal single-tip configuration, and
  - the optimal loading for the horizontal sections of a box-wing.

I have organized the thesis as follows. Chapters 2 through 4 describe the components of the optimizer in the sequence they are encountered in a typical iteration: geometry update and mesh movement (Chapter 2), inviscid flow analysis (Chapter 3), and gradient evaluation (Chapter 4). The optimizer is validated and applied to inviscid shape optimization in Chapter 5. Finally, I present conclusions, contributions, and recommendations in Chapter 6.

Chapter 2

Geometric Parameterization and Mesh Movement

Geometric parameterization is a compromise between the number of design variables and the ability to represent arbitrary shapes. For example, aerodynamic shapes are infinite dimensional objects. In contrast, numerical optimization algorithms operate on a discrete set of variables, and computational time increases with the size of this discrete set. A good parameterization approximates arbitrary aerodynamic shapes well using as few variables as necessary. In this chapter, I present the parameterization that I have adopted for aerodynamic shapes, and argue that it meets the criteria of a good parameterization.

I also present the mesh-movement algorithm that I have implemented. Mesh generation and movement are usually considered distinct steps from parameterization. I believe this is an unfortunate consequence of computer aided design (CAD) systems developing independently from meshing software. This independent evolution was no doubt encouraged by the individual steps of the conventional design process: define CAD geometry; construct mesh; perform analysis; use analysis to update geometry; iterate. With the advent of numerical optimization, it has become necessary to integrate the disparate software used for each step. The problems associated with integrating CAD and analysis software were acknowledged at the "Forward Looking Session" of the Tenth Society of Industrial and Applied Mathematics Conference on Geometric Design and Computing [142].

My thesis does not require exact representation of CAD surfaces; this permits greater freedom in pursuing a unified and integrated approach to parameterization and mesh generation/movement. There are many potential advantages with a unified approach. In the context of aerodynamic optimization, I will argue that a significant advantage is robust and efficient mesh movement.

2.1 Review of Parameterizations and Mesh-Movement Algorithms

2.1.1 Review of Parameterizations

Some authors consider a very narrow definition of aircraft geometries, and, consequently, construct parameterizations that limit the design space. Such an approach is appropriate when making incremental improvements to existing designs. In contrast, this thesis represents the beginning of a research programme to investigate optimal aerodynamic shapes with as few assumptions as possible; hence, the space of admissible geometries must be as broad as possible.

In the following review, I focus on parameterizations that have been successfully applied to, or show promise for, aerodynamic shape optimization. I group the parameterizations into categories analogous to those used by Samareh in his survey of geometry parameterizations for multidisciplinary shape optimization [155]. I consider the advantages and disadvantages of each category when applied to the problem of interest.

Analytical Perturbation

In their early work on numerical optimization, Hicks and Henne [70] parameterized airfoils and wings by adding shape functions to a baseline geometry. The Hicks-Henne shape functions are of the form [143]

$$h(x) = \left[\sin\left(\pi x^{\frac{\ln(0.5)}{\ln(t_1)}}\right)\right]^{t_2}, \qquad 0 \leq x \leq 1.$$

The parameter t_1 controls the position of the "sine bump," while t_2 controls its width. In general, these shape functions do not form a basis for the space of functions on $[0, 1]$ that vanish at the end points [84], although this problem can easily be overcome by using a different set of functions.

Reuther *et al.* successfully used Hicks-Henne type shape functions in the redesign of a wing for a transonic business jet [143]. They found that Hicks-Henne functions produce well-behaved (non-wavy) geometries, in contrast to B-spline surfaces, when a large number of control points are needed; however, this reported behaviour may be related to the Hicks-Henne functions failing to form a basis.

With a careful choice of shape functions, it may be possible to use analytical perturbations to morph between arbitrary initial and final geometries; however, representing arbitrary perturbations is essentially the same as representing arbitrary geometries, and equally as difficult.

The CST Method

Kulfan [98] developed the class function/shape function transformation — the CST method — specifically for aerodynamic shape optimization. Two dimensional shapes are defined by a class function and a shape function: the class function controls the general shape, while the shape function adds perturbations to obtain a specific shape. For example, the general airfoil class function can be modified by a particular shape function to obtain an approximation of the NACA0012 airfoil geometry. For three dimensional shapes, a distribution of two dimensional shapes is specified resulting in a "scalar" lofting technique.

While the CST method can capture a large class of aerodynamic shapes, it appears to ignore parametric surfaces as a viable alternative. Consequently, the method described by Kulfan cannot parameterize complicated implicit surfaces, i.e. level sets. For example, a planar wing cannot be morphed continuously into a C-wing.

Discrete Surface Points

Jameson [84, 82, 83] and Mohammadi [115] parameterize their geometries using the individual grid points on the airfoil or wing. The grid points can represent any geometry within the resolution of the computational mesh spacing; however, in practice, the gradient with respect to the grid points must be filtered to maintain smooth geometries. An implicit Helmholtz filter is often used to smooth the gradient [84]:

$$\bar{G} - \boldsymbol{\nabla} \cdot (\alpha^2 \boldsymbol{\nabla} \bar{G}) = G,$$

where G is the gradient, $\bar{G}$ is the filtered gradient, and α is a length scale related to the filter width. The filter damps high-frequency modes, so this approach effectively defines a parameterization based on the low frequency eigenvectors of the discrete Helmholtz filter. Although the high frequency modes are damped, the number of degrees of freedom remains equal to the number of grid points; thus, this parameterization becomes impractical for

quasi-Newton optimization methods. A parameterization based on the low frequency modes alone could be realized by applying a low-pass filter. The advantages of such a modal parameterization are not clear. A possible disadvantage is the global nature of the basis functions.

Polynomial, Spline, and Fourier Approaches

The most general and flexible geometry parameterizations use some form of truncated series of basis functions. These parameterizations typically map a two-dimensional rectangular patch in $\mathbb{R}^2$ to a surface in $\mathbb{R}^3$. The basis functions may be polynomial, spline, or harmonic functions.

Several authors have used Bézier curves and surfaces to parameterize geometries for aerodynamic shape optimization. Burgreen and Baysal [16] used Bézier-Bernstein surfaces for inviscid transonic wing optimization studies. Bézier curves are polynomial curves that use a particular set of basis functions, Bernstein polynomials, which can be defined recursively and minimize round-off errors [38]. Unfortunately, complex curves must be represented using high-degree Bézier polynomials, which increases round-off error and computation time.

Spline curves offer many of the advantages of Bézier curves. Unlike Bézier curves, splines can capture complicated geometries by using knot insertion rather than degree elevation, eliminating problems associated with high-order polynomials. Basis-spline (B-spline) curves have been applied successfully to parameterize airfoils for multiobjective optimization of fully turbulent flows [121] and flows with laminar-turbulent transition [33]. Often, researchers use only one coordinate of the B-spline control points; the remaining coordinates are frozen.

Splines can be further generalized using nonuniform rational B-splines (NURBS), the standard geometric representation in CAD systems. NURBS can represent any Bézier or B-spline surface, and they can approximate cylinders and cones with fewer control points [38]. The work of Lépine *et al.* [100, 99] provides an example of a NURBS-based parameterization in aerodynamic shape optimization.

Series based on trigonometric functions and certain orthogonal polynomials exhibit spectral rates of convergence when approximating smooth functions. Their spectral convergence makes these series very attractive for parameterization, since, relative to other methods, fewer parameters will be required to represent the same surface. Bruno *et al.* [15] have proposed using Fourier continuation analysis to represent surfaces with Fourier series. Using

their approach, the coefficients of the Fourier series could be adopted as design parameters. Such a parameterization would share some characteristics with the eigenfunction-based approach described earlier, including the global nature of the basis functions.

The above methods are not well suited to surfaces that are logically triangular in shape, or have more general topologies. For example, when a tensor product B-spline surface is used to approximate a triangular surface, a singularity in the mapping results at one of the corners of the triangle. Bézier triangles and subdivision surfaces overcome these difficulties and may offer a viable parameterization for more general topologies; however, further research is needed to confirm these remarks.

CAD-Based Parameterization

Adopting a CAD-based parameterization is attractive for a number of reasons. Such an approach would provide a single source for the geometry [155]. CAD systems use NURBS, whose excellent approximation power has already been discussed. The difficult task of linking the CAD system with the analysis software can be simplified using tools like CAPRI [120, 44].

Unfortunately, CAD systems are presently unable to provide sensitivity information, necessitating the use of expensive, and potentially inaccurate, finite-difference evaluations. Discontinuous objective functions may also arise if the underlying topology of the CAD model changes [155]. These issues would present a challenge for gradient-based optimizations.

Free-Form Deformation

Sederberg and Parry [161] developed free-form-deformation (FFD) algorithms as a universal way to manipulate solid geometric models. FFD methods are now considered a subset of computer graphics algorithms called soft object animations (SOA) [155]. The idea of FFD can best be described using a physical analogy. Suppose the geometry to be deformed is embedded in a rubber region. As the (geometrically simple) rubber region is deformed, the embedded geometry is deformed as well. In FFD algorithms the "rubber" can be a trivariate Bézier, B-spline, or NURBS volume, or any other suitable volume transformation that maps $\mathbb{R}^3$ to $\mathbb{R}^3$. Newton's method can be used to find the set of computational coordinates that map to the physical coordinates of the geometry. Subsequently, the geometry can be deformed by manipulating the control points of the B-spline volume, for example.

FFD is very flexible, since it can handle arbitrary geometry definitions. Like the analytical perturbation approach, this method focuses on parameterizing the perturbations. It is also capable of performing large shape changes with few parameters. These attributes make FFD an attractive approach for shape optimization. Indeed, FFD has been successfully applied in this field by Perry *et al.* [133] and Samareh [154].

While the FFD method is well suited to global, large-scale shape changes, it does present problems if we want to perform many local, small-scale perturbations. Specifically, the FFD volume may require thousands of design variables in order to capture local perturbations. Nevertheless, the method is promising, and some of its elements have been incorporated into the present work.

2.1.2 Review of Mesh-Movement Algorithms

During the optimization process, as the design variables are updated and change the shape of the aerodynamic surface, the computational mesh must be adapted to conform to the geometric shape. Several algebraic and computational approaches have been proposed to move the mesh in response to the surface shape changes. I review these briefly in this section.

Grid Regeneration

Jameson [81] proposed the use of explicit conformal mappings to transform simple polar meshes into airfoil meshes. This was an early example of using grid regeneration in optimal shape design. For simple geometries and grid topologies, grid regeneration may be suitable for finding a body-fitted grid for aerodynamic shape optimization.

In Jameson's early work, an orthogonal computational mesh was entirely defined by the airfoil shape. This is arguably the ideal solution for mesh generation without a priori knowledge of the flow field. Unfortunately, despite years of research effort, automated generation of body-fitted grids around arbitrary three-dimensional shapes remains a difficult problem. Moreover, even for those cases where automatic mesh generation is possible, obtaining accurate mesh sensitivities involves differentiating very large and complicated software.

Cut-cell meshes offer an alternative to body-fitted grids for some flow discretizations. Adaptive Cartesian grid methods — see [1] for an overview — and simplex cut-cell methods

[41] can adapt to a new shape by "cutting" those cells that the geometry passes through. Objective gradients can be obtained for a fixed grid using cut-cell methods, but gradient-based optimization may be challenging, since the discrete objective becomes inherently discontinuous as the geometry changes and the grid adapts. Despite this difficulty, Nemec and Aftosmis [120] demonstrated that the gradients may be of sufficient accuracy for practical optimization of complex configurations.

Algebraic Mesh Movement

Algebraic mesh movement can be applied to multi-block structured grids [88, 143], but this approach is typically limited to small shape changes. Liu *et al.* [103] have proposed an algebraic mesh movement based on mapping the mesh to a Delaunay graph. They demonstrate that the Delaunay graph approach is robust for large shape changes, provided multiple increments are used; however, analysis of their method suggests that, in general, using multiple increments produces a discontinuous objective function. This may limit the Delaunay graph approach to gradient-free optimizations.

Radial Basis Functions

Jakobsson and Amoignon [80] developed a promising mesh-movement algorithm based on radial basis functions (RBFs); see also [2], where a similar approach is applied in the context of aeroelastic problems. In RBF mesh movement, the surface geometry displacements are interpolated into the interior. The interpolation coefficients are obtained by inverting a relatively large, dense system for each mesh movement. After the coefficients are determined, applying the interpolation is equivalent to a sparse matrix-vector multiplication. Depending on whether this sparse matrix is stored or recomputed, the RBF approach can be computationally or memory intensive. In addition, the mesh may not interpolate the geometry unless the geometry is also parameterized using radial basis functions [80].

PDE-Based Approaches

Batina [9] introduced the spring analogy method, which models the mesh edges as springs. While more robust than most algebraic algorithms, the spring analogy method can produce negative cell volumes [124]. To address this problem in 2-dimensions, Farhat *et al.* [37] incor-

porated torsional springs into the spring analogy method. Subsequently, Degand and Farhat [32] extended this torsional-spring mesh-movement method to 3-dimensional applications. To reduce the computational cost incurred by the torsional-spring methods, Blom [13] and Zeng and Ethier [187] have developed semi-torsional spring methods.

Johnson and Tezduyar [87] demonstrated that the equations of linear elasticity can be used to achieve robust mesh movements, even for large shape changes. This approach has been used successfully for aeroelastic problems [184] and aerodynamic optimization [124, 175]. Unfortunately, the equations of linear elasticity are typically less diagonally dominant than the spring analogy equations, so the elasticity approach tends to be more computationally expensive.

2.1.3 Integrated Parameterization and Mesh Movement

I cannot argue that one method of parameterization is clearly superior to all others; however, approximation theory provides a powerful argument in favour of B-splines. Consider the space of smooth functions $f \in C^m[a,b]$; most practical geometric shapes can be broken into functions in this space. Let $\mathcal{S}_p$ denote the space of splines of order p (degree $p-1$) with simple knots, and let $\bar{\Delta}$ denote the maximum knot interval associated with $\mathcal{S}_p$. Then the space $\mathcal{S}_p$ converges asymptotically to any $f \in C^m[a,b]$ with order p [160]. In particular, cubic B-splines will converge at a fourth-order rate to the smooth functions typically used in geometric applications. While approximations based on Fourier continuation [15] or Chebyshev partial sums may converge faster, B-splines provide local control of the geometry.

Recent studies and surveys of parameterizations by Castonguay and Nadarajah [21] and Mousavi *et al.* [116] confirm that B-spline curves are an efficient choice. It should be remarked, that neither of these studies considered adapting the knot vector in B-spline parameterizations; this would have greatly improved the effectiveness of the B-splines curves.

In light of the above observations and the objectives of the present work, I have chosen to use B-spline surfaces for this thesis. To address mesh movement, I have implemented a method inspired by free-form deformation. The method applies a robust mesh-movement algorithm to a B-spline control grid that mimics a coarse mesh; the actual computational mesh is regenerated algebraically from the B-spline volumes [185, 186]. In addition, the control points on the geometry surface are the design variables, so mesh movement and

geometry parameterization are tightly integrated. Indeed, this approach can be considered a volume parameterization of the computational mesh, with the volume variables constrained by the surface variables via a mesh-movement equation.

The remainder of this chapter describes the proposed integrated approach in detail. The content has evolved from a conference paper that I presented at the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference [67]. Section 2.2 reviews B-spline concepts and their extension to volume grids. The linear elasticity mesh movement, which is applied to the B-spline control grid, is outlined in Section 2.3. The parameterization and mesh movement are demonstrated in Section 2.4 with two examples.

2.2 B-spline Mesh Concepts

2.2.1 B-Spline Curves and Basis Functions

A B-spline curve of order p is a weighted sum of basis functions:

$$\mathbf{C}(\xi) = \sum_{i=1}^{N} \mathbf{B}_i \mathcal{N}_i^{(p)}(\xi), \qquad 0 \leq \xi \leq 1. \tag{2.1}$$

The points $\{\mathbf{B}_i\}_{i=1}^{N}$ are called de Boor control points, or simply control points. The functions $\mathcal{N}_i^{(p)}(\xi)$ are the B-spline basis functions of order p; they are $p-1$ degree spline polynomials. The polynomials making up the splines are joined at non-decreasing knot locations, $\{T_i\}_{i=1}^{p+N}$, such that the basis is C^{p-2} continuous at the knots. The parameter range for the curve is given by $[0,1] = [T_1, T_{p+N}]$; other ranges can be scaled to this domain. The basis functions can be defined recursively using the parameter value and the knots.

$$\begin{aligned}
\mathcal{N}_i^{(1)}(\xi) &= \begin{cases} 1 & \text{if } T_i \leq \xi < T_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \\
\mathcal{N}_i^{(p)}(\xi) &= \left(\frac{\xi - T_i}{T_{i+p-1} - T_i} \right) \mathcal{N}_i^{(p-1)}(\xi) + \left(\frac{T_{i+p} - \xi}{T_{i+p} - T_{i+1}} \right) \mathcal{N}_{i+1}^{(p-1)}(\xi).
\end{aligned} \tag{2.2}$$

In general, a B-spline curve can be controlled using the order of its basis, the number and position of its de Boor points, and its knot values and their multiplicity. Unless stated otherwise, I use fourth-order basis functions (cubic splines) and drop the superscript:

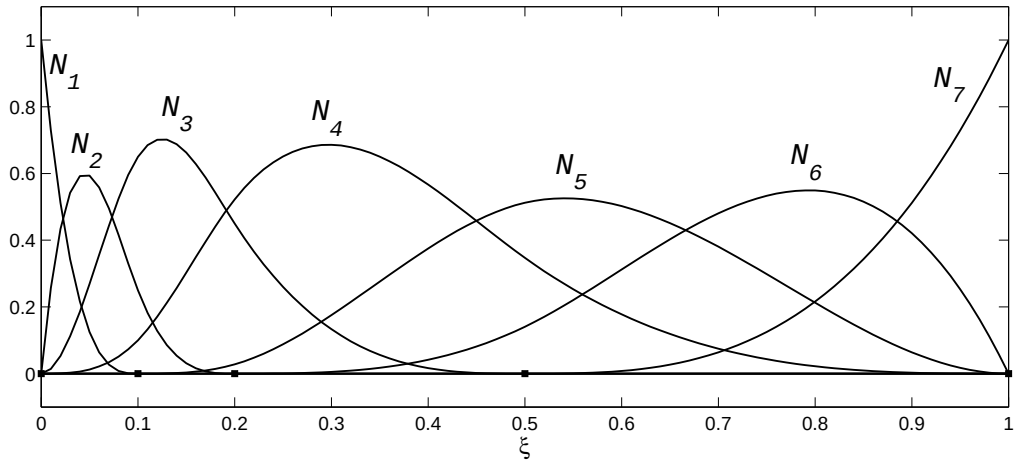


Figure 2.1: Example B-spline basis for the knot vector $(0, 0, 0, 0, 0.1, 0.2, 0.5, 1, 1, 1, 1)^T$; the knot locations are denoted by the square symbols along the ξ axis.

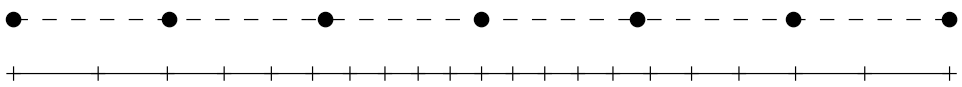
$\mathcal{N}_i(\xi) \equiv \mathcal{N}_i^{(4)}(\xi)$. This choice of basis is common, and provides a good compromise between continuity and computational speed.

The knot vector consists of the $p + N$ knot values listed in non-decreasing order. I use open knot vectors of the form $(0, 0, 0, 0, T_5, T_6, \ldots, T_N, 1, 1, 1, 1)^T$. The multiplicity of $p = 4$ at the ends of the open knot vector ensures that the fourth-order B-spline curve passes through $\mathbf{B}_1$ and $\mathbf{B}_N$. Figure 2.1 illustrates the fourth-order B-spline basis functions for the nonuniform knot vector $(0, 0, 0, 0, 0.1, 0.2, 0.5, 1, 1, 1, 1)^T$.

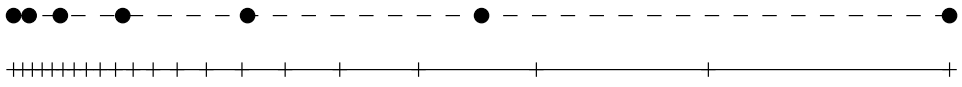
2.2.2 1-Dimensional B-Spline Meshes

The B-spline curve equation (2.1) is typically used to define a mapping of the form $\mathbf{C} : [0, 1] \rightarrow P$, where P is a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$. However, we can also use (2.1) to map one interval of the real line to another. Such 1-dimensional B-spline curves provide a suitable analogy and introduction to the target application, volume mesh generation and movement.

Consider a set of uniformly spaced 1-dimensional control points $\{B_i\}_{i=1}^N$ and a uniform open knot vector, i.e. a knot vector where the internal knots are evenly spaced between the end knots. The B-spline "curve" constructed using these control points and knots produces a mapping $C : [0, 1] \rightarrow [B_0, B_N]$. If we discretize the parameter space into L points,



(a) initial B-spline mesh and control points (solid circles).



(b) mesh perturbed using control points.

Figure 2.2: Example showing influence of control points on the grid spacing of a 1-dimensional B-spline mesh.

$(\xi_1, \xi_2, \ldots, \xi_L)^T$, then the image of these points is a 1-dimensional B-spline mesh. Figure 2.2(a) provides an example of a B-spline mesh and its control points. In this particular example, the control points, knots, and parameter values are uniformly spaced. The resulting mesh itself is not uniform because of the asymmetry of the basis functions. However, a uniform mesh can be obtained from uniformly spaced control points by changing the ξ parameter values [75].

We can adapt the B-spline mesh by altering the positions of the control points. Such an adaptation is shown in Figure 2.2(b) for illustrative purposes. Notice that the control points act like a coarse mesh; when they move closer together, the spacing on the actual mesh also decreases.

The coarse-grid analogy for the control points applies not only to mesh spacing but to space curves as well. B-spline curves of order p are constrained to lie within the convex hull of p neighbouring control points, and they also obey a variation diminishing property [146]. Therefore, the B-spline control points approximate the 2-dimensional or 3-dimensional space curves they define. Indeed, as more control points are added, using knot insertion for example, the control points converge to the geometry they define. This coarse grid property of the control points is particularly useful when moving B-spline volume meshes.

2.2.3 B-Spline Volume Meshes and Modified Basis Functions

B-spline surfaces and volumes can be created using tensor products. I will focus on B-spline volumes, since surfaces can be considered a special case. Tensor product B-spline volumes are well suited to structured-grid generation [185, 186]; however, B-spline volumes can also be generated by generalizing triangular Bézier patches, which may be of interest to unstructured grid users.

A B-spline tensor product volume is a mapping from the cubic domain $D = \{\boldsymbol{\xi} = (\xi, \eta, \zeta) \in \mathbb{R}^3 | \xi, \eta, \zeta \in [0, 1]\}$ to $P \subset \mathbb{R}^3$ and is defined by

$$\mathbf{x}(\boldsymbol{\xi}) = \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} \sum_{k=1}^{N_k} \mathbf{B}_{ijk} \mathcal{N}_i(\xi) \mathcal{N}_j(\eta) \mathcal{N}_k(\zeta). \tag{2.3}$$

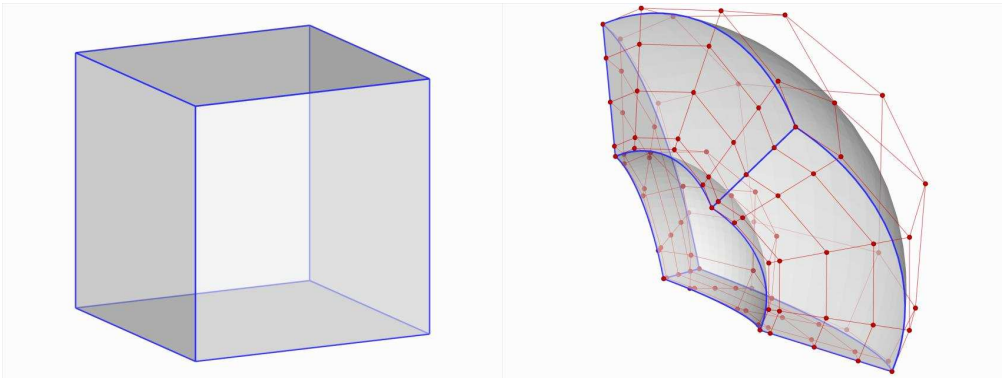
A B-spline volume mesh is produced by simply discretizing the domain D. The parameters ξ, η, and ζ do not need to be discretized in a uniform way. Indeed, nonuniform parameter spacing is usually necessary for precise control of mesh spacing in physical space. If a flow solver requires uniform mesh spacing in parameter space, an intermediate mapping is implied.[1] B-spline volume meshes are exemplified in Figure 2.3.

The basis functions appearing in (2.3) can be generalized to permit spatially varying knot functions, also called curved knot lines [60]. The usefulness of such knot functions will be explained below. Consider the basis functions in the ξ-direction, and suppose the knots are functions of η and ζ. Then, the modified basis functions are given by

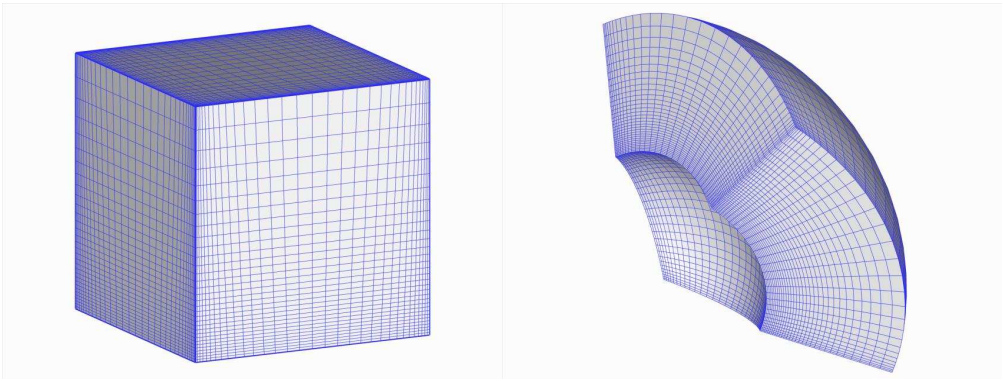
$$\begin{aligned}
\mathcal{N}_i^{(1)}(\xi; \eta, \zeta) &= \begin{cases} 1 & \text{if } T_i(\eta, \zeta) \leq \xi < T_{i+1}(\eta, \zeta), \\ 0 & \text{otherwise.} \end{cases} \\
\mathcal{N}_i^{(p)}(\xi; \eta, \zeta) &= \left(\frac{\xi - T_i(\eta, \zeta)}{T_{i+p-1}(\eta, \zeta) - T_i(\eta, \zeta)} \right) \mathcal{N}_i^{(p-1)}(\xi; \eta, \zeta) \\
&\quad + \left(\frac{T_{i+p}(\eta, \zeta) - \xi}{T_{i+p}(\eta, \zeta) - T_{i+1}(\eta, \zeta)} \right) \mathcal{N}_{i+1}^{(p-1)}(\xi; \eta, \zeta).
\end{aligned} \tag{2.4}$$

Analogous definitions apply for the basis functions $\mathcal{N}_j^{(p)}(\eta; \zeta, \xi)$ and $\mathcal{N}_k^{(p)}(\zeta; \xi, \eta)$.

[1] In practice, finite-difference and other mapping-based solvers use the grid coordinates only, so the details of the B-spline parameters are not important; however, the smoothness of the mapping must be consistent with the order of the discretization being used. The fourth-order splines used here are suitable for discretizations up to third-order.



(a) Analytical mapping described by B-spline volume



(b) Mesh obtained by discretizing computational space

Figure 2.3: Example B-spline mesh. The upper figures illustrate the analytical mapping from a cubic computational space to the region defined by the B-spline volume; the control points are shown as red spheres. The lower figures show how the mesh in physical space is obtained by discretizing the computational space.

For fixed (η, ζ), the modified basis function (2.4) reduces to the standard definition (2.2); hence, the modified basis retains C^{p-2} continuity at the knots in the ξ direction. Although less obvious, the modified basis is also C^{p-2} in the η- and ζ-directions, provided $T_i(\eta, \zeta) \in C^{p-2}$ and the internal knots have a multiplicity at most one. This result is a consequence of the chain rule and the smoothness of the derivative of a B-spline with respect to its knots [160, 182, 134].

The internal knots of the modified basis functions must be strictly increasing and sufficiently smooth, but the user is otherwise free to choose the functional form. For simplicity, I use bilinear knots of the form

$$T_i(\eta, \zeta) = T_{i,(0,0)}(1-\eta)(1-\zeta) + T_{i,(1,0)}\eta(1-\zeta) + T_{i,(0,1)}(1-\eta)\zeta + T_{i,(1,1)}\eta\zeta. \tag{2.5}$$

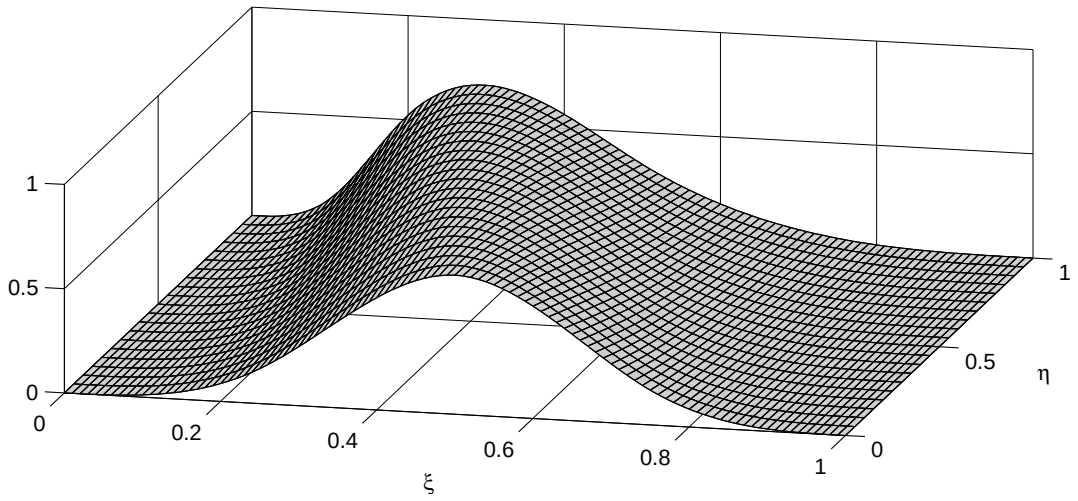


Figure 2.4: Example 2-dimensional modified basis, or, equivalently, a 3-dimensional modified basis for fixed ζ.

The four constants $T_{i,(0,0)}$, $T_{i,(1,0)}$, $T_{i,(0,1)}$, and $T_{i,(1,1)}$ denote the knot values at the (η, ζ) edges of the parameter space. Again, similar knot definitions are used for $T_i(\zeta, \xi)$ and $T_i(\xi, \eta)$. Figure 2.4 shows an example modified basis function. The salient feature is the changing basis location.

2.2.4 Approximating Grids Using B-spline Volume Meshes

I have described how B-spline volumes can be used, in theory, to define mappings from computational space to physical space. Many practical issues remain unanswered. For example, how do we choose the surface control points such that the mappings conform to the geometry of interest? This is certainly an important question when a mesh is generated for analysis, but it is less important in preliminary design where the geometry of interest is the optimized one. Nevertheless, as mentioned in Section 2.1.3, approximation theory implies that an initial geometry can be approximated to any specified tolerance using B-spline meshes.

Issues related to mesh generation also need to be addressed. How should the volume control points and parameter values be determined to obtain a high quality mesh? Should the domain be broken into multiple mappings using a semi-structured approach? These are

open-ended questions, and to provide thorough and exhaustive answers would go beyond the scope of this thesis. For the purposes of demonstrating the B-spline parameterization and mesh movement, I fit existing grids to determine the initial control-point positions and the (ξ, η, ζ) parameter values.

A least-squares fit with parameter correction [75] is often used to find spline curve and surface approximations to data points. This algorithm can easily be extended to B-spline volume approximations of multi-block structured grids, as I now demonstrate.

Each block of the structured grid is associated with a B-spline volume. For each B-spline volume, the user chooses the number of control points and the B-spline order in the parameter directions (ξ, η, ζ). At interfaces where blocks meet, the number of control points and order must be consistent to ensure continuity.

First, parameter values $\boldsymbol{\xi}_{q,r,s}$ are calculated for each point in the grid $G = \{\mathbf{x}_{q,r,s} | q = 1, \ldots, L_q, r = 1, \ldots, L_r, s = 1, \ldots, L_s\}$. I use a chord-length parameterization to be consistent with the choice of knot vectors (see below), but other parameterizations are possible [38]. To illustrate the chord-length parameterization, consider the parameter $\xi_{q,r,s}$ along the grid line $\{q = 1, \ldots, L_q, r = r_0, s = s_0\}$:

$$\begin{aligned} \xi_{1,r_0,s_0} &= 0 \\ \xi_{q,r_0,s_0} &= \frac{1}{\Xi} \sum_{t=1}^{q-1} \|\mathbf{x}_{t+1,r_0,s_0} - \mathbf{x}_{t,r_0,s_0}\|, \qquad q = 2, \ldots, L_q, \end{aligned}$$

where the total arc length is approximated by

$$\Xi = \sum_{t=1}^{L_q-1} \|\mathbf{x}_{t+1,r_0,s_0} - \mathbf{x}_{t,r_0,s_0}\|.$$

Next, the spatially varying knot vectors must be determined. The importance of the knot vector in fitting B-spline curves is well documented [76, 38]. When fitting surface or volume data points with conventional algorithms, there are an infinite number of curves being fit with the same knot vector. By allowing the knots to vary, it should be possible to improve the B-spline approximation of surface and volume data. I use the spatially varying bilinear knots defined by (2.5), with the knots on each parameter edge defined by a chord length parameterization. I have found that chord-length-based knots produce control meshes that approximate the mesh spacing of a coarse grid, a feature that is important for the mesh-movement algorithm. The specific formula for the edge knots will be demonstrated using

the edge $\{(\xi, 0, 0)|\xi \in [0, 1]\}$. For this edge we have

$$T_{i,(0,0)} = (\xi_{q+1,1,1} - \xi_{q,1,1})[\bar{q}_i - (q-1)] + \xi_{q,1,1}, \quad q - 1 \leq \bar{q}_i \leq q$$

where q is chosen such that $q - 1 \leq \bar{q}_i \leq q$, with $\bar{q}_i$ given by

$$\bar{q}_i = \left(\frac{L_q - 1}{N_i + 1 - p}\right)(i - p).$$

These edge knots are constructed to have approximately the same number of parameter values in each knot interval, $[T_i, T_{i+1}], i = p, N_i - 1$. Consequently, the knots obtained using the above formula have a chord length parameterization, because the parameters use a chord length parameterization.

Finally, the algorithm solves least-squares systems to determine the control points that best approximate the grid; see reference [38], for example. Least-squares problems are solved sequentially for the edge, face, and internal control points. This ordering ensures that adjacent blocks have consistent grid point locations. If necessary, the iterative parameter correction algorithm of Hoschek [75] can be applied to improve the fit by adjusting the parameter values.

2.2.5 Applications of B-Spline Meshes

The control mesh is designed to mimic a coarse grid, so the surface control points provide a low dimensional approximation of the surface. This makes the surface control points a suitable choice for the design variables in shape optimization. As discussed at the beginning of the chapter, the use of Bézier or B-spline control points in design optimization is not new [16]; the difference here is that the design variables are a subset of a volume control mesh.

While this chapter is focused on parameterization and mesh movement in design optimization, B-spline volumes may be useful in other contexts:

- the surface control points can be used to control the shape of the wing in studies of aero-elasticity;

- the volume control points can be used for mesh adaptation, rather than the individual grid points;

- the parameter space can be refined in a smooth way to achieve rigorous mesh convergence studies (see Section 5.3 for an example);
- hierarchical grids can easily be constructed for multigrid, and;
- grids and geometries suitable for high-order discretizations can be obtained with a suitable choice of B-spline basis order.

2.3 B-spline Mesh Movement

2.3.1 Possible Mesh-Movement Algorithms

Once fitted using B-spline volumes, the grid can be manipulated using the control mesh of points $\{\mathbf{B}_{ijk}\}$. Suppose that a subset of control points on the surface, denoted by $\{\mathbf{B}_{ijk}|(i,j,k) \in S\}$, has been chosen to define the design variables. A very simple mesh-movement algorithm is obtained by fixing the control points that are not in this subset. Although this fixes the internal control points, the grid points near the surface still move, because the boundary B-spline basis functions extend into the interior. Such a mesh-movement strategy may be useful if only small shape changes are necessary.

If larger shape changes are expected, then an algorithm is needed that moves the internal control points based on the surface control points. Essentially, any grid movement algorithm can be applied to the control points: algebraic, spring analogy, linear elasticity, radial basis functions, etc. For B-spline volume meshes, there are typically two orders of magnitude fewer control points than grid points, so the CPU time of the mesh movement becomes insignificant relative to the flow solution. For this reason I have chosen to use a robust, albeit expensive, linear elasticity-based mesh movement.

2.3.2 Control-Point Movement Based on Linear Elasticity

I model the control mesh as a linear elastic solid with stiffness controlled using a non-constant Young's modulus, E. The algorithm is very similar to the one used in Truong *et al.* [175], so only a brief outline of the method is given here.

The control points are updated according to the equilibrium equation

$$\frac{\partial\left(\tau_{ij}\right)}{\partial x_j} + f_i = 0, \tag{2.6}$$

where the stress tensor, τ_{ij}, is assumed to be linearly related to the Cauchy strain tensor, e_{ij}. I also assume the solid is isotropic and homogeneous, and that strains are small. Under these assumptions we have

$$\begin{aligned}\tau_{ij} &= \frac{E}{1+\nu}\left(e_{ij} + \frac{\nu e_{kk}\delta_{ij}}{1-2\nu}\right)\\ e_{ij} &= \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right),\end{aligned}$$

where u_i is the i^{th} component of the displacement, and ν is the Poisson's ratio. Note, when the small-strain assumption is not valid, the problem can be broken into a sequence of mesh-movement problems by moving the surface in increments. I will elaborate on this below.

The governing equation (2.6) is discretized on the control mesh using the finite element method with trilinear elements. The force vector f_i is defined implicitly based on the displacement of known degrees of freedom, such as the surface and far-field control points. The resulting system of algebraic equations is of the form

$$\boldsymbol{\mathcal{M}} = K(\mathbf{b} - \mathbf{b}^{(0)}) - \mathbf{f} = 0, \tag{2.7}$$

where $\mathbf{b}$ is a block-column vector of control-point coordinates, $\mathbf{f}$ is the discrete force, and the stiffness matrix K is sparse, symmetric, and positive-definite. The initial control-point coordinates are denoted by $\mathbf{b}^{(0)}$. I solve the linear system (2.7) using the conjugate gradient method preconditioned with ILU(p) [114]. The system is solved to a relative tolerance of 10^{-12} with respect to the L^2 norm of the residual.

As noted above, when large shape changes are expected, the mesh movement can be applied in increments. Suppose m increments are used, with $\mathbf{b}_{\text{srf}}^{(0)}$ and $\mathbf{b}_{\text{srf}}^{(m)}$ denoting the initial and final locations of the surface control points, respectively. Then the intermediate values of the surface control points are determined by the linear interpolation

$$\mathbf{b}_{\text{srf}}^{(i)} = \frac{i}{m}\left(\mathbf{b}_{\text{srf}}^{(m)} - \mathbf{b}_{\text{srf}}^{(0)}\right) + \mathbf{b}_{\text{srf}}^{(0)}, \qquad i = 1, \ldots, m. \tag{2.8}$$

In addition, when increments are used, the stiffness matrix becomes a function of the control-point coordinates at the previous level through the spatially varying Young's modulus (explained below). Thus, the linear equation for the control points at increment i has the form

$$\boldsymbol{\mathcal{M}}^{(i)} = \mathsf{K}^{(i)}\left(\mathbf{b}^{(i-1)}\right)\left[\mathbf{b}^{(i)} - \mathbf{b}^{(i-1)}\right] - \mathbf{f}^{(i)} = 0, \qquad i = 1, \ldots, m. \tag{2.9}$$

Using increments helps improve the robustness of the mesh movement while maintaining linearity; however, this approach does have implications for gradient-based optimization, since $\mathsf{K}^{(i)}$ is a function of $\mathbf{b}^{(i-1)}$ (see Section 4.4 of Chapter 4).

Element stiffness is controlled using a spatially varying Young's modulus. The Young's modulus is calculated at the beginning of the mesh movement, or the beginning of each increment if the movement is broken into smaller steps. The goal is to vary the element stiffness in such a way that mesh quality is maintained in critical regions of the grid, e.g. the boundary layer. First, I assume that $E_{\mathcal{E}} \propto V_{\mathcal{E}}^{-1}$, where $V_{\mathcal{E}}$ is the element volume; this ensures that small elements are stiffer than large elements. In regions where element volumes have similar magnitudes, an additional stiffening mechanism is required to maintain quality. I have chosen to use an orthogonality measure, so that elements that are at risk of becoming more skewed have their stiffness increased. The orthogonality measure for an arbitrary element $\mathcal{E}$ is given by

$$\Phi_{\mathcal{E}} = \left(\prod_{v=1}^{8} \frac{1}{(\mathbf{u}_v \times \mathbf{v}_v) \cdot \mathbf{w}_v}\right)^2,$$

where the index v denotes a vertex of the element. The vectors $\mathbf{u}_v$, $\mathbf{v}_v$, and $\mathbf{w}_v$ are unit vectors parallel to the element edges that meet at vertex v, and they form a right-handed system. Hence, the triple product $(\mathbf{u}_v \times \mathbf{v}_v) \cdot \mathbf{w}_v$ is positive for valid elements, equal to 1 for orthogonal vectors, and tends to zero as the vectors become coplanar. Following Bar-Yoseph [7], the orthogonality measure is normalized by its value on the initial mesh. In summary, the Young's modulus for element $\mathcal{E}$ at increment i is given by

$$E_{\mathcal{E}}^{(i)} = \frac{\Phi_{\mathcal{E}}^{(i-1)}}{\Phi_{\mathcal{E}}^{(0)} V_{\mathcal{E}}^{(i-1)}}, \qquad i = 1, 2, \ldots, m.$$

2.4 Examples

I demonstrate the proposed mesh movement and parameterization scheme with two examples. In the first example, I parameterize an existing shape — the ONERA M6 wing [157] — to test the algorithm in a typical design optimization setting. The second example is intended to mimic a conceptual design optimization and involves morphing a flat plate into a blended-wing body with winglets. In both cases the proposed mesh movement is compared with a node-based-linear-elasticity approach, i.e. the algorithm described in Section 2.3.2 is applied to individual nodes rather than control points. In all cases, the mesh movement is broken into five increments ($m = 5$), and Poisson's ratio ν is fixed at -0.2.

To assess the mesh-movement algorithms, I adopt an orthogonality measure similar to the one used to stiffen elements in the linear elasticity algorithm. Specifically, the quality for an element $\mathcal{E}$ is given by

$$Q_{\mathcal{E}} = \sqrt{\frac{1}{\Phi_{\mathcal{E}}}} = \prod_{v=1}^{8} (\mathbf{u}_v \times \mathbf{v}_v) \cdot \mathbf{w}_v. \tag{2.10}$$

It follows from this definition that elements with perfect orthogonality have $Q_{\mathcal{E}} = 1$ while highly skewed elements have $Q_{\mathcal{E}} \approx 0$. In the case of the B-spline mesh movement algorithm, $Q_{\mathcal{E}}$ is measured for elements on the interpolated mesh and not the control mesh.

The fitting process used to determine the B-spline control mesh and parameter spacing produces an approximation to the initial mesh. To quantify the quality of the fit of the initial mesh, I use the root-mean-square (RMS) and L^∞ norm errors. The RMS error is based on the distances between corresponding nodes in the initial and fitted grids:

$$\text{RMS}_{\text{error}} = \sqrt{\frac{\sum_{q,r,s} \left\| \mathbf{x}_{q,r,s} - \mathbf{x}(\boldsymbol{\xi}_{q,r,s}) \right\|^2}{\sum_{q,r,s} 1}}.$$

To be clear, the sum in the RMS error is over all nodes and all blocks. For the infinity norm errors, I report the maximum absolute error for each coordinate. The RMS and L^∞ norm errors for the two cases are listed in Table 2.1. These errors are acceptable for most inviscid analyses. If tighter tolerances are required, more control points can be added or parameter correction — which is not used here — can be invoked.

Table 2.1: Errors in fitted B-spline grids.

case	$\mathbf{RMS}_{\text{error}}$	$\|\mathbf{\Delta x}\|_\infty$	$\|\mathbf{\Delta y}\|_\infty$	$\|\mathbf{\Delta z}\|_\infty$
ONERA M6	4.04×10^{-4}	2.01×10^{-3}	1.52×10^{-3}	1.54×10^{-3}
blended-wing body	7.11×10^{-5}	1.35×10^{-4}	5.21×10^{-4}	2.41×10^{-4}

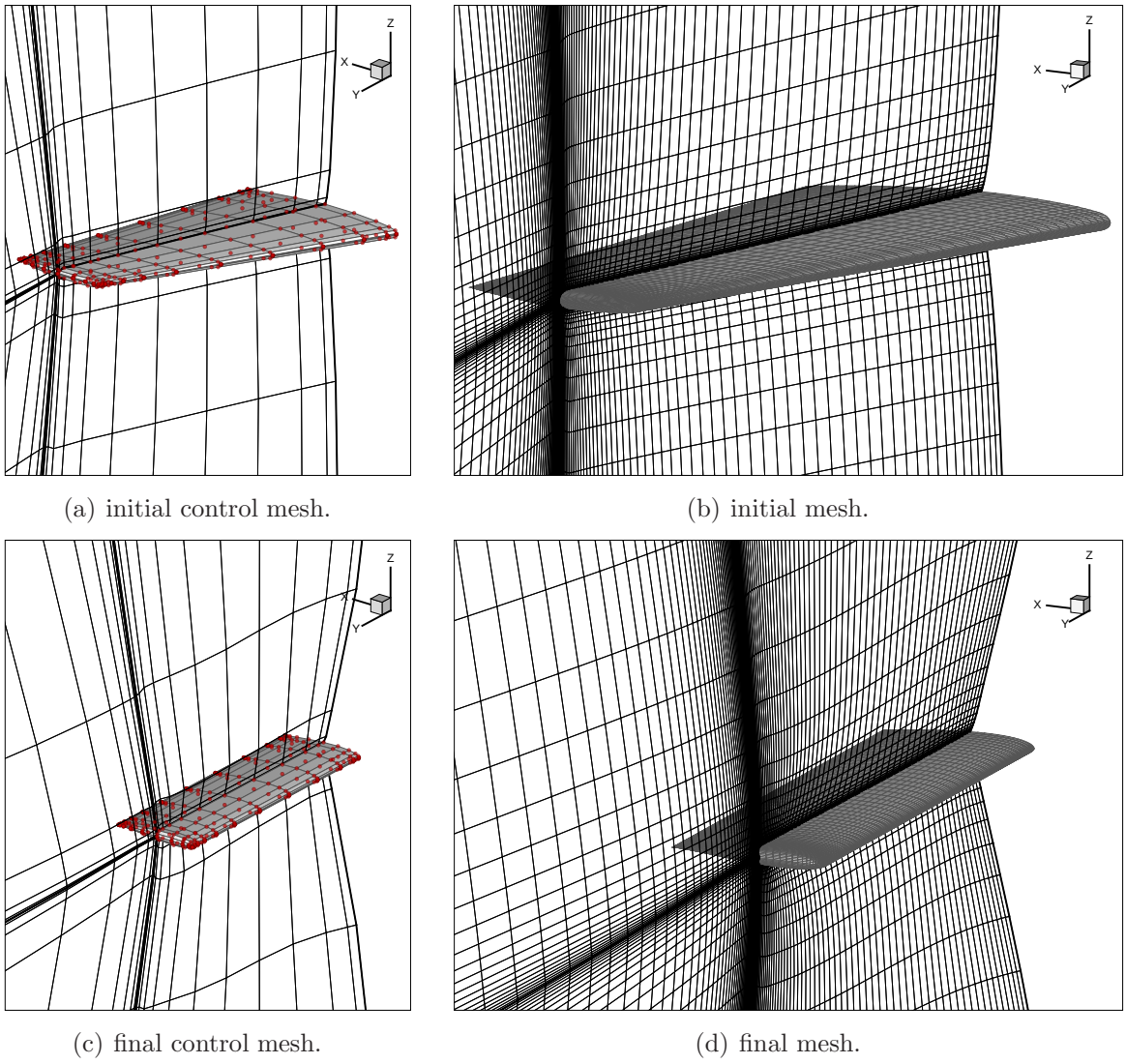
2.4.1 ONERA M6 wing

In this example, I parameterize the ONERA M6 wing and transform it into a wing with NACA 0012 airfoil sections and a rectangular planform. The root chord of the M6 wing is normalized to 1.0, and the rectangular wing has a root chord of 0.49664. The shape transformation involves sweep, scaling, and section modifications.

The parameterization is obtained from the surface control points of a B-spline mesh fitted to an initial 12 block, HH-topology grid. Each of the 12 blocks in the grid consists of $L_q \times L_r \times L_s = 45 \times 65 \times 33$ nodes, producing approximately 1.158×10^6 nodes in total. The mesh spacing is typical for an inviscid flow analysis with the off-wall, leading edge, trailing edge, and tip spacings set at 0.001. The B-spline volumes for each block use $N_i \times N_j \times N_k = 13 \times 13 \times 9$ control points: the B-spline grid is approximately 60 times smaller than the computational grid. Figure 2.5 shows the initial and final B-spline control meshes and their corresponding flow-analysis grids.

The final computational grid is compared to the initial grid using the orthogonality measure (2.10). The measure is calculated for each element, and then grouped into fifty bins that uniformly divide the range of possible values, namely $[0, 1]$. These bins are then used to produce a distribution of orthogonality. Integrating the distribution over the orthogonality range $[a, b]$ gives the ratio of elements that lie in this range. In particular, integrating the distribution over $[0, 1]$ gives 1.

Figure 2.6 plots the orthogonality distribution for the initial ONERA M6 grid and the grid for the rectangular wing. For comparison, the figure also includes the distribution for the grid obtained using the node-based-linear-elasticity approach. The final grids have distributions that are qualitatively similar to the initial distribution, which we would expect for a mesh movement based on linear elasticity. More notable are the similarities between the B-spline and node-based mesh-quality distributions. Indeed, in some cases the B-spline distribution is better; consider the first peak, near the low end of the quality range, which



(a) initial control mesh.

(b) initial mesh.

(c) final control mesh.

(d) final mesh.

Figure 2.5: Control grids for the ONERA M6 wing and modified geometry are shown in Figures 2.5(a) and 2.5(c), respectively. The corresponding interpolated grids are shown on the right.

is smaller for the B-spline mesh. This can be attributed to the smoothing properties of the B-spline volumes.

For this problem, the B-spline and node-based mesh movement required 227 seconds and 27.79 hours, respectively, on a single 1500 MHz Itanium 2 processor. For the B-spline mesh movement I found that a fill level of 1 was optimal in the ILU(p) preconditioner, while a fill level of 2 was better for the larger node-based problem. While better preconditioners may exist, the relative performance of the two approaches is ultimately bounded by the relative

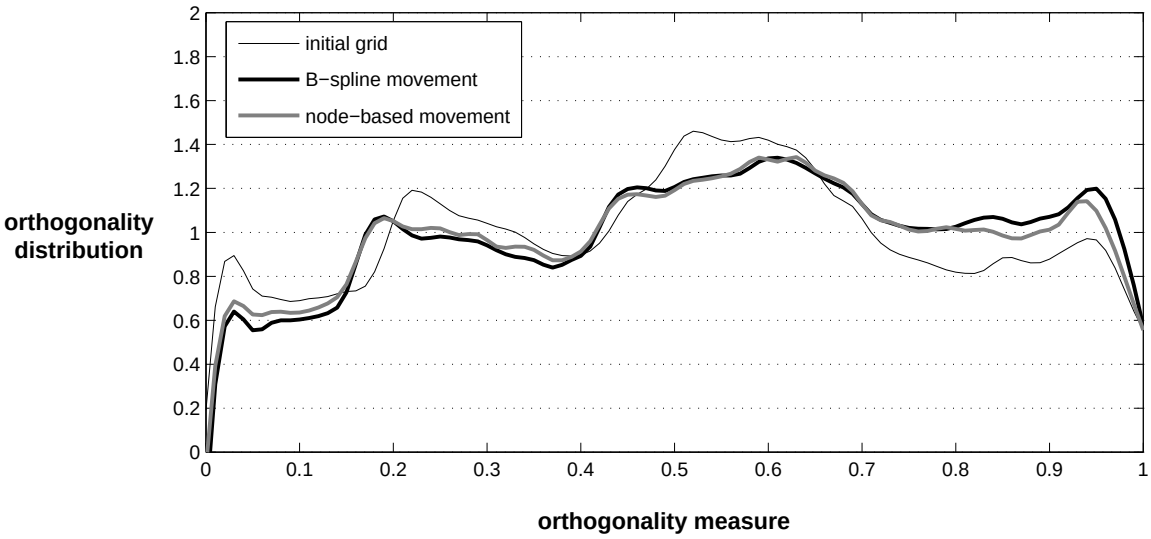


Figure 2.6: Orthogonality distribution of cells for the initial ONERA M6 grid and grids for the rectangular wing obtained using the B-spline mesh movement and node-based mesh movement.

size of their linear systems.

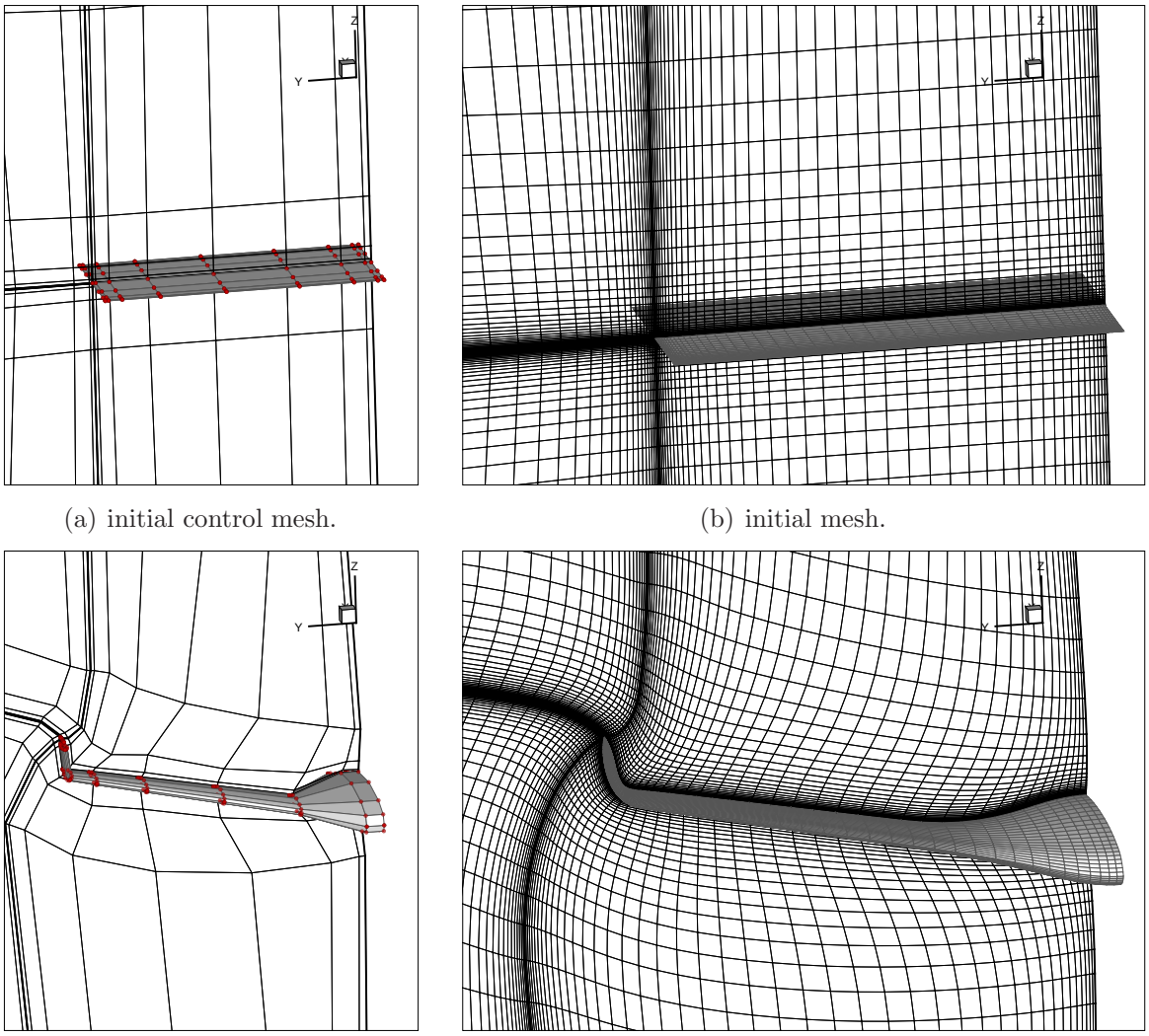
2.4.2 Blended-Wing Body

The initial shape is a flat plate with unit chord and a semi-span of 2. As in the previous example, the mesh consists of 12 blocks in an H-H topology. Each block has $45 \times 45 \times 45$ nodes. The off-wall spacing is 0.001, while the leading edge, trailing edge, and tip spacings are 0.005.

The initial mesh is fit using 12 B-spline volumes, with $9 \times 9 \times 9$ control points per volume. The control mesh reduces the number of degrees of freedom by a factor of 125 relative to the fine mesh. The fitted mesh for the flat plate is shown in Figure 2.7(b), together with its control mesh in Figure 2.7(a).

Again, the control points on the surface of the plate are the design variables. In this example, I set these design variables to obtain a generic blended-wing geometry with winglets. The perturbed surface control points and control mesh are shown in Figure 2.7(c). The resulting mesh for the blended-wing-body shape is shown in Figure 2.7(d).

The orthogonality distribution for the initial grid is plotted in Figure 2.8, together with



(a) initial control mesh.

(b) initial mesh.

(c) final control mesh.

(d) final mesh.

Figure 2.7: Control grids for the initial plate and BWB geometry are shown in Figures 2.7(a) and 2.7(c), respectively. The corresponding interpolated grids are shown on the right.

the distributions for the grids obtained using the B-spline and node-based mesh-movement algorithms. The initial grid is almost Cartesian and its distribution reflects this: all the elements have orthogonality measures greater than 0.74. In transforming from the flat plate to the blended-wing body, some loss of element orthogonality is unavoidable. As in the ONERA M6 example, the important observation is that the B-spline and node-based mesh movements produce very similar quality distributions, despite a significant difference in CPU time: the B-spline mesh movement required 128 seconds while the node-based mesh move-

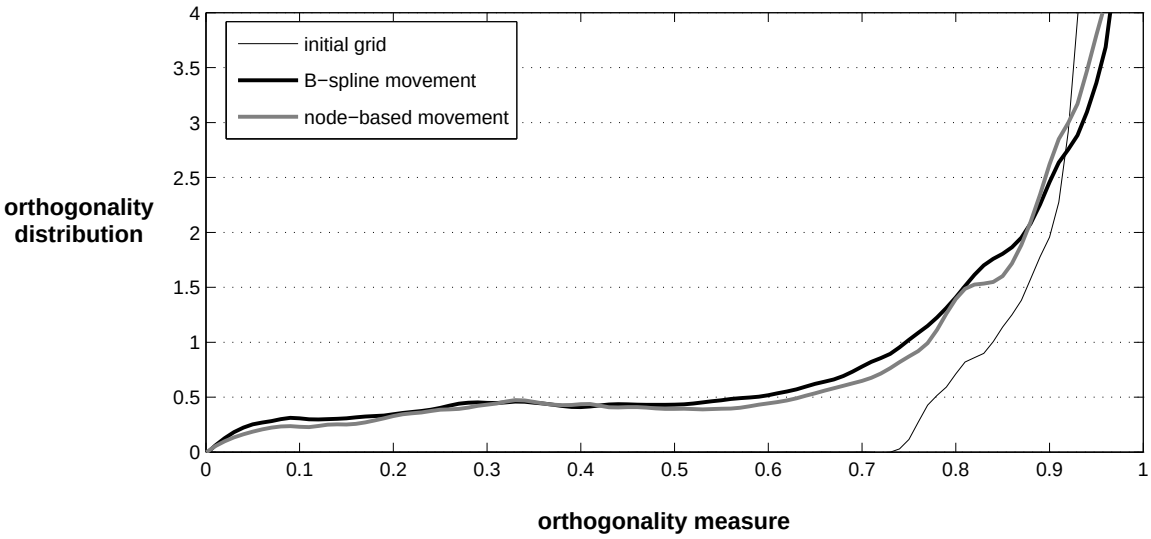


Figure 2.8: Orthogonality distribution of cells for blended-wing-body grids obtained using the B-spline mesh movement and node-based mesh movement.

ment required 32.4 hours.

Finally, I present the results of a flow solution around the blended-wing body to demonstrate the B-spline mesh movement in an analysis-type application. Figure 2.9 shows the surface mesh and C_p distribution for the blended-wing-body grid obtained using the B-spline algorithm. The inviscid flow is defined by an angle of attack of 4 degrees and Mach number of 0.3. The flow solution was obtained using the algorithm described in Chapter 3.

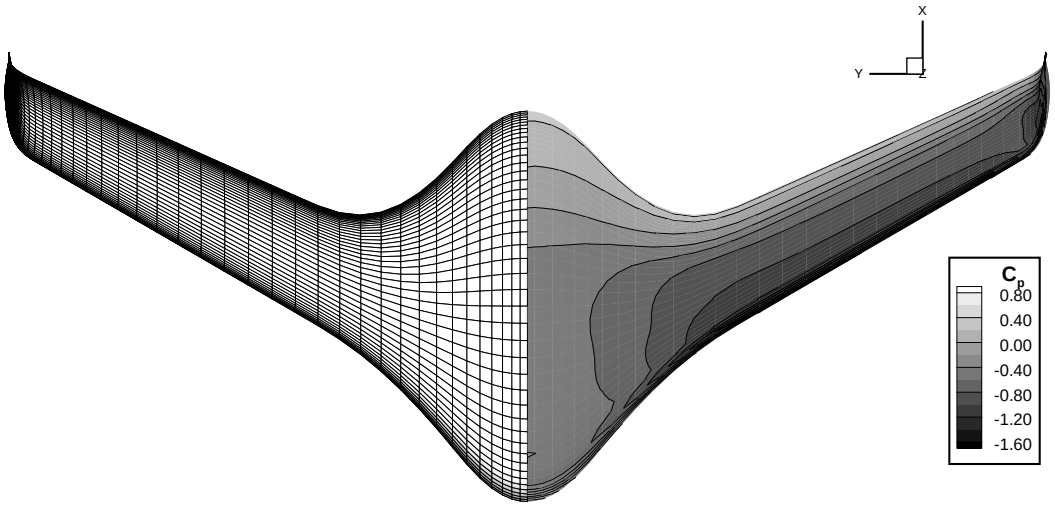


Figure 2.9: Euler flow solution on the blended-wing-body grid produced using the B-spline mesh-movement algorithm.

Chapter 3

Numerical Solution of the Euler Equations

The flow solver is a critical component in aerodynamic shape optimization. Consider the force and moment coefficients produced by the flow solver. If these are not sufficiently accurate, the optimization algorithm will produce a shape that exploits numerical errors rather than the physics of the problem. Thus, the flow solver dictates the validity of the optimization process. In addition, the solver often determines the practicality of high-fidelity aerodynamic shape optimization, since it is the most time-consuming component. In this chapter, I describe the methodology used to obtain a flow solution for a given geometry. The content is adapted from my work in references [69], [64], and [65].

3.1 Discretization and Solution Strategy Overview

Clearly, aerodynamic optimization requires an efficient discretization and solution strategy, but the choice of solver is also dictated by geometry and flow regime. For arbitrary geometries in inviscid flows, an adaptive Cartesian mesh with cut cells, such as Cart3D [1], is an excellent choice; however, Cartesian cut-cell algorithms have not been successfully extended to viscous and turbulent flows. Although my thesis focuses on inviscid flows, the long-term research programme requires a flow solver that can be easily adapted to model turbulent flows.

In his review of finite-difference (FD), finite-volume (FV), and discontinuous Galerkin (DG) finite-element discretizations, Shu [162] concluded that finite-difference discretizations are the most efficient choice, provided we relax the requirement to model flows around arbitrary geometries. Indeed, multi-block structured grids provide sufficient geometric flexibility to accommodate the large-scale features of most aircraft. In addition, the apparent geo-

metric compromise when using multi-block FD schemes is due more to the limitations of mesh generators than to the discretization itself. I also remark that, in consideration of subsequent research, multi-block FD schemes can easily accommodate the Reynolds-averaged Navier-Stokes (RANS) equations and high-order discretizations [28, 169].

Despite their apparent simplicity, multi-block finite-difference schemes present some challenges that must be addressed. Consider the discretization at nodes along a block interface. If we use the interior discretization for the interface nodes, for example through halo nodes, then the mesh must be sufficiently smooth at the interface to maintain the desired accuracy. This often motivates the use of elliptic smoothing; however, while elliptic smoothing can improve the grid continuity between blocks, it cannot eliminate mesh singularities along edges and vertices that are inherent in the geometry or the block topology. Resolving these singularities, through isotropic mesh spacing for example, is inefficient and unnecessary. Finally, the requirement for smoothness in the interface-normal direction places significant restrictions on how the mesh can be refined. One cannot, for example, refine one block independently of another.

Another issue posed by multi-block finite-difference schemes is the discretization along edges and vertices of the blocks. At these points the coordinate directions are either non-smooth, as discussed above, or ambiguous — consider an edge where only three blocks meet. An accurate and stable treatment of these exceptional points is not obvious, even for second-order methods, and, as the order of accuracy of the scheme increases, neighbouring points become exposed to this difficulty. In parallel computations, the exceptional points can introduce additional difficulties that diminish the appeal of finite-difference schemes.

The problems associated with block interfaces and exceptional points can be eliminated with summation-by-parts (SBPs) operators [96, 167] and simultaneous-approximation terms (SATs) [45, 19]. The SAT methodology was originally developed to enforce boundary conditions in an accurate and time-stable manner, but the method has also been extended to handle domain interfaces [62, 20, 129]. SATs have been successfully used by Mattsson *et al.* [112] for 3rd- and 5th-order discretizations of the Euler equations. High-order SBP-SAT methods retain the advantages of the second-order scheme described here — C^0 mesh interfaces, time-stability, exceptional point treatment — without the increased communication overhead typical of halo approaches.

Coupling blocks with SAT penalties has been shown to reduce the maximum stable

Courant-Friedrichs-Lewy (CFL) number for explicit schemes [63, 128]. This suggests that an implicit, Newton-Krylov solution strategy may be well suited to SAT discretizations. For serial computations, Newton-Krylov solution strategies have proven to be efficient, both in flow simulation [86, 126, 6, 137, 12, 24, 122] and optimization [121, 138]; however, for practical three-dimensional optimization problems, the solution algorithm must be parallel. Fortunately, there are also many examples demonstrating that the excellent serial performance of Newton-Krylov algorithms can be extended to parallel algorithms [8, 92, 89, 55, 95].

Krylov solvers are readily parallelizable with the possible exception of the preconditioner. The preconditioner poses a difficulty, because many of the best serial preconditioners tend to be inefficient when parallelized directly — consider, for example, incomplete lower upper factorizations [18] — and many parallel preconditioners tend to scale poorly. In this chapter, I evaluate an additive-Schwarz preconditioner [18] and an approximately factored Schur-complement preconditioner [153].

The chapter is organized as follows. Sections 3.2 and 3.3 focus on the governing equations and their discretization. In particular, I summarize the use of SBP operators and SATs for one-dimensional problems and their extension to three-dimensional problems. In Section 3.4, I review the Newton-Krylov method for solving the nonlinear discrete equations and the resulting distributed linear systems. I present verification and validation studies of the discretization in Section 3.5, and in Section 3.6 I assess the solution algorithm and compare the parallel preconditioners.

3.2 Governing Equations

3.2.1 The Euler Equations

The Euler equations model the conservation of mass, momentum, and energy for inviscid, compressible fluids. They can be considered a subset of the Navier-Stokes equations with vanishing viscosity. Note that the Euler equations require the continuum hypothesis, which is valid for the length scales and flow regimes considered in this thesis.

The Euler equations are defined by the following set of partial differential equations, given here in strong conservative form over a domain Ω:

$$\partial_t \mathbf{Q} + \partial_{x_i} \mathbf{F}_i = \mathbf{0}, \quad \forall\, \mathbf{x} \in \Omega, \tag{3.1}$$

where $\mathbf{x} = (x_1, x_2, x_3)^T = (x, y, z)^T$, $\partial_{x_i} \equiv \frac{\partial}{\partial x_i}$,

$$\mathbf{Q} = \begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ e \end{pmatrix}, \qquad \text{and} \qquad \mathbf{F}_i = \begin{pmatrix} \rho u_i \\ \rho u_1 u_i + p\delta_{1i} \\ \rho u_2 u_i + p\delta_{2i} \\ \rho u_3 u_i + p\delta_{3i} \\ (e+p)u_i \end{pmatrix}.$$

The symbol δ_{ij} is the Kronecker delta. The Einstein summation convention is used in (3.1) and throughout this thesis, unless noted otherwise. The non-dimensional conservative variables appearing in $\mathbf{Q}$ are the density, ρ, momentum per unit volume, $\rho\mathbf{u} = \rho(u_1, u_2, u_3)^T$, and energy per unit volume, e. The equation of state for a perfect gas leads to an equation relating the pressure to the five flow variables:

$$p = (\gamma - 1)\left(e - \tfrac{1}{2}\rho\mathbf{u}\cdot\mathbf{u}\right),$$

where $\mathbf{u}\cdot\mathbf{u} = \mathbf{u}^T\mathbf{u}$ (the dot product notation is reserved for 3-vectors). The ratio of specific heats is taken to be $\gamma = 1.4$, which is appropriate for the flows of interest here. The variables are non-dimensionalized using a characteristic length scale, l, the free-stream density, ρ_∞, and the free-stream sound speed, a_∞; see [139] for details.

Our objective is to solve for the conservative flow variables, $\mathbf{Q}$, on multi-block structured grids; thus, we consider diffeomorphisms of the following form.

$$\begin{gathered} \mathbf{x} = \boldsymbol{\mathcal{G}}(\boldsymbol{\xi}), \\ \boldsymbol{\mathcal{G}} : D \to P \quad \text{and} \quad \boldsymbol{\mathcal{G}} \in C^m \text{ on } D \setminus \partial D, \\ D = \left\{\boldsymbol{\xi} \in \mathbb{R}^3 | \xi_i \in [0, L_i], i = 1, 2, 3\right\}, \qquad P \subset \mathbb{R}^3. \end{gathered}$$

The hexahedral domain D represents a block in computational space, and P represents the same block in physical space. The smoothness of the transformation is left unspecified, since it must be consistent with the design order of the discretization scheme. For the second-order scheme considered here, we must have $\boldsymbol{\mathcal{G}} \in C^1$. Applying the diffeomorphism, the Euler equations become

$$\partial_t \hat{\mathbf{Q}} + \partial_{\xi_i} \hat{\mathbf{F}}_i = \mathbf{0}, \tag{3.2}$$

where $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)^T = (\xi, \eta, \zeta)^T$,

$$\hat{\mathbf{Q}} = \frac{1}{J}\begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ e \end{pmatrix}, \qquad \text{and} \qquad \hat{\mathbf{F}}_i = \frac{1}{J}\begin{pmatrix} \rho U_i \\ \rho u_1 U_i + p\partial_x \xi_i \\ \rho u_2 U_i + p\partial_y \xi_i \\ \rho u_3 U_i + p\partial_z \xi_i \\ (e+p)U_i \end{pmatrix}.$$

The scalar J is the Jacobian of the mapping, and U_i are the contravariant velocity components, defined by $U_i \equiv u_j \partial_{x_j} \xi_i$.

In the context of this thesis, the transformation $\boldsymbol{\mathcal{G}}$ can be identified with a B-spline mapping; however, this is not necessary in general. For the flow solver described in this chapter, $\boldsymbol{\mathcal{G}}$ is used only in a formal sense to define the transformed Euler equations (3.2). Any quantities that depend on this mapping can be approximated using finite differences.

3.2.2 Boundary Conditions

Boundary conditions must be provided with the Euler equations to obtain a well-posed problem. On the aerodynamic surface, denoted by $\partial\Omega_a$, I specify a flow-tangency condition for the velocity:

$$U_n = \mathbf{u} \cdot \hat{\mathbf{n}} = 0, \quad \forall\, \mathbf{x} \in \partial\Omega_a,$$

where $\hat{\mathbf{n}} = (n_1, n_2, n_3)^T$ is a normal vector to the surface $\partial\Omega_a$, pointing into the domain.

For practical flow simulations, the computational domain Ω must be finite. I will use $\partial\Omega_\infty$ to denote the far-field boundary of the finite domain. The hyperbolic nature of the Euler equations requires careful imposition of boundary conditions on $\partial\Omega_\infty$. For example, specifying boundary conditions for all five flow variables leads to an over-determined problem [72]. To avoid such problems, characteristic boundary conditions are typically applied on the far-field boundaries:

$$\frac{1}{2}\left(|A_n| + A_n\right)\left(\mathbf{Q} - \mathbf{Q}_\infty\right) = 0, \quad \forall\, \mathbf{x} \in \partial\Omega_\infty,$$

where

$$\mathbf{Q}_\infty = \lim_{\|\mathbf{x}\| \to \infty} \mathbf{Q}, \qquad A_n = \frac{\partial \mathbf{F}_i}{\partial \mathbf{Q}} n_i,$$

and $|A_n|$ is equal to A_n with its eigenvalues replaced with their absolute values.

Finally, I consider only geometries that are symmetric with respect to a vertical streamwise plane, which allows the computational domain to be reduced by one half. At the symmetry plane, $\partial\Omega_s$, I require the flow variables to be continuous and the velocity to be reflected about the plane. These conditions imply a flow-tangency boundary condition equivalent to the one used on $\partial\Omega_a$.

3.3 Spatial Discretization

3.3.1 Discretization for One-Dimensional Problems

In an effort to keep the presentation self-contained, I use this section to review SBP operators and SATs applied to one-dimensional problems, specifically constant-coefficient advection and the quasi-one-dimensional Euler equations. For further information on SBP operators and SATs, I direct the reader to the literature on these topics [96, 167, 19, 20, 129, 112, 169].

Summation-By-Parts Operators

The multi-block discretization relies on difference operators that satisfy a discrete summation-by-parts property. These SBP operators are designed to mimic certain properties of differential operators. For example, consider the three-dimensional constant coefficient advection equation:

$$\begin{aligned} \partial_t u + \boldsymbol{\nabla} \cdot (\mathbf{a}u) = 0, \qquad & \forall\, \mathbf{x} \in \Omega \\ u = u_{\text{bc}}, \quad & \forall\, \mathbf{x} \in \partial\Omega_{\text{in}}, \end{aligned}$$

where $\mathbf{a}$ is the advection velocity, and $\partial\Omega_{\text{in}} = \{\partial\Omega \mid \mathbf{a} \cdot \hat{\mathbf{n}} > 0\}$. Taking the inner product of u with the PDE we find

$$\begin{aligned} \partial_t \iiint_{\Omega} \frac{1}{2}u^2 \, dV &= -\iiint_{\Omega} u\boldsymbol{\nabla} \cdot (\mathbf{a}u) \, dV \\ &= \iint_{\partial\Omega_{\text{in}}} \frac{1}{2}u_{\text{bc}}^2 \, (\mathbf{a} \cdot \hat{\mathbf{n}}) \, dA + \iint_{\partial\Omega \setminus \partial\Omega_{\text{in}}} \frac{1}{2}u^2 \, (\mathbf{a} \cdot \hat{\mathbf{n}}) \, dA, \end{aligned} \tag{3.3}$$

where the sign change in the last line is due to the definition of $\hat{\mathbf{n}}$. Thus, the L^2 norm of u can only increase due to boundary contributions. This is a very important stability

property of the continuous system — it guarantees a unique and bounded solution. This is the motivation behind the mimetic properties of SBP operators.

The difference operator $H^{-1}Q$ is an SBP operator if it satisfies the following three properties [96]:

1. The vector of discrete derivatives, $\mathbf{u}_x$, has the form

$$H\mathbf{u}_x - Q\mathbf{u} = \mathbf{0},$$
$$H\partial_x\mathbf{v} - Q\mathbf{v} = H\mathbf{T},$$

   where $\mathbf{u}$ is the discrete solution vector, $\mathbf{v}$ is the exact solution evaluated at the node locations, and $\mathbf{T}$ is the truncation error. For an m^{th} order operator, the truncation error satisfies $\|\mathbf{T}\| = \mathrm{O}(\Delta x^m)$. This form of discrete first derivative encompasses both compact and non-compact schemes.

2. The matrix H is symmetric positive definite. In particular,

$$c\Delta x \leq \lambda(H) \leq C\Delta x$$

   where $\lambda(H)$ denotes an eigenvalue of H, and c, C are real, positive constants independent of the number of grid points.

3. The matrix Q is nearly skew-symmetric in the sense that $Q + Q^T = D$, where D is a diagonal matrix of the form $D = \mathrm{diag}(-1, 0, \ldots, 0, 1)$. Furthermore, $Q_{0,0} = -\frac{1}{2}$ and $Q_{N,N} = \frac{1}{2}$.

The SBP definition encompasses operators of all orders; however, I consider only second-order operators in the present work. The standard second-order centered-difference operator with first-order differences at the boundaries is an SBP operator if we define

$$Q = \frac{1}{2}\begin{bmatrix} -1 & 1 & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ & & & & -1 & 1 \end{bmatrix}, \qquad H = \begin{bmatrix} \Delta x_0 & & & \\ & \Delta x_1 & & \\ & & \ddots & \\ & & & \Delta x_N \end{bmatrix},$$

where $\Delta x_0 = (x_1 - x_0)/2$, $\Delta x_N = (x_N - x_{N-1})/2$, and $\Delta x_i = (x_{i+1} - x_{i-1})/2$, $i = 2, 3, \ldots, N-1$. In the subsequent analysis, we shall see how the SBP definition allows this operator to produce linearly-stable discretizations that mimic the result in (3.3).

Figure 3.1: Example domain consisting of two subdomains with an interface at $x_s = x_{s+1}$.

Constant-Coefficient Advection

By construction, the SBP operators approximate the derivative and do not contain any boundary information. In fact, the SBP matrix operators are singular, and they must be augmented with boundary data to form well-posed problems. This is entirely consistent with continuous PDE systems, which are also ill-posed in the absence of boundary conditions. This leads to a question: how do we impose the boundary conditions without destroying the properties of the SBP operators? Mattsson [111] investigated several approaches of imposing boundary conditions with SBP operators, including injection, simultaneous-approximation terms (SATs) [19], projection [130, 131], and modified projection [57]. His results suggest that the SAT approach is the best method of imposing boundary conditions with SBP operators.

The SAT method adds a penalty term that forces the solution at the boundary or interface toward the desired value. To introduce the method, I will use the constant-coefficient one-dimensional linear advection problem

$$\partial_t u + a\partial_x u = 0, \tag{3.4}$$

where a is the advection speed. Equation (3.4) is discretized on the mesh

$$\mathbf{x} = (x_0, x_1, \ldots, x_s, x_{s+1}, \ldots, x_N)^T$$

with $N + 1$ points and an interface at $x_s = x_{s+1}$; see Figure 3.1. For simplicity, we will apply the SATs at the interface only. The application of SATs for the boundary conditions is similar [19].

Let the discrete solutions on the left and right domains be $\mathbf{u}_L = (u_0, u_1, \ldots, u_s)^T$ and $\mathbf{u}_R = (u_{s+1}, u_{s+2}, \ldots, u_N)^T$, respectively. The systems to be solved on each domain are

$$H_L \partial_t \mathbf{u}_L + a Q_L \mathbf{u}_L = \sigma_1 (u_s - u_{s+1}) \mathbf{e}_L, \tag{3.5}$$

$$H_R \partial_t \mathbf{u}_R + a Q_R \mathbf{u}_R = \sigma_2 (u_{s+1} - u_s) \mathbf{e}_R. \tag{3.6}$$

Note that the unit vectors $\mathbf{e}_L = (0, 0, \ldots, 1)^T$ and $\mathbf{e}_R = (1, 0, \ldots, 0)^T$ have lengths $(s+1)$ and $(N-s)$, respectively.

The terms appearing on the right sides of (3.5) and (3.6) are the penalty terms that couple the two domains. For smooth solutions they introduce no truncation errors since the points coincide. Therefore, while the discrete solution may develop a discontinuity at the interface, this will not affect the global order of accuracy. As an aside, if we were applying a boundary condition at x_s instead of an interface condition, u_{s+1} would be replaced with u_{bc} in (3.5), where u_{bc} is the appropriate boundary value of u.

The coefficients σ_1 and σ_2 are determined using time-stability and conservation arguments. First, consider conservation. Premultiplying (3.5) by the constant vector $\mathbf{1} = (1, 1, \ldots, 1)^T$, and ignoring boundary contributions, we obtain

$$\begin{aligned}
\mathbf{1}^T \mathsf{H}_L \partial_t \mathbf{u}_L + a\mathbf{1}^T \mathsf{Q}_L \mathbf{u}_L &= \sigma_1 (u_s - u_{s+1}) \mathbf{1}^T \mathbf{e}_L \\
\frac{d}{dt}(\mathbf{1}^T \mathsf{H}_L \mathbf{u}_L) + a\mathbf{1}^T (\mathsf{D}_L - \mathsf{Q}_L^T) \mathbf{u}_L &= \sigma_1 (u_s - u_{s+1}) \\
\frac{d}{dt}(\mathbf{1}^T \mathsf{H}_L \mathbf{u}_L) - a\mathbf{u}_L^T \mathsf{Q}_L \mathbf{1} &= \sigma_1 (u_s - u_{s+1}) - a u_s \\
\frac{d}{dt}(\mathbf{1}^T \mathsf{H}_L \mathbf{u}_L) &= \sigma_1 (u_s - u_{s+1}) - a u_s.
\end{aligned}$$

Note that $\mathbf{u}_L^T \mathsf{Q}_L \mathbf{1} = 0$, since the constant vector is in the null space of Q_L. Adding a similar expression for the right domain we find

$$\frac{d}{dt}(\mathbf{1}^T \mathsf{H}_L \mathbf{u}_L + \mathbf{1}^T \mathsf{H}_R \mathbf{u}_R) = (u_s - u_{s+1})(\sigma_1 - \sigma_2 - a). \tag{3.7}$$

We want the right-hand side of (3.7) to vanish for global conservation. In general, u_s and u_{s+1} will not have the same value; thus, to ensure conservation on the whole domain, (3.7) implies

$$\sigma_2 = \sigma_1 - a. \tag{3.8}$$

The relation (3.8) has also been shown to hold for nonlinear fluxes using a weak form of conservation [20].

To further fix the penalty parameters, a time-stability requirement is imposed. Consider the time evolution of the norm $\|\mathbf{u}\|_H^2 = \mathbf{u}_L^T \mathsf{H}_L \mathbf{u}_L + \mathbf{u}_R^T \mathsf{H}_R \mathbf{u}_R$. Premultiplying (3.5) from the

left by $\mathbf{u}_L^T$, and adding the transpose, we find

$$\begin{aligned}\frac{d}{dt}(\mathbf{u}_L^T \mathsf{H}_L \mathbf{u}_L) + a\mathbf{u}_L^T(\mathsf{Q}_L + \mathsf{Q}_L^T)\mathbf{u}_L &= 2\sigma_1(u_s^2 - u_s u_{s+1}) \\ \frac{d}{dt}(\mathbf{u}_L^T \mathsf{H}_L \mathbf{u}_L) + a\mathbf{u}_L^T \mathsf{D}_L \mathbf{u}_L &= 2\sigma_1(u_s^2 - u_s u_{s+1}) \\ \frac{d}{dt}(\mathbf{u}_L^T \mathsf{H}_L \mathbf{u}_L) &= 2\sigma_1(u_s^2 - u_s u_{s+1}) - a u_s^2\end{aligned}$$

Adding a similar expression for the right domain, we obtain the following energy estimate:

$$\frac{d}{dt}\|\mathbf{u}\|_{\mathsf{H}}^2 = \begin{pmatrix} u_s & u_{s+1} \end{pmatrix} \begin{bmatrix} 2\sigma_1 - a & -(\sigma_1 + \sigma_2) \\ -(\sigma_1 + \sigma_2) & 2\sigma_2 + a \end{bmatrix} \begin{pmatrix} u_s \\ u_{s+1} \end{pmatrix}. \tag{3.9}$$

The eigenvalues of the symmetric matrix on the right-hand-side of (3.9) are $\lambda_1 = 0$ and $\lambda_2 = -2a + 4\sigma_1$. The norm will not grow with time if $\lambda_2 \leq 0$, which implies $\sigma_1 \leq \frac{a}{2}$. To satisfy this stability requirement, as well as the conservation requirement (3.8), I tentatively adopt the following values for σ_1 and σ_2:

$$\begin{aligned}\sigma_1 &= -\frac{1}{2}\left[|a| - a\right], \\ \sigma_2 &= -\frac{1}{2}\left[|a| + a\right],\end{aligned} \tag{3.10}$$

This choice for σ_1 and σ_2 deactivates the SAT penalty for the upwind domain, and activates it for the downwind domain.

An additional modification is necessary for spatially varying advection velocities. If we use (3.10) in its current form, both penalty parameters vanish if $a = 0$. This can cause problems if the interface is located at a sonic or stagnation point; thus, in analogy with Swanson and Turkel's matrix dissipation model [170], I propose limiting the penalty parameters as follows:

$$\begin{aligned}\sigma_1 &= -\frac{1}{2}\left[\max\left(|a|, V\right) - a\right], \\ \sigma_2 &= -\frac{1}{2}\left[\max\left(|a|, V\right) + a\right],\end{aligned} \tag{3.11}$$

where $V > 0$ is a constant. These penalty parameters satisfy the conservation requirement (3.8). What about time-stability? If $|a| \geq V$, then $\lambda_2 = -2|a| < 0$ as required. Moreover, if $|a| < V$ then $\lambda_2 = -2V < 0$, so the penalty parameters defined by (3.11) ensure stability.

The Quasi-One-Dimensional Euler Equations

In this section, I review the use of SBP operators and SATs in discretizing the Euler equations. Consider the (transformed) quasi-one-dimensional Euler equations applied to a converging-diverging nozzle:

$$\partial_t \hat{\mathbf{Q}} + \partial_\xi \hat{\mathbf{F}} - \hat{\mathbf{G}} = \mathbf{0}, \tag{3.12}$$

where

$$\hat{\mathbf{Q}} = \frac{1}{J}\begin{pmatrix} \rho S \\ \rho u S \\ \rho E S \end{pmatrix}, \qquad \hat{\mathbf{F}} = \frac{1}{J}\begin{pmatrix} \xi_x \rho u S \\ \xi_x(\rho u^2 + p) S \\ \xi_x \rho u H S \end{pmatrix}, \qquad \hat{\mathbf{G}} = \frac{1}{J}\begin{pmatrix} 0 \\ p \partial_x S \\ 0 \end{pmatrix},$$

$J = \xi_x = (x_\xi)^{-1}$ is the metric Jacobian, $H = (e+p)/\rho$ is the enthalpy, and S is the nozzle area.

Suppose the one-dimensional domain is divided into two subdomains. As before, assume the grid points are located at $\mathbf{x} = (x_0, x_1, \ldots, x_s, x_{s+1}, \ldots, x_N)^T$, and let $x_s = x_{s+1}$ define the interface between the two subdomains. The semi-discrete form of (3.12) becomes

$$\begin{aligned} (H_L \otimes I_3)\partial_t \hat{\mathbf{q}}_L + (Q_L \otimes I_3)\hat{\mathbf{f}}_L - (H_L \otimes I_3)\hat{\mathbf{g}}_L &= \mathbf{\Sigma}_L, \\ (H_R \otimes I_3)\partial_t \hat{\mathbf{q}}_R + (Q_R \otimes I_3)\hat{\mathbf{f}}_R - (H_R \otimes I_3)\hat{\mathbf{g}}_R &= \mathbf{\Sigma}_R, \end{aligned} \tag{3.13}$$

where I_3 is the 3×3 identity matrix and

$$\begin{aligned} \hat{\mathbf{q}}_L &= (\hat{\mathbf{Q}}_0^T, \hat{\mathbf{Q}}_1^T, \ldots, \hat{\mathbf{Q}}_s^T)^T, & \hat{\mathbf{q}}_R &= (\hat{\mathbf{Q}}_{s+1}^T, \hat{\mathbf{Q}}_{s+2}^T, \ldots, \hat{\mathbf{Q}}_N^T)^T, \\ \hat{\mathbf{f}}_L &= (\hat{\mathbf{F}}_0^T, \hat{\mathbf{F}}_1^T, \ldots, \hat{\mathbf{F}}_s^T)^T, & \hat{\mathbf{f}}_R &= (\hat{\mathbf{F}}_{s+1}^T, \hat{\mathbf{F}}_{s+2}^T, \ldots, \hat{\mathbf{F}}_N^T)^T, \\ \hat{\mathbf{g}}_L &= (\hat{\mathbf{G}}_0^T, \hat{\mathbf{G}}_1^T, \ldots, \hat{\mathbf{G}}_s^T)^T, & \hat{\mathbf{g}}_R &= (\hat{\mathbf{G}}_{s+1}^T, \hat{\mathbf{G}}_{s+2}^T, \ldots, \hat{\mathbf{G}}_N^T)^T \end{aligned}$$

The subscripts indicate the location of the variable or flux (e.g. $\hat{\mathbf{Q}}_i = \hat{\mathbf{Q}}(x_i)$). The binary operator $\otimes$ denotes the Kronecker product for matrices: if $A \in M^{m\times n}$ and $B \in M^{p\times q}$ then $C = A \otimes B \in M^{mp\times nq}$, is defined by $C_{p(i-1)+k,q(j-1)+l} = A_{ij}B_{kl}$. In matrix form,

$$C = A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}.$$

I highlight the following properties of the Kronecker product, which are useful for manipulating SBP-discretized systems:

$$\begin{aligned} (A \otimes B)(C \otimes D) &= (AC \otimes BD), \\ (A \otimes B)^T &= (A^T \otimes B^T), \end{aligned}$$

and

$$(A \otimes B)^{-1} = (A^{-1} \otimes B^{-1}).$$

The penalty terms in (3.13) are $3(s+1)$ and $3(N-s)$ column vectors given by

$$\mathbf{\Sigma}_L = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ -\frac{1}{2}(|\hat{A}| - \hat{A})(\hat{\mathbf{Q}}_s - \hat{\mathbf{Q}}_{s+1}) \end{pmatrix}, \qquad \mathbf{\Sigma}_R = \begin{pmatrix} -\frac{1}{2}(|\hat{A}| + \hat{A})(\hat{\mathbf{Q}}_{s+1} - \hat{\mathbf{Q}}_s) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix},$$

where $\hat{A} = \partial\hat{\mathbf{F}}/\partial\hat{\mathbf{Q}}$ is the flux Jacobian matrix at an averaged state; for the present work I use the simple average $\frac{1}{2}(\mathbf{Q}_s + \mathbf{Q}_{s+1})$. The matrix $|\hat{A}| = X|\hat{\Lambda}|X^{-1}$ where X denotes the right eigenvectors of $\hat{A}$, and

$$|\hat{\Lambda}| = \begin{bmatrix} \hat{\lambda}_1 & 0 & 0 \\ 0 & \hat{\lambda}_2 & 0 \\ 0 & 0 & \hat{\lambda}_3 \end{bmatrix},$$

where

$$\begin{aligned} \hat{\lambda}_1 &= \max\left(\xi_x|u+a|, V_n\rho(\hat{A})\right), \\ \hat{\lambda}_2 &= \max\left(\xi_x|u-a|, V_n\rho(\hat{A})\right), \\ \hat{\lambda}_3 &= \max\left(\xi_x|u|, V_l\rho(\hat{A})\right), \end{aligned}$$

and $\rho(\hat{A})$ denotes the spectral radius of $\hat{A}$. The constants V_n and V_l are used to scale the spectral radius. For subsonic flows $V_n = \frac{1}{40}$ and for transonic flows $V_n = \frac{1}{4}$. The constant V_l is fixed at $\frac{1}{40}$ for all flows. These values are based on the matrix dissipation model as implemented by Chisholm [23], but I have found they also work well in the context of SAT penalties.

3.3.2 Numerical Dissipation

Nonlinear hyperbolic PDEs can produce modes whose wavelengths are not resolved by the computational mesh. The energy of these modes must be accounted for, so it is aliased to frequencies resolved by the mesh. Centered-difference operators, which are skew-symmetric,

or nearly skew-symmetric, cannot remove this spurious energy; consequently, high-frequency oscillations can appear in the solution. Centered-difference schemes also suffer from oscillations caused by decoupling of adjacent nodes.

Numerical dissipation can be added to a centered-difference discretization to remove energy from the high-frequency modes and couple adjacent nodes, thus eliminating spurious oscillations. Numerical dissipation can be added either explicitly, using a negative semi-definite operator, or implicitly, using an upwind operator [104]. In the present work I use the scalar dissipation model developed by Jameson *et al.* [85] and refined by Pulliam [139]. I elaborate on the dissipation model in Appendix A.1.

Mattsson *et al.* [112] have shown that a dissipation model based on the classical difference operators does not produce an energy estimate; however, an energy estimate can be obtained if the diagonal norm H is properly accounted for by the dissipation. For example, if Z denotes the classical dissipation operator, then $H^{-1}Z$ will produce an energy estimate.

3.3.3 Discretization for Three-Dimensional Problems

The one-dimensional SBP-SAT discretization is easily extended to three-dimensional problems using Kronecker products. Let us consider a single block with N_1, N_2, and N_3 nodes in the ξ, η, and ζ directions, respectively. Let $\hat{\mathbf{q}}$ be a block-column vector of the scaled flow variables $\hat{\mathbf{Q}}$, and $\hat{\mathbf{f}}_i$ a block-column vector of the fluxes $\hat{\mathbf{F}}_i$. For simplicity, assume that $\hat{\mathbf{q}}$ and $\hat{\mathbf{f}}_i$ are ordered first by variable and then by the coordinate ξ, then η, and finally ζ. Using this ordering we can define the following three-dimensional operators.

$$
\begin{gathered}
H = H_\zeta \otimes H_\eta \otimes H_\xi \otimes I_5, \\
Q_1 = H_\zeta \otimes H_\eta \otimes Q_\xi \otimes I_5, \qquad Q_2 = H_\zeta \otimes Q_\eta \otimes H_\xi \otimes I_5, \qquad Q_3 = Q_\zeta \otimes H_\eta \otimes H_\xi \otimes I_5, \\
S_1^\pm = H_\zeta \otimes H_\eta \otimes E_\xi^\pm \otimes I_5, \qquad S_2^\pm = H_\zeta \otimes E_\eta^\pm \otimes H_\xi \otimes I_5, \qquad S_3^\pm = E_\zeta^\pm \otimes H_\eta \otimes H_\xi \otimes I_5, \\
Z = H_\zeta \otimes H_\eta \otimes Z_\xi \otimes I_5 + H_\zeta \otimes Z_\eta \otimes H_\xi \otimes I_5 + Z_\zeta \otimes H_\eta \otimes H_\xi \otimes I_5.
\end{gathered}
$$

The subscripts ξ, η, and ζ are used to distinguish one-dimensional operators in the corresponding coordinate direction. For example, H_ξ and Q_ξ define the SBP operator in the ξ-direction, while Z_η is the (nonlinear) dissipation operator in the η-direction. Similarly, $E_{\xi_i}^+$ and $E_{\xi_i}^-$ are one-dimensional operators used to isolate boundary and interface nodes; these $N_i \times N_i$ matrices are defined by $E_{\xi_i}^+ = \mathrm{diag}\,(1, 0, 0, \ldots, 0)$ and $E_{\xi_i}^- = \mathrm{diag}\,(0, \ldots, 0, 0, 1)$. The

matrix I_5 is the 5×5 identity matrix.

Using the above operators, the semi-discretization of the three-dimensional Euler equations for a single block is

$$\partial_t \hat{\mathbf{q}} + \left(H^{-1} Q_i\right) \hat{\mathbf{f}}_i - \left(H^{-1} Z\right) \hat{\mathbf{q}} + \left(H^{-1} \boldsymbol{\Sigma}\right) = \mathbf{0}. \tag{3.14}$$

The SAT terms can be expanded as

$$\boldsymbol{\Sigma} = S_i^+ \boldsymbol{\Delta} \hat{\mathbf{f}}_i^+ + S_i^- \boldsymbol{\Delta} \hat{\mathbf{f}}_i^-,$$

with summation over the coordinate directions i, and where

$$\boldsymbol{\Delta} \hat{\mathbf{f}}_i^{\pm} = \frac{1}{2} \operatorname{diag}\left(|\hat{A}_i| \pm \hat{A}_i\right) \left(\hat{\mathbf{q}} - \hat{\mathbf{q}}_{\mathrm{bc}}\right), \qquad i = 1, 2, 3. \tag{3.15}$$

In (3.15), the diag$(\cdot)$ notation is abused slightly, since it is used in this context to indicate a block diagonal matrix. The flux Jacobians for the 3-dimensional Euler equations are defined by $\hat{A}_i = \partial \hat{\mathbf{F}}_i / \partial \hat{\mathbf{Q}}$. The vector $\hat{\mathbf{q}}_{\mathrm{bc}}$ contains either boundary data, for nodes on boundaries, or neighbouring block variables, for nodes adjacent to another block; this is discussed further below. Its entries corresponding to interior nodes are not important (they are multiplied by zero entries in $S_i^{\pm}$), and can be set to zero.

Numerical Boundary Data

Appropriate boundary or interface data must be supplied to the SAT penalties in (3.15). For a node on a block interface, the corresponding entry in $\hat{\mathbf{q}}_{\mathrm{bc}}$ is set to the variable value of the coincident node on the adjacent block. In addition, the flux Jacobian matrices $\hat{A}_i$ and $|\hat{A}_i|$ are evaluated using a simple average of the two states. This is equivalent to the procedure described for the one-dimensional SATs.

For a node coinciding with a boundary, the data in $\hat{\mathbf{q}}_{\mathrm{bc}}$ must be consistent with the prescribed boundary conditions. At far-field boundaries, the boundary data is simply the Jacobian-scaled free-stream flow variable values:

$$\hat{\mathbf{Q}}_{\mathrm{bc},\infty} = \frac{1}{J} \mathbf{Q}_{\infty}.$$

The flux Jacobian matrices for the far-field SATs are evaluated using the boundary-node values, rather than the far-field value.

To obtain the boundary state on aerodynamic surfaces, the flow variables are manipulated to penalize non-zero normal velocities and enforce constant enthalpy. In particular, recalling that $H = (e + p)/\rho$,

$$\hat{\mathbf{Q}}_{\mathrm{bc},a} = \frac{1}{J} \begin{pmatrix} \rho \\ \rho u_1^{\shortparallel} \\ \rho u_2^{\shortparallel} \\ \rho u_3^{\shortparallel} \\ \rho H_\infty - p \end{pmatrix},$$

where

$$\left(u_1^{\shortparallel}, u_2^{\shortparallel}, u_3^{\shortparallel}\right)^T = \mathbf{u}^{\shortparallel} \equiv P\mathbf{u} = (I_3 - \hat{\mathbf{n}}\hat{\mathbf{n}}^T)\mathbf{u}.$$

The 3×3 matrix P projects out the normal component of the velocity at the surface. The unit normal $\hat{\mathbf{n}}$, which defines P, is calculated by normalizing the appropriate metric term, $\boldsymbol{\nabla}\xi_i$, at the node of interest. The flux Jacobian matrices for solid boundary SATs are evaluated using the boundary data; consequently, three of the eigenvalues of $\hat{A}_i$ are zero, consistent with the physical boundary conditions.

SAT penalties for the symmetry plane are obtained using the procedure described for the solid surface boundaries.

Discretization of the Grid Metrics

Recall that the transformed Euler fluxes contain grid metrics, $\boldsymbol{\nabla}\xi_i$, and the Jacobian of the mapping, $J = \frac{\partial(\xi,\eta,\zeta)}{\partial(x,y,z)}$. In general, these geometric quantities must be approximated numerically, since the mapping between (ξ, η, ζ) and (x, y, z) is not explicitly known. Even if the mapping is known, numerical evaluation of the metrics is preferable, since it can decrease the truncation error [110].

The metrics for fixed grids can be represented using the derivatives of spatial coordinates with respect to the computational coordinates as follows [139, 123]:

$$(\xi_i)_{x_l} = \frac{\partial \xi_i}{\partial x_l} = J\left(\frac{\partial x_m}{\partial \xi_j}\frac{\partial x_n}{\partial \xi_k} - \frac{\partial x_n}{\partial \xi_j}\frac{\partial x_m}{\partial \xi_k}\right) \tag{3.16}$$

where $i, l \in \{1, 2, 3\}$ and

$$j = \mathrm{mod}\,(i, 3) + 1, \quad k = \mathrm{mod}\,(i + 1, 3) + 1,$$
$$m = \mathrm{mod}\,(l, 3) + 1, \quad n = \mathrm{mod}\,(l + 1, 3) + 1.$$

The partial derivatives on the right-hand side of (3.16) are much easier to approximate than the grid metrics. For example, we could use an SBP operator to discretize $\partial x/\partial \xi$.

If we substitute the free-stream flow variables into the transformed Euler equations (3.2), we expect that the PDE will be satisfied. This is indeed the case, but the result relies on the commutative property of analytical differentiation [139]. In general, discrete-derivative operators do not commute, so the free-steam flow may not satisfy the discretized Euler equations (3.14).

Pulliam and Steger [140] proposed a grid metric discretization that ensures the free-stream is satisfied for second-order centered-differences. Their scheme involves averaging the difference approximations. For example, the terms y_η and z_ζ appearing in ξ_x are approximated as follows:

$$\mathbf{y}_\eta \approx \left[W_\zeta \otimes \left(H_\eta^{-1} Q_\eta\right) \otimes I_\xi\right] \mathbf{y},$$
$$\mathbf{z}_\zeta \approx \left[\left(H_\zeta^{-1} Q_\zeta\right) \otimes W_\eta \otimes I_\xi\right] \mathbf{z},$$

where $\mathbf{y}$ and $\mathbf{z}$ are vectors of the y and z nodal coordinates on the block, and the one-dimensional averaging operators are of the form

$$W = \frac{1}{2} \begin{bmatrix} 1 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 1 \end{bmatrix}.$$

The metrics that result from this averaging procedure are closely related to area-weighted face normals in a finite-volume scheme, which explains why this approach satisfies the free-stream. I use the Pulliam-Steger metric calculation for the results presented in this thesis.

The Pulliam-Steger metrics have not been extended to higher-order discretizations; therefore, I conclude this section by noting an alternative metric calculation, proposed by Thomas and Lombard [174], which guarantees that the free-stream is satisfied for higher-order explicit and Padé discretizations [73].

3.4 Solution Method

3.4.1 Newton-Krylov Approach

The discrete Euler equations are a set of nonlinear algebraic equations, represented here by the vector equation

$$\boldsymbol{\mathcal{R}}(\mathbf{q}) = \mathbf{0}, \tag{3.17}$$

where $\mathbf{q}$ is a block-column vector of the conservative flow variables. The vector $\boldsymbol{\mathcal{R}}$ is called the nonlinear residual.

Applying Newton's method to the discrete equations (3.17) we obtain the following linear system for each (outer) iteration n:

$$A^{(n)}\Delta\mathbf{q}^{(n)} = -\boldsymbol{\mathcal{R}}^{(n)}, \tag{3.18}$$

where $\boldsymbol{\mathcal{R}}^{(n)} = \boldsymbol{\mathcal{R}}(\mathbf{q}^{(n)})$, $\Delta\mathbf{q}^{(n)} = \mathbf{q}^{(n+1)} - \mathbf{q}^{(n)}$, and

$$A_{ij}^{(n)} = \frac{\partial \mathcal{R}_i}{\partial q_j}\left(\mathbf{q}^{(n)}\right).$$

Newton's method will converge quadratically in n provided that $A^{(n)}$ is nonsingular and the initial iterate, $\mathbf{q}^{(0)}$, is sufficiently close to the solution of (3.17) [90]. As is well known, finding a suitable initial iterate for Newton's method can be difficult; thus, I break the algorithm into two phases.

1. An approximate-Newton, or start-up, phase whose objective is to find a suitable initial iterate as efficiently as possible.

2. An inexact-Newton phase, which uses the initial iterate and a slightly modified form of (3.18).

Both of these phases include a number of parameters. In most cases, these parameters are bounded by robustness, on the one hand, and CPU time, on the other. The parameter values I present below have been validated on a range of cases and provide a good compromise between these two objectives. For the algorithm assessment in Section 3.6 the parameters are fixed unless stated otherwise.

Approximate-Newton Phase

The approximate-Newton phase uses a form of pseudo-transient continuation to find the initial iterate [139, 91, 95]. This strategy is similar to discretizing the unsteady equations (3.1) using implicit Euler in time; however, since we are seeking an initial iterate for Newton's method, and not a time accurate solution, there are several important modifications that we can make to the implicit Euler scheme. These modifications include a first-order Jacobian matrix, a lagged Jacobian update, and a spatially varying time step.

A first-order Jacobian matrix can be effective during start-up [91, 122], and is obtained here by eliminating the fourth-difference dissipation terms from $\mathsf{A}^{(n)}$ and increasing the coefficient for the second-difference dissipation. Let κ_4 and κ_2 denote the fourth- and second-difference dissipation coefficients used in the discrete equations, $\boldsymbol{\mathcal{R}}$, and let $\tilde{\kappa}_4$ and $\tilde{\kappa}_2$ denote the corresponding coefficients used in the modified Jacobian matrix during start-up. Then,

$$\tilde{\kappa}_4 = 0, \qquad \tilde{\kappa}_2 = \kappa_2 + \sigma\kappa_4$$

Previous work suggests that the optimal value for the lumping coefficient σ is between 4 and 6 for three-dimensional inviscid flows [122]. Lumping the dissipation coefficients in this way produces a modified Jacobian that is only first-order accurate, but this does not affect the accuracy of the steady solution. I will use A_1 to denote the first-order Jacobian, to emphasize its order of accuracy and distinguish it from the exact Jacobian.

The first-order Jacobian is factored using an incomplete lower/upper factorization to produce the preconditioner. Factoring the matrix is one of the most expensive tasks required by the algorithm. The approximate-Newton phase often requires many outer iterations, so the cost of the factorization can be particularly acute if it is performed each iteration. This suggests that we update and factor the first-order Jacobian periodically rather than at every iteration, an idea first proposed by Kim and Orkwis [93]. Let m be the number of outer iterations between Jacobian updates. Then, A_1 is updated and factored on iteration n if $\mathrm{mod}(n, m) = 0$. Values $m = \{3, 4, 5\}$ provide a good compromise between CPU time and robustness [66].

Finally, a spatially varying time step has been shown to improve the rate of convergence for schemes based on approximate factorizations [139] as well as Newton-Krylov algorithms

[122, 23]. The time step used in the current work is, for node (j,k,m),

$$\Delta t^{(n)}_{j,k,m} = \frac{\Delta t^{(n)}_{\text{ref}}}{J_{j,k,m}(1+\sqrt[3]{J_{j,k,m}})}. \tag{3.19}$$

This time step roughly approximates a constant CFL [139]. The appearance of the first J in the denominator of (3.19) is due to the use of $\mathbf{Q}$ rather than $\hat{\mathbf{Q}}$ in the column vector $\mathbf{q}$; see (3.2). The reference time step is steadily increased according to the geometric formula

$$\Delta t^{(n)}_{\text{ref}} = a(b)^{m\lfloor \frac{n}{m} \rfloor},$$

where $\lfloor \cdot \rfloor$ is the floor operator[1]. This operator ensures that updates to $\Delta t^{(n)}_{\text{ref}}$ are consistent with the update period m. Values for a and b used in the present work are $a = 0.1$ and $b \in [1.4, 1.7]$.

To summarize, during start-up I replace (3.18) with the approximate-Newton update equation

$$\tilde{A}^{(n)} \boldsymbol{\Delta q}^{(n)} = -\boldsymbol{\mathcal{R}}^{(n)}, \tag{3.20}$$

where

$$\tilde{A}^{(n)} \equiv \mathsf{T}^{(n^*)} + \mathsf{A}_1^{(n^*)},$$

$n^* = m\lfloor n/m \rfloor$, and $\mathsf{T}^{(n)}$ is a diagonal matrix containing the (inverse) local time steps appropriate to each equation. Finally, note that the update equation (3.20) is not solved exactly, but rather inexactly to a relative tolerance of 0.5 using a Krylov iterative solver. The solution of the linear system is discussed further in Section 3.4.2.

Switching Between Phases

The algorithm should switch to the inexact-Newton phase as soon as a suitable initial (Newton) iterate has been obtained; thus, we must determine what qualifies as a suitable initial "guess." Several authors have suggested switching when the nonlinear relative residual is reduced below a certain threshold [122, 23]:

$$\frac{\|\boldsymbol{\mathcal{R}}^{(n)}\|_2}{\|\boldsymbol{\mathcal{R}}^{(0)}\|_2} \equiv R_d^{(n)} \leq \tau.$$

[1] $\lfloor x \rfloor$ gives the largest integer less than or equal to x

For the Euler equations, $\tau = 0.1$ is usually sufficient and has been used for all the results presented in this thesis.

Inexact-Newton Phase

As with the start-up phase, the inexact-Newton phase uses a diagonal matrix of spatially varying time-steps. The reference time step during the Newton phase is based on the successive evolution relaxation method of Mulder and van Leer [117]:

$$\Delta t_{\text{ref}}^{(n)} = \max\left[\alpha\left(R_d^{(n)}\right)^{-\beta}, \Delta t_{\text{ref}}^{(n-1)}\right],$$

where, for the present results, I have used $\beta = 2$. The max function ensures that $\Delta t_{\text{ref}}^{(n)}$ is nondecreasing. I have found this safeguard useful in transonic flows where the residual norm may increase temporarily during convergence. The value of α is calculated to avoid an abrupt change between the approximate Newton and inexact-Newton time steps. Specifically, if n_{Newt} is the first inexact-Newton iteration, then

$$\alpha = a(b)^{m\lfloor\frac{n_{\text{Newt}}}{m}\rfloor}\left(R_d^{(n_{\text{Newt}})}\right)^{\beta}.$$

Each outer iteration during the inexact-Newton phase produces the following linear system:

$$\left(\mathsf{T}^{(n)} + \mathsf{A}^{(n)}\right)\boldsymbol{\Delta}\mathbf{q}^{(n)} = -\boldsymbol{\mathcal{R}}^{(n)}, \tag{3.21}$$

where, as before, $\mathsf{T}^{(n)}$ is a diagonal matrix of inverse time steps. Note that the diagonal time step matrix tends to zero quadratically with $R_d^{(n)}$ due to the choice $\beta = 2$. Unlike the approximate Newton phase, the matrices on the left-hand side of (3.21) are recomputed at each iteration, and the Jacobian matrix $\mathsf{A}^{(n)}$ is not explicitly modified. Indeed, since I use Krylov subspace methods to solve (3.21), only Jacobian-vector products need to be computed. These products can be approximated using a first-order-accurate forward difference:

$$\mathsf{A}^{(n)}\mathbf{v} \approx \frac{\boldsymbol{\mathcal{R}}(\mathbf{q}^n + \epsilon\mathbf{v}) - \boldsymbol{\mathcal{R}}(\mathbf{q}^n)}{\epsilon}. \tag{3.22}$$

The perturbation parameter must be chosen carefully to minimize truncation errors and avoid round-off errors [188]. For this work I have used [126]

$$\epsilon = \sqrt{\frac{N\delta}{\mathbf{v}^T\mathbf{v}}},$$

where $\delta = 10^{-13}$, and N is the number of unknowns. In light of the approximation (3.22), I do not need to compute or store the Jacobian matrix[2]; however, I do need to compute and store the first-order Jacobian A_1 at the beginning of each inexact-Newton iteration, since this matrix is needed to build the preconditioner.

The inexact-Newton algorithm does not solve (3.21) exactly, but rather to a certain relative tolerance:

$$\left\| \boldsymbol{\mathcal{R}}^{(n)} + \left(T^{(n)} + A^{(n)} \right) \boldsymbol{\Delta}\mathbf{q}^{(n)} \right\|_2 \leq \eta_n \|\boldsymbol{\mathcal{R}}^{(n)}\|_2.$$

The forcing parameter $\eta_n \in [0, 1)$ controls the accuracy of the solution update $\boldsymbol{\Delta}\mathbf{q}^{(n)}$ and the convergence rate of the inexact-Newton method. If η_n is too small, we obtain quadratic convergence at the expense of over-solving the linear system. If η_n is too large, the linear system will be cheap to solve, but the number of outer iterations will increase.

For this work, the forcing parameter is gradually decreased from its start-up value of 0.5 to 0.01 using a safeguard proposed by Eisenstat and Walker [35]:

$$\eta_n = \max \left\{ 0.01, \eta_{n-1}^{(1+\sqrt{5})/2} \right\}. \tag{3.23}$$

I have found that formula (3.23) helps avoid over-solving during the early iterations of the inexact-Newton phase. Eisenstat and Walker also give an adaptive formula for the forcing parameter that allows q-superlinear convergence [35]. While their adaptive η formula reduces the number of Newton iterations, the CPU and memory costs do not warrant the additional work needed to satisfy the smaller tolerances.

3.4.2 Solving The Distributed Linear System

During both the start-up and Newton phases I use a Krylov solver — for example, the generalized minimal residual method (GMRES) [152] — to inexactly solve sparse systems of the form[3]

$$A\mathbf{x} = \mathbf{b}. \tag{3.24}$$

[2]The Jacobian matrix is not needed for flow solution algorithm, but it is needed for the optimization algorithm.

[3]The notation used is this section has been adopted to be consistent with notation used in the matrix analysis and linear algebra literature. Hence, the generic vector $\mathbf{x}$ should not be confused with the 3-vector of Cartesian coordinates, nor should the generic right-hand-side vector $\mathbf{b}$ be confused with the vector of B-spline control points, etc.

To solve the above equation in parallel, the unknowns, $\mathbf{x}$, and their corresponding equations are assigned to unique processes according to some domain decomposition; for the present work one or more blocks are assigned to a process. For a given process, i, three types of unknowns can be identified for the linear system (3.24):

1. Internal unknowns are assigned to the process i and are coupled only to other variables assigned to i.

2. Internal-interface unknowns are assigned to process i but are coupled to unknowns on another process $j \neq i$.

3. External-interface unknowns are assigned to a subdomain $j \neq i$ but appear in equations on process i.

For example, when the Euler equations are discretized using SATs, the internal- and external-interface unknowns correspond to nodes that are coincident for adjacent blocks.

If the unknowns and equations are grouped (i.e. ordered) by subdomain, then we can write the equations corresponding to process i as

$$A_i \mathbf{x}_{(i)} + E_i \mathbf{y}_{(i,\text{ext})} = \mathbf{b}_{(i)}, \tag{3.25}$$

where $\mathbf{x}_{(i)}$ and $\mathbf{b}_{(i)}$ denote the unknowns and right-hand-sides assigned to process i, and $\mathbf{y}_{(i,\text{ext})}$ are the external-interface unknowns coupled with unknowns on process i. With this grouping, the global linear system for four subdomains has the structure shown in Figure 3.2. Note that internal-interface unknowns are ordered last in each subdomain. This convention allows more efficient interprocessor communication, and reduced local indirect addressing during matrix-vector multiplication [153].

With the internal-interface unknowns ordered last, we obtain the (local) partitioning

$$\mathbf{x}_{(i)} = \begin{pmatrix} \mathbf{u}_{(i)} \\ \mathbf{y}_{(i)} \end{pmatrix}, \qquad \mathbf{b}_{(i)} = \begin{pmatrix} \mathbf{f}_{(i)} \\ \mathbf{g}_{(i)} \end{pmatrix},$$

where $\mathbf{u}_{(i)}$ are the local internal variables, and $\mathbf{y}_{(i)}$ are the local internal-interface variables. The subvectors $\mathbf{f}_{(i)}$ and $\mathbf{g}_{(i)}$ are the analogous partitions of $\mathbf{b}_{(i)}$. Hence, the local equations (3.25) on process i take the form

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} \mathbf{u}_{(i)} \\ \mathbf{y}_{(i)} \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} \mathbf{y}_{(j)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{(i)} \\ \mathbf{g}_{(i)} \end{pmatrix}. \tag{3.26}$$

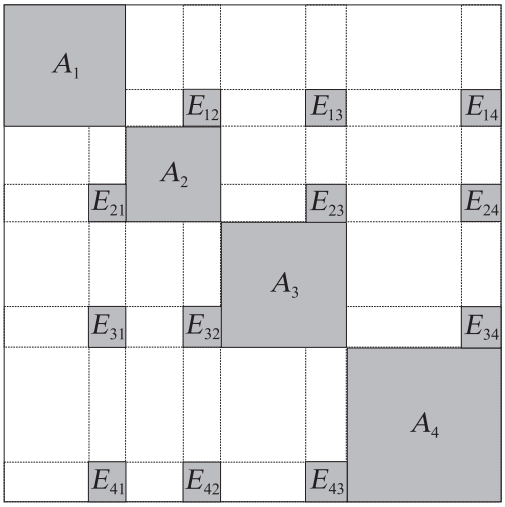


Figure 3.2: The sparsity structure of the global system matrix with unknowns grouped by subdomain, and internal-interface variables listed last. Sparse submatrices with non-zero elements are shaded.

The neighbouring subdomains of subdomain i are denoted by the set N_i.

In using a Krylov-subspace method to solve the distributed system (3.24), we must parallelize the inner-products, the matrix-vector products, and the preconditioner. The inner-products are straightforward: they are computed by summing the local products (e.g. $\mathbf{v}_{(i)}^T \mathbf{z}_{(i)}$) using the MPI command `MPI_Allreduce()`. Parallelizing the matrix-vector products and the preconditioner requires more care.

For the matrix-vector products, we see that only internal-interface unknowns are affected by the product $E_i \mathbf{y}_{(i,\text{ext})} = \sum_{j \in N_i} E_{ij} \mathbf{y}_{(j)}$. Thus, communication time can be partially "hidden" by using a non-blocking communication of the interface variables [151]. During the communication, the local matrix is multiplied, and, once the external-interface unknowns are received, the contribution due to $E_i \mathbf{y}_{(i,\text{ext})}$ is calculated and added to $\mathbf{y}_{(i)}$. Note that explicit matrix-vector products are only required for the approximate-Newton stage or for adjoint solves during optimization.

Preconditioners are the most critical component of an iterative parallel linear solver. While excellent serial preconditioners exist for Newton-Krylov flow solvers [137, 12, 122], these preconditioners cannot be implemented efficiently in parallel; for example, while using ILU(p) on the global system matrix, A, has proven to work well in serial, a parallel ver-

sion results in substantial idle time and communication. The challenge in constructing a good parallel preconditioner is balancing the competing objectives of scalability and serial performance.

3.4.3 Preconditioning

I investigated two parallel preconditioners: one based on the additive-Schwarz method and one based on an approximate-Schur method. Both of the underlying methods require an exact or inexact inversion of local submatrices. The preconditioners presented here use an incomplete lower-upper factorization of the local submatrix of the modified Jacobian:

$$L_i U_i = \left[T^{(n)} + A_1^{(n)} \right]_i + R_i,$$

where R_i is the error in the factorization. A block version of incomplete lower-upper factorization with a level of fill p, ILU(p) [114] is used to obtain the factorization $L_i U_i$. The 5×5 blocks correspond with the flow unknowns at each node. Performance issues motivate the choice of Block ILU(p) (BILU(p)) over scalar ILU(p); see Section 3.6.1. Notice that the factorization itself does not require inter-processor communication, since BILU(p) is applied to the local submatrices only. For the remaining sections, the submatrix A_i refers to the modified Jacobian submatrix $[T^{(n)} + A_1^{(n)}]_i$.

Additive-Schwarz Preconditioner

The simplest form of additive-Schwarz preconditioning is essentially a block Jacobi iteration; see, for example, Saad [150]. Given the vector $\mathbf{w}$, the local component of the preconditioned vector $\mathbf{z}$ is given by the exact or inexact solution to the system

$$A_i \mathbf{z}_{(i)} = \mathbf{w}_{(i)}. \tag{3.27}$$

Equation (3.27) can be solved using a direct method or iteratively using, for example, GMRES. The additive-Schwarz preconditioner considered here solves (3.27) approximately using a single application of the ILU factorization: $\mathbf{z}_{(i)} = U_i^{-1} L_i^{-1} \mathbf{w}_{(i)}$. I have investigated the use of preconditioned GMRES to solve (3.27), but solving the local system more accurately was not competitive.

Additive-Schwarz methods can employ overlapping domains to improve the quality of the preconditioner, i.e. reduce the number of Krylov iterations. Numerical experiments by Gropp *et al.* [54] suggest that domain overlap, while capable of reducing the number of Krylov iterations, increases the overall CPU time; hence, I do not consider overlapping as a convergence strategy in this work.

Approximate-Schur Preconditioner

The idea behind Schur complement methods is the elimination of the internal unknowns to form a reduced system of equations, called the Schur-complement system. Saad and Sosonkina [153] have proposed a preconditioning technique based on an approximate factorization of the Schur-complement system. Their contribution is summarized below.

Considering (3.26), we see that the internal variables can be written as

$$\mathbf{u}_{(i)} = B_i^{-1}(\mathbf{f}_{(i)} - F_i\mathbf{y}_{(i)}). \tag{3.28}$$

Substituting $\mathbf{u}_{(i)}$ into the equation for $\mathbf{y}_{(i)}$ we obtain the following system for the internal-interface variables on process i:

$$S_i\mathbf{y}_{(i)} + \sum_{j\in N_i} E_{ij}\mathbf{y}_{(j)} = \mathbf{g}_{(i)} - E_iB_i^{-1}\mathbf{f}_{(i)} \equiv \mathbf{g}'_{(i)}. \tag{3.29}$$

where $S_i = C_i - E_iB_i^{-1}F_i$ is the "local" Schur-complement matrix [153]. Assembling all the local Schur-complement systems for each process, we obtain a linear system for all the internal-interface unknowns:

$$\underbrace{\begin{pmatrix} S_1 & E_{12} & \dots & E_{1P} \\ E_{21} & S_2 & \dots & E_{2P} \\ \vdots & & \ddots & \vdots \\ E_{P1} & E_{P2} & \dots & S_P \end{pmatrix}}_{S} \begin{pmatrix} \mathbf{y}_{(1)} \\ \mathbf{y}_{(2)} \\ \vdots \\ \mathbf{y}_{(P)} \end{pmatrix} = \begin{pmatrix} \mathbf{g}'_{(1)} \\ \mathbf{g}'_{(2)} \\ \vdots \\ \mathbf{g}'_{(P)} \end{pmatrix}. \tag{3.30}$$

The coefficient matrix S, appearing in (3.30), is the Schur complement [150]. Following Saad and Sosonkina, I will refer to this matrix as the global Schur complement to distinguish it from the S_i, which are the diagonal blocks of S.

We could assemble the global Schur-complement matrix, solve the system (3.30), and then solve for the local internal unknowns on each process using (3.28). In practice, however,

forming the Schur-complement matrix and solving for the interface unknowns exactly is not competitive with other methods [153]. Instead, we shall consider systems that approximate (3.30) and act as preconditioners for the global system (3.24).

Consider the following block factorization of A_i;

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} = \begin{pmatrix} B_i & 0 \\ E_i & S_i \end{pmatrix} \begin{pmatrix} I & B_i^{-1}F_i \\ 0 & I \end{pmatrix}. \tag{3.31}$$

Next, suppose A_i has been factored instead into $A_i = L_i U_i$, where

$$L_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix}, \qquad \text{and} \qquad U_i = \begin{pmatrix} U_{B_i} & L_{B_i}^{-1}F_i \\ 0 & U_{S_i} \end{pmatrix}.$$

Comparing the factors in (3.31) with L_i and U_i we can show that [153]

$$S_i = L_{S_i} U_{S_i}. \tag{3.32}$$

Thus we can obtain an LU decomposition of the local Schur complement by extracting the relevant blocks from the LU decomposition of A_i. Similarly, and more relevant to preconditioning, we can obtain an approximate factorization of S_i by extracting the relevant blocks from the ILU factorization of A_i.

Suppose the local matrix has been factored as $A_i = L_i U_i + R_i$. Then we can define the following approximate local-Schur-complement system using (3.32) and (3.29);

$$\mathbf{y}_{(i)} = U_{S_i}^{-1} L_{S_i}^{-1} \left(\mathbf{g}'_{(i)} - \sum_{j \in N_i} E_{ij} \mathbf{y}_{(j)} \right). \tag{3.33}$$

The above equation is a single iteration of block Jacobi on the local internal-interface unknowns. This system can be further accelerated using a Krylov-subspace method such as GMRES [150]. Once the approximations to the $\mathbf{y}_{(i)}$ have been exchanged, we can substitute the $\mathbf{y}_{(i,\text{ext})}$ into (3.25), and apply the ILU factorization to obtain approximate values for $\mathbf{x}_{(i)}$.

I have implemented an approximate-Schur preconditioner, which is closely based on algorithm 3.1 of Saad and Sosonkina [153]. Two important clarifications to the original algorithm involve the complete forward and backward solves, $(L_i U_i)^{-1}$, on lines 5 and 25 of their algorithm. These forward-backward solves are modified with partial operations involving U_{S_i} and L_{S_i}, such that the proposed algorithm is mathematically equivalent, but it is approximately

20% more computationally efficient. The relevant modifications can be found on lines 1, 22, and 23 of algorithm 1 in Appendix A.4. These modifications, while transparent in a mathematical sense, are essential if the approximate-Schur preconditioner is to be competitive with additive Schwarz.

A linear solver that uses the approximate-Schur preconditioner must be a flexible variant. That is, the solver must be compatible with preconditioning that varies from iteration to iteration. For this reason, I use Flexible GMRES (FGMRES) [149] with the Schur preconditioner. FGMRES uses approximately twice the memory of GMRES, although it is essentially identical in terms of CPU time.

In the context of the approximate-Schur preconditioner, one advantage of using SATs to couple the blocks is that the reduced system size is independent of the order of the interior scheme; therefore, I anticipate that the approximate-Schur preconditioner will be well suited to parallel implicit high-order finite-difference schemes.

3.5 Verification and Validation

The use of SATs in computational aerodynamics is not common, so a demonstration of solution accuracy and grid convergence is warranted. I use the converging-diverging nozzle flow to verify the discretization, since this quasi-one-dimensional flow has an analytical solution for both subsonic and transonic regimes. To demonstrate the discretization in a more practical setting, I consider the flow around the ONERA M6 wing.

3.5.1 Verification Based on a Converging-Diverging Nozzle

The steady flow in a converging-diverging nozzle provides a simple example that can be used to demonstrate and verify the SAT interface treatment. The nozzle cross-sectional area is given by

$$\frac{S(x)}{S_{\min}} = \begin{cases} 1 + 1.5\left(1 - \frac{x}{5}\right)^2 & 0 \le x \le 5 \\ 1 + 0.5\left(1 - \frac{x}{5}\right)^2 & 5 < x \le 10 \end{cases}$$

where $S_{\min}$ is the area at the throat of the nozzle (in m^2), and $x/(1\mathrm{m})$ is the non-dimensional coordinate along the nozzle axis. The stagnation variables are given by $T_0 = 300K$ and $p_0 = 1\mathrm{bar}$, with air considered a perfect gas. I consider subsonic and transonic flows with

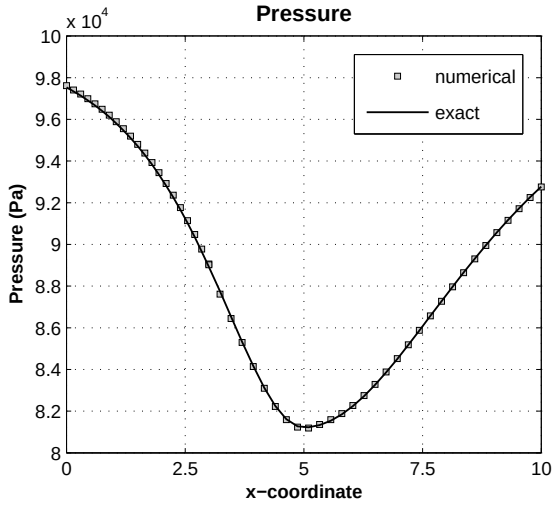


Figure 3.3: Solution for the subsonic converging-diverging nozzle flow.

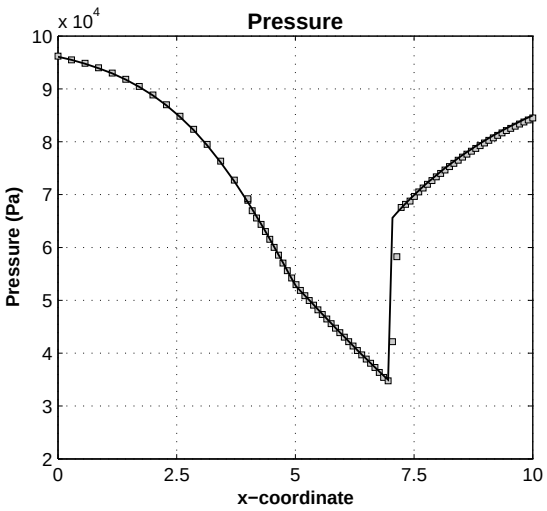


Figure 3.4: Solution for the transonic converging-diverging nozzle flow.

$S^* = 0.8$ and $S^* = 1$, respectively.

Figure 3.3 compares the numerical and exact solutions for the subsonic nozzle flow. The domain for the numerical solution consists of two subdomains with $N = 51$ and 21 nodes ($s = 20$) in the left domain (see Figure 3.1 on page 42 for an explanation of N and s). The interface is located at $x_s = x_{s+1} = 3$, and the mesh spacing ratio between the domains is $\Delta x_L/\Delta x_R = 9/14$. Similarly, results for the transonic flow are shown in Figure 3.4. For this case, the subdomain interface is located at $x_i = 4$, the refinement level is given by $(N, s) = (80, 14)$, and $\Delta x_L/\Delta x_R = 26/9$.

The two plots in Figure 3.5 compare the exact subsonic solution with the numerical solution obtained using two different refinements: 51 nodes in the upper and 251 nodes in the lower plot. Only a portion of the domain is shown, centered around the interface. The SAT treatment permits a multivalued solution at the interface, but, as this figure demonstrates, the interface discontinuity diminishes with refinement.

I conducted a mesh convergence study to confirm the order of accuracy of the SBP-SAT discretization. The RMS error in local Mach number was calculated for grids with (N, s) pairs equal to $(51, 20)$,$(251, 100)$,$(1251, 500)$, and $(6251, 2500)$. Each subdomain used a uniform mesh, and the interface location was held fixed at $x = 3$. Figure 3.6 plots the

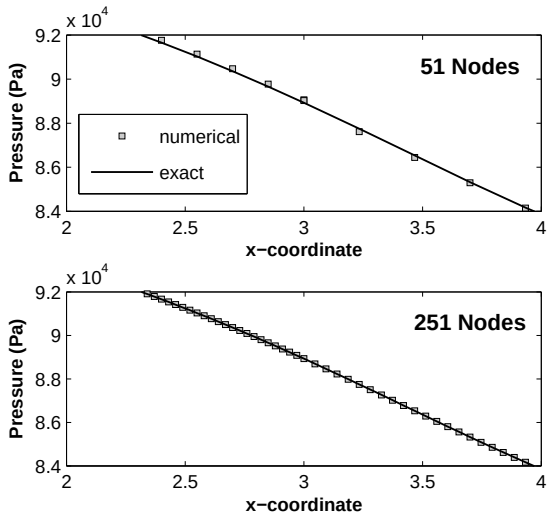


Figure 3.5: Close up of interface for two mesh densities; subsonic flow

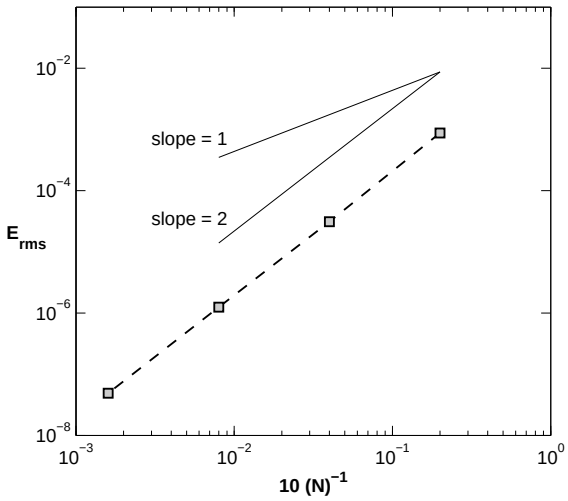


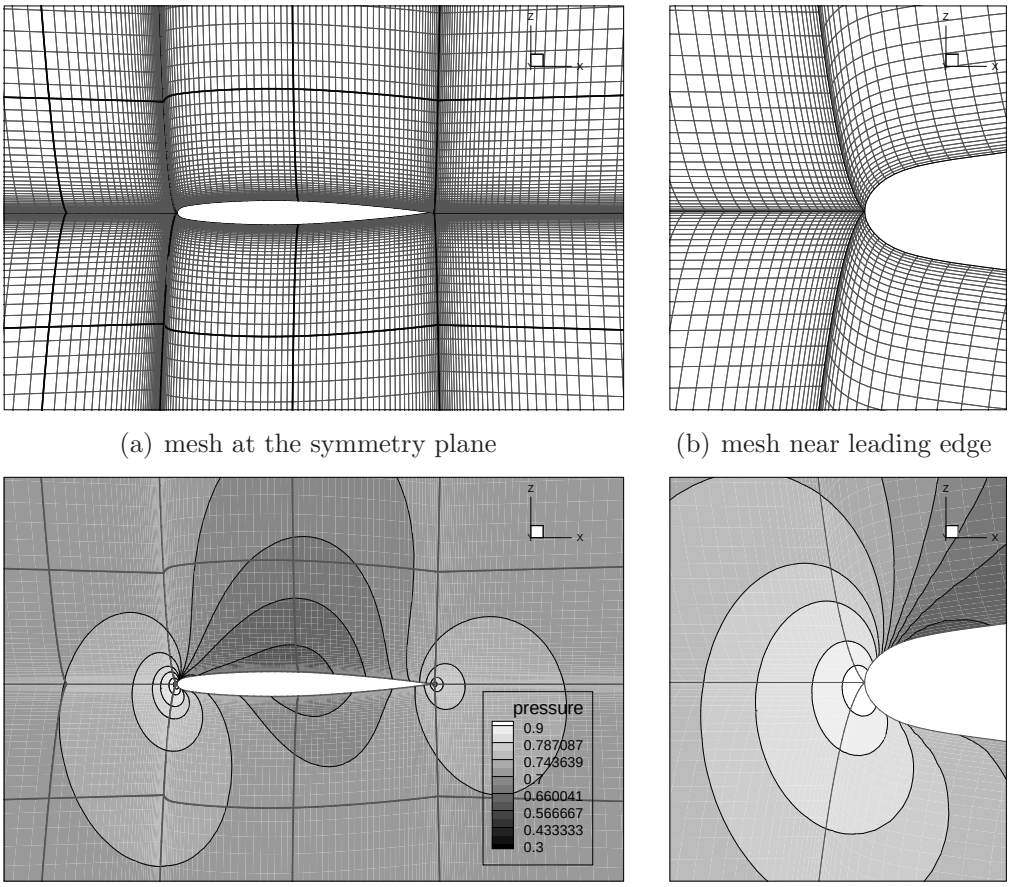
Figure 3.6: Mesh convergence plot for the subsonic nozzle.

RMS error versus a nominal mesh spacing of $10/N$. The convergence rate is clearly second-order despite a non-smooth global mesh ($\Delta x_L/\Delta x_R = \frac{9}{14}$), and the discretization being only first-order accurate at the interface; this latter point is consistent with Gustafsson's analysis [56].

3.5.2 Validation Based on the ONERA M6 Wing

Validating the code using experimental results is also useful, although only qualitative comparisons can be made using the inviscid code. To validate the solver, I obtained flow solutions for the ONERA M6 wing at Mach numbers 0.699 and 0.84 using a 96 block grid with $33 \times 33 \times 33$ nodes per block. The angle of attack was fixed at 3.06 degrees. Figures A.1 and A.2, located in Appendix A, compare C_{p} distributions obtained using the SBP-SAT discretization with the experimental results of Schmitt and Charpin [157]. Discontinuities in the solutions are visible near the mid chord and mark the location of block interfaces. As discussed earlier, such discontinuities are permitted by the SAT methodology, and do not affect the stability or the order of accuracy, as shown in Figure 3.6. Consequently, any dependence the solution has on the block topology is on the order of the discretization.

The SAT methodology requires only C^0 continuity at block interfaces; this is one of the



(a) mesh at the symmetry plane

(b) mesh near leading edge

(c) pressure contours at the symmetry plane

(d) contours near leading edge

Figure 3.7: Pressure contours and mesh at the symmetry plane of the ONERA M6 wing, Mach number of 0.699.

advantages of the scheme over the halo-node approach. Indeed, the 96 block grid used in this study was produced using trans-finite interpolation without elliptic smoothing, leading to slope discontinuities at the block interfaces. Figure 3.7 shows the pressure contours and mesh on the symmetry plane of the 96 block grid, for a Mach number of 0.699. A close-up of the leading edge shows that the slope discontinuities in the mesh do not affect the solution.

3.6 Preconditioner and Algorithm Performance

In this section, I assess the solution algorithm and the preconditioners. The assessment is based on results obtained on an SMP HP Itanium Beowulf-class cluster. Each node on the cluster consists of 4 Itanium 2 processors with 6 MBytes L3 cache and a clock speed

Table 3.1: Details of the ONERA M6 grids used to evaluate the preconditioners.

blocks	**dimensions** $(N_1 \times N_2 \times N_3)$	**grid size** (nodes)
96	$23 \times 23 \times 23$	1 168 032
48	$33 \times 17 \times 17$	457 776
12	$23 \times 23 \times 23$	146 004

of 1500 MHz. The nodes each have at least 8 GBytes of RAM, and are connected with a high-bandwidth low-latency Myrinet network.

The results in the following subsections were obtained using the ONERA M6 wing geometry and the three grids listed in Table 3.1. The grids use an HH topology and 25 chord lengths to the far field. For a given grid, each block has identical N_1, N_2, and N_3 dimensions. Sizing the blocks this way allows for better load balancing; however, future work should consider block decomposition as a means of load balancing arbitrary structured grids.

3.6.1 Memory Contention Issues

While validating the parallel solver, I observed that some communication-free operations were not 100% efficient. Memory contention was identified as the problem; when the same cases were run with 1 process per node, the efficiency improved to the expected levels.

One way to reduce memory contention is by improving cache residency. If the number of floating point operations per memory access can be increased, i.e. fewer cache misses, then the likelihood of contention should decrease. The subroutines that experience the most cache misses are the explicit matrix-vector products and the $U_i^{-1}L_i^{-1}$ forward-backward solves. These subroutines loop through compressed matrix rows and access non-sequential elements of a vector. This non-sequential access of vector elements is responsible for most of the memory contention.

The equations and unknowns can easily be ordered to give the Jacobian matrix a 5×5 block structure. If the matrix is stored as rows of blocks rather than rows of numbers, then matrix-vector operations should experience fewer cache misses. This matrix storage scheme is referred to as a block compressed row (BCR) storage format; early versions of the code used the traditional compressed sparse row (CSR) storage scheme.

For the CSR format, the ILU(p) factorization is slightly modified to allow fill-in for any block that contains non-zeros, i.e., Block-Fill ILU(p) (BFILU(p), [132]). As noted earlier, I use BILU(p) for the BCR format, which is analogous to ILU(p) except that matrix operations replace scalar operations. For the Euler equations with scalar dissipation, BFILU(p) and BILU(p) lead to almost identical ILU factorizations for CSR and BCR formats, respectively. The only differences arise at the interfaces due to the SAT linearization. It follows that differences in CPU time between the two formats are attributable primarily to changes in cache residency.

Flow solutions for the 48 block grid (see Table 3.1) were obtained for 4, 8, and 12 processors at a Mach number of 0.699 and an angle of attack of 3.06 degrees. Both the CSR and BCR matrix formats were used together with the approximate-Schur and additive-Schwarz preconditioners. Run times using 4 processors per node, T(4 proc/node), were scaled by run times using 1 processor per node, T(1 proc/node); these ratios are plotted in Figure 3.8. For all cases, memory contention decreases as the number of processors increases, since the local linear systems become smaller. The results demonstrate that the BCR format outperforms the CSR format on this architecture, independent of preconditioner. The figure also shows that the additive-Schwarz preconditioner suffers more memory contention than the approximate Schur. More GMRES iterations are required when using the Schwarz preconditioner, which means more calls to the subroutines that suffer the greatest memory contention.

While the BCR-storage scheme reduces memory contention on the given architecture, it does not eliminate the problem; therefore, the remaining results in this chapter were obtained using one processor per node. Consequently, the conclusions will be applicable to more efficient architectures, and will not be biased against the Schwarz preconditioner.

3.6.2 Preconditioner Comparison

To test the parallel efficiency of the preconditioners, flow solutions were timed for each of the three grids listed in Table 3.1. The distinct grids allow an evaluation of the preconditioners for varying numbers of blocks and mesh sizes on the same ONERA M6 geometry.

The operating conditions were fixed at a Mach number of 0.699 and an angle of attack of 3.06 degrees. The discrete equations were solved to a relative tolerance of 10^{-10} using the Newton-Krylov algorithm. For the linear sub-problems, FGMRES was limited to 60 Krylov

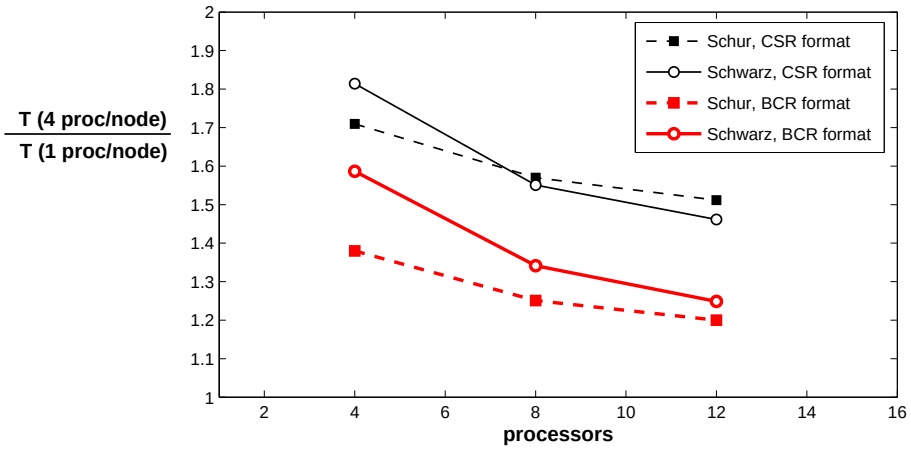


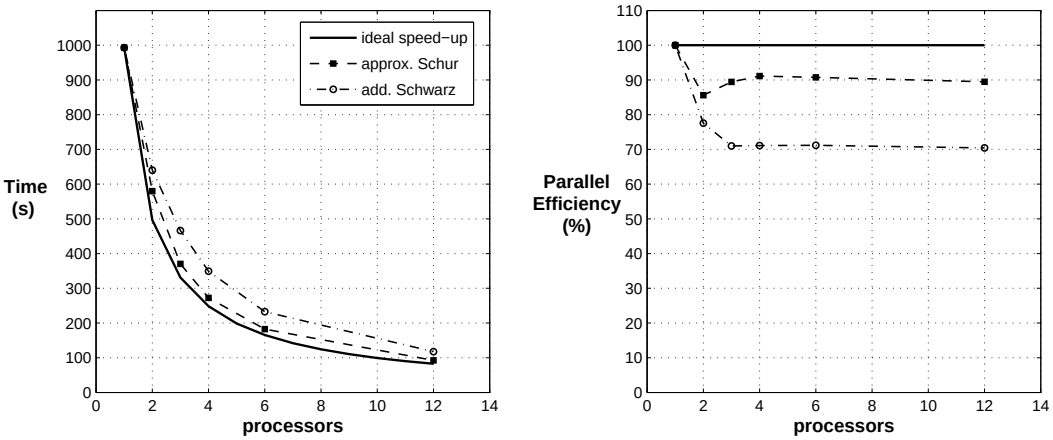
Figure 3.8: Ratio of solver run times using 4 processors per node to solver run times using 1 processor per node, as a function of number of processors.

subspace vectors per Newton iteration for the 12 block grid, and 80 subspace vectors per Newton iteration for the 48 and 96 block grids. During the approximate-Newton phase, a Jacobian update period of $m = 5$ was used for the 12 block grid, and a period of $m = 4$ was used for the two larger grids.

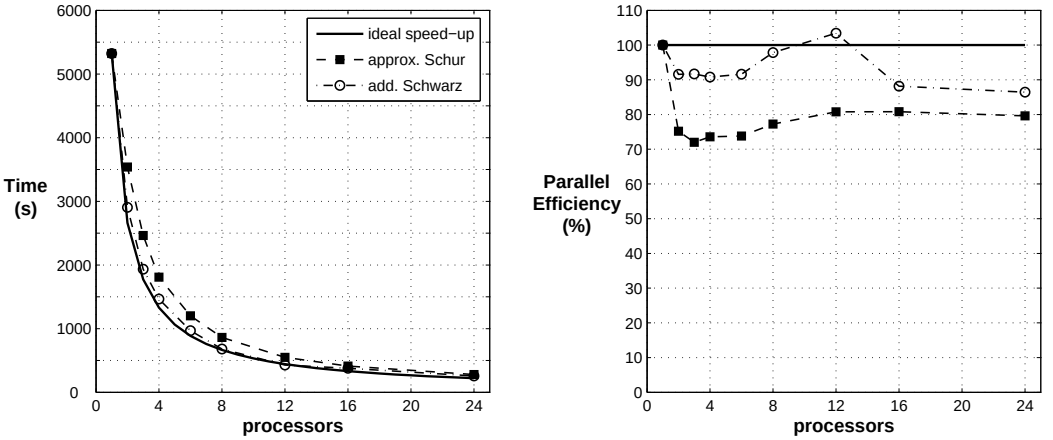
Figures 3.9(a)–3.9(c) compare the CPU time and parallel efficiency of the two preconditioners. Note that both preconditioners reduce to BILU(1) for serial computations. The results demonstrate that both preconditioners scale well on all three grids, and neither one is clearly superior in terms of CPU time.

Some remarks are necessary regarding the "super-linear" speed-up for the 12 processor additive-Schwarz time observed in Figure 3.9(b). Both parallel preconditioners change as the number of processors increases; therefore, the solution algorithm with N processors is not strictly the same as the serial algorithm. The super-linear speed-up in Figure 3.9(b) would likely not appear if I had implemented an identical block Jacobi preconditioner for the serial computation. This leads to another question: why is global BILU(1) worse than additive Schwarz in this case? One possible explanation could be additional cache misses experienced in the serial computation. Another explanation is that ILU is sensitive to the ordering of unknowns [24], so the 12 processor case may have a better ordering.

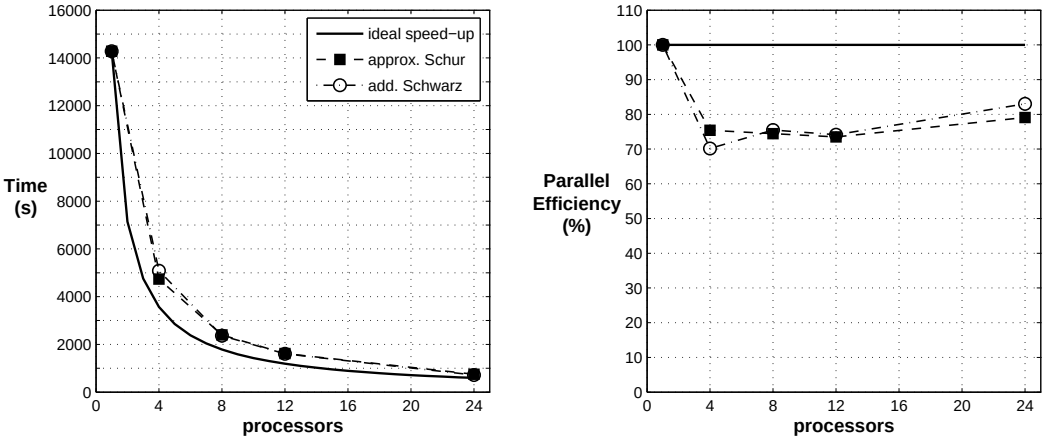
The preconditioners can also be assessed by comparing the total number of FGMRES



(a) CPU time and parallel efficiency on the 12 block, 1.46×10^5 node grid.



(b) CPU time and parallel efficiency on the 48 block, 4.58×10^5 node grid.



(c) CPU time and parallel efficiency on the 96 block, 1.168×10^6 node grid.

Figure 3.9: CPU time (s) and parallel efficiency of the approximate-Schur and additive-Schwarz preconditioners

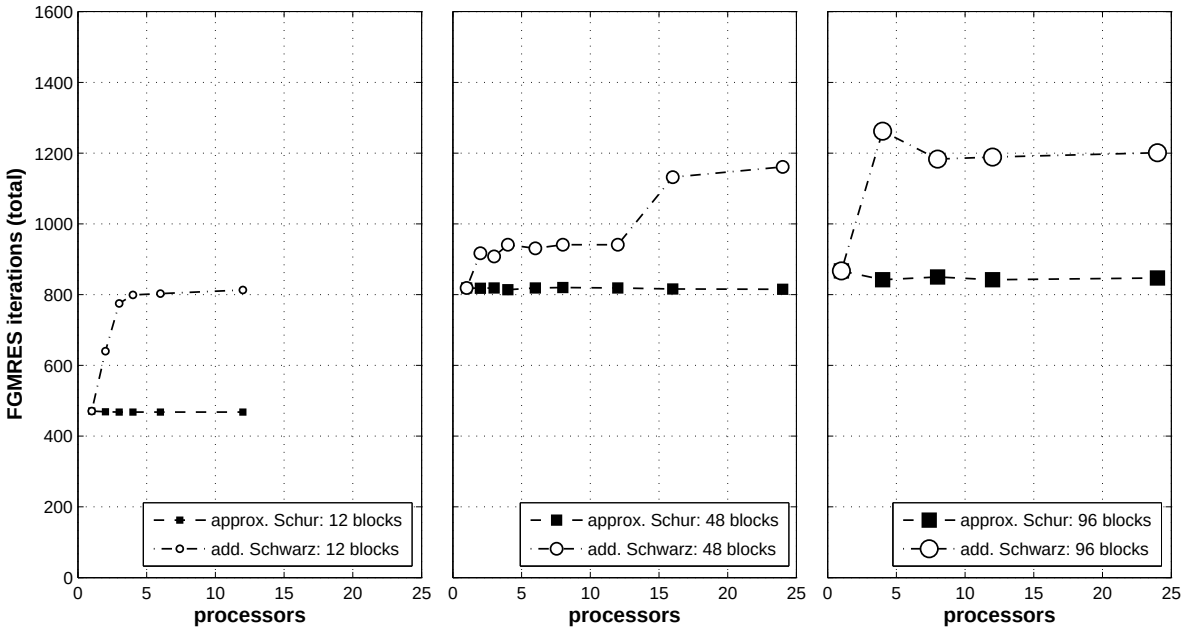


Figure 3.10: Total number of FGMRES iterations as a function of the number of processors for the 12, 48, and 96 block grids in Table 3.1.

iterations used during a flow solution. Figure 3.10 shows that the iterations required by the approximate-Schur preconditioner are independent of the number of processors, for the range of processors considered here. In contrast, the additive-Schwarz preconditioner requires more iterations as more processors are added. This difference reflects the implicit coupling between domains achieved by the approximate-Schur preconditioner.

While the approximate-Schur preconditioner requires fewer Krylov iterations, the results suggest that, on average, both preconditioners are comparable in terms of CPU time. The Schwarz preconditioner is preferable if memory considerations are important, since the Schur preconditioner requires FGMRES, while the Schwarz preconditioner can use GMRES (see the discussion at the end of Section 3.4.3). The Schur preconditioner is a better choice if many Krylov iterations are needed, or the matrix-vector products are expensive. In particular, the Schur preconditioner may prove more efficient for viscous and turbulent flows.

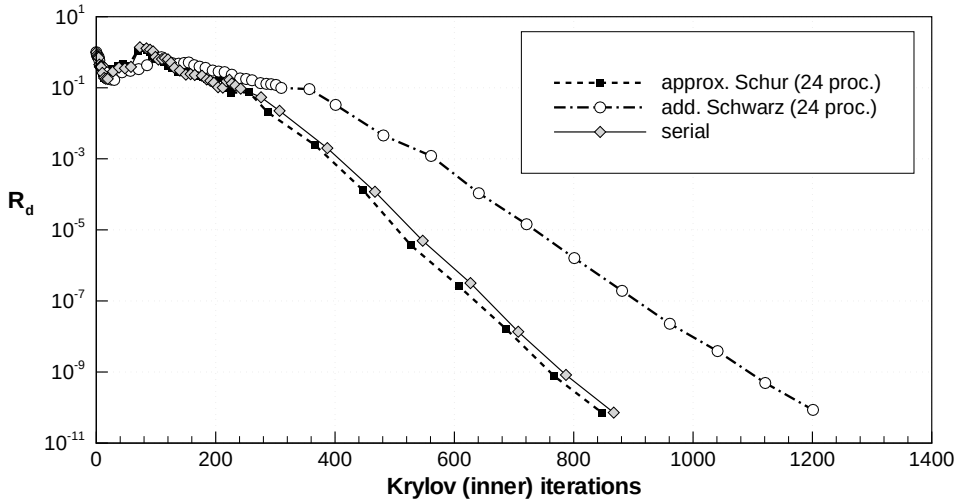


Figure 3.11: Relative residual, R_d, versus the number of Krylov iterations; the symbols denote the outer iterations. Results for the 96 block, 1.168×10^6 node grid, Mach number of 0.699, and angle of attack 3.06 degrees.

3.6.3 Algorithm Performance

I will evaluate the performance of the algorithm from two perspectives. First, I consider the number of Krylov iterations needed to reduce the relative residual, R_d, by ten orders. This measure will be useful for other developers of Newton-Krylov solvers. Second, I plot the residual convergence versus a normalized form of CPU time, which offers an architecture-independent comparison with other flow solvers.

The relative residual is plotted versus Krylov iterations in Figure 3.11. A Krylov, or inner, iteration is defined as the application of a matrix-vector product with preconditioning. The symbols in the figure denote the outer (Newton) iterations. Residual histories are shown for serial and parallel computations; for the parallel runs, both preconditioners are shown and use 24 processors. The flow solution is for the 96 block 1.168×10^6 node grid and the far-field Mach number and angle of attack are 0.699 and 3.06°, respectively. The figure shows that, for the inexact-Newton phase, the maximum number of allowable matrix-vector products (80) are used during each outer iteration: the linear solver is not able to reduce the residual to the requested tolerance. As a result, we observe linear convergence in the Newton iterations.

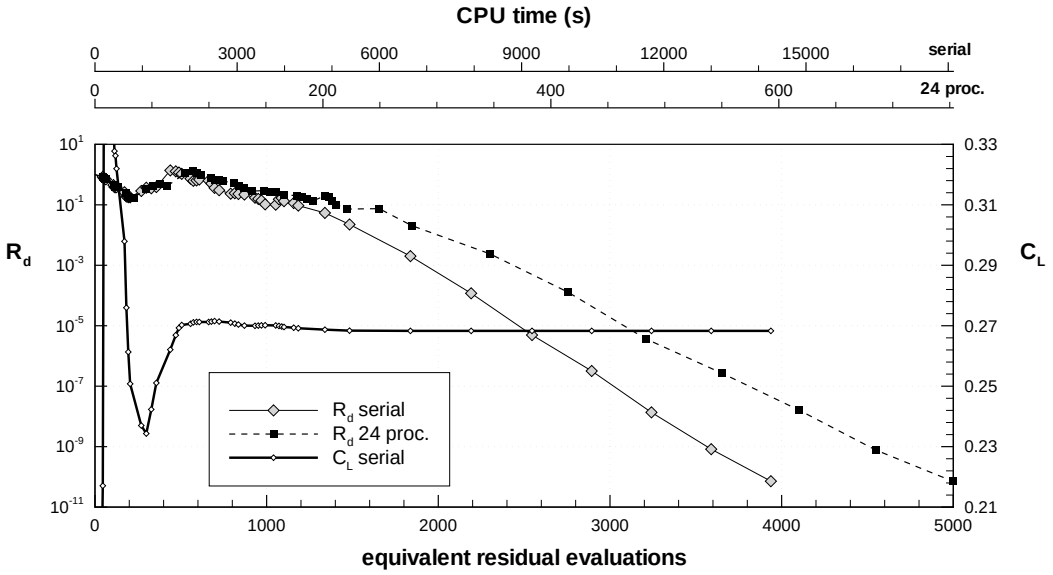


Figure 3.12: Relative residual, R_d, and C_L versus CPU time (upper axes) and equivalent residual evaluations (lower axis). The 24 processor residual history corresponds to the approximate-Schur preconditioner. Results for the 96 block, 1.168×10^6 node grid with a Mach number of 0.699 and angle of attack 3.06 degrees.

As noted earlier, it is possible to achieve super-linear convergence using a smaller tolerance and more matrix-vector products; however, super-linear convergence tends to increase CPU time, since the computational work in GMRES and FGMRES grows quadratically with matrix-vector products.

While counting inner iterations is useful for comparing with other Krylov solvers, this measure ignores preconditioner factorization time and the change in matrix-vector products, i.e. from explicit products to the matrix-free equation (3.22). For more general comparisons, I include Figure 3.12, which plots R_d versus both CPU time and equivalent residual evaluations. An equivalent residual evaluation is the CPU time needed to compute the residual, $\boldsymbol{\mathcal{R}}$, which includes all flux and SAT evaluations. I use the same grid and flow conditions as Figure 3.11. The residual history for the serial and 24 processor cases are shown. I have plotted the Schur preconditioner results, since the residual history for the Schwarz preconditioner is similar, see Figure 3.9(c). The coefficient of lift obtained at each outer iteration of the serial computation is also included in Figure 3.12. The history of C_L for 24 processors is

Table 3.2: CPU times to reach specified convergence level (digits accuracy in drag), for various grid sizes and processors.

		drag convergence level	
grid size	P **proc.**	**3 digits**	**8 digits**
1.46×10^5	12	34 s	65 s
4.58×10^5	24	108 s	216 s
11.68×10^5	24	278 s	484 s

similar. The algorithm obtains 3-digit accuracy in C_L in 1483 and 1845 equivalent residual evaluations for the serial and 24 processor runs, respectively.

Table 3.2 lists the CPU times required to achieve 3 and 8 digit accuracy in the drag. These times are provided to give a more intuitive sense of the algorithm's efficiency. Table 3.2 demonstrates that the flow solver is fast, making it practical for use in aerodynamic optimization.

Chapter 4

Gradient-Based Optimization

Like a flow solver, a numerical optimization algorithm should be efficient, producing an improved design with as few function calls as possible. The algorithm should also provide metrics that indicate the level of convergence, so we can assess whether or not the final design is a local optimum. Finally, the algorithm should explore the design space thoroughly, to assure us that the final design is a "good" local optimum, if not the global optimum.

This chapter has been adapted from my contributions in reference [69]. It begins with a review of the optimization algorithms suitable for aerodynamic shape optimization. A gradient-based algorithm has been chosen for this work; consequently, the bulk of the chapter is dedicated to the procedure used to evaluate the gradient.

4.1 Review of Optimization Methods

Optimization methods can be divided into two categories: those that use gradients and those that do not. Generally, gradient-based algorithms are more efficient at locating a local optimum, provided the gradient evaluation is independent of n, the number of design variables. The advantage of some gradient-free methods, particularly the stochastic class of algorithms, is their ability to (approximately) find the global minimum.

4.1.1 Gradient-Free Optimization Methods

Powell's Method of Conjugate Directions

Powell's gradient-free method [136] is closely related to the conjugate gradient method. The method uses a set of linearly independent search directions, $\{\mathbf{d}\}_{i=1}^{n}$, along which one-dimensional optimizations are performed each iteration. In the case of quadratic functions, these one-dimensional optimizations can be performed using interpolation. As the algorithm

progresses, the initial search directions are gradually replaced with conjugate directions[1] using the parallel subspace property [127].

The method can be applied to more general functions, but the one-dimensional optimizations can pose difficulties [127]. Moreover, computational expense is an issue, even for convex quadratic functions, for which Powell's method converges in $n-1$ iterations and performs $\mathrm{O}(n^2)$ function evaluations. To my knowledge, the method has not been applied to aerodynamic shape optimization.

Nelder-Mead Simplex

Coincidentally, Nelder and Mead [118] introduced their simplex-based method for optimization in the same journal issue that Powell introduced his gradient-free method. In $\mathbb{R}^n$, a simplex is polytope with $n+1$ vertices: it is a triangle in 2-dimensions and a tetrahedron in 3-dimensions. The Nelder-Mead method begins with a simplex in the design space, and evaluates the objective value at its vertices. The algorithm then identifies the vertex with the largest function value and attempts to improve this design point. The strategies used to improve the worst design include reflection, expansion, or contraction operations about the centroid of the remaining points. If these operations fail, the simplex is reduced in size, keeping the vertex with the best design fixed.

Despite its simplicity, the Nelder-Mead simplex method works reasonably well [127]. For example, the method was used by Sturdza [168] in the preliminary design of the Aerion natural-laminar-flow supersonic business jet.

In their original paper, Nelder and Mead used numerical experiments to estimate the average computational complexity of their method. Their results suggest that the cost of the simplex method follows an $\mathrm{O}(n^2)$ relationship. Like Powell's method, this quadratic behaviour may limit the simplex method to low-dimensional problems. In addition, the method may converge to a non-stationary point [113].

The Simulated Annealing Method

Simulated annealing (SA) is a stochastic method inspired by statistical mechanics [94]. In the method, individual designs are considered thermodynamic states at a certain "temperature"

[1] Two vectors $\mathbf{u}$ and $\mathbf{v}$ are conjugate if $\mathbf{u}^T H \mathbf{v} = 0$, where H is the Hessian matrix.

T, with the optimal designs corresponding to ground states at low temperature. For example, an initial design is perturbed randomly, and the objective function is evaluated. If the function is lower at the perturbed design, the displacement is accepted and the algorithm proceeds to the next step. If the function value increases at the perturbed design, the displacement is accepted with probability $P = \exp(-\Delta\mathcal{J}/kT)$, where $\Delta\mathcal{J}$ is the change in the objective from initial to perturbed state, T is a nominal temperature, and k is a constant. The non-zero probability allows the method to "escape" local minima. As the temperature tends to zero, so does the probability P. Not surprisingly, SA is sensitive to the rate at which the parameter T is decreased [59].

As mentioned above, SA can escape local minima. This makes the method attractive for complicated multi-modal problems. For example, Aly *et al.* [3] applied a modified SA algorithm to supersonic drag minimization of an axisymmetric forebody and found that SA was superior at locating global minima relative to a gradient-based algorithm. SA was also used by Schramm *et al.* [158] to optimize the shape of a labyrinth seal in a gas turbine; however, they considered only two variables and the design space appeared simple, if not convex. In both examples, the SA required hundreds of function evaluations for problems with fewer than 10 variables.

Genetic Algorithms

Genetic algorithms (GAs) are stochastic methods based on an analogy between optimization and the evolution of biological systems; this analogy is often attributed to Holland [74]. These algorithms work with a set of designs, called the population, rather than a single design point. The basic operations, common to most GAs, consist of selection, crossover, and mutation. The selection process is used to preferentially select the fittest members of the population for the crossover operation, i.e. those designs with the lowest function values are more likely to be parents. Crossover represents the reproductive stage of the algorithm, during which the selected parent designs are combined in some way to produce children; since the parents are good designs, the offspring are likely to be good as well. Finally, mutation is applied to a few members of the population. Mutation is essentially a random perturbation, and it allows GAs to thoroughly explore the design space. After the mutation step, the process is repeated with the new population.

Genetic algorithms are attractive for a number of reasons. Like simulated annealing, the

stochastic nature of GAs prevents them from becoming "trapped" in local optima. For example, using relatively simple problems from structural analysis, Hajela [58] showed that GAs can identify the global optimum in nonconvex and disjoint design spaces. In addition, these algorithms can operate on discontinuous functions, and integer valued variables. Finally, GAs are also well-suited to finding Pareto optimal solutions in multi-objective optimization. The principal disadvantage of GAs is their excessive cost, in terms of function evaluations. Finding suitable stopping criteria is also problematic.

There are numerous examples of GAs in the literature of aerodynamic and multidisciplinary shape optimization. The work of Quagliarella and Cioppa [141] is an early example demonstrating the potential of genetic algorithms applied to aerodynamic shape optimization. Their work also demonstrates the staggering number of function evaluations — 15878 in one example — that may be necessary. Gage *et al.* [46] present a GA for the topological design of wings and trusses; here, topological wing design refers to the number of spanwise sections. Their work is also notable for using a gradient-based search to refine the final design, making it one of the first hybrid methods in aerodynamic design. In references [177], [173] and [138] the authors apply GAs to multi-objective problems.

4.1.2 Gradient-Based Optimization Methods

Gradient-based optimization methods for nonlinear problems are mature, at least compared with stochastic methods like GAs. Indeed, most software packages are variations of only a few core ideas. Line-search and trust-region methods are two of these ideas; they are the basic algorithms used to move from the current iterate/design to an improved iterate/design. To satisfy nonlinear constraints, there are also two competing methods: sequential quadratic programming (SQP) and interior-point methods. All of the these algorithms have evolved into efficient techniques, and none has demonstrated a clear advantage over the others.

There is insufficient room and scope in this thesis for a complete review of gradient-based algorithms. Instead, I will focus on the common advantages and disadvantages that the methods provide. For a thorough review, I recommend Nocedal and Wright's book [127].

Quasi-Newton Methods and Convergence Rate

Both line-search and trust-region frameworks require the Hessian matrix; however, the Hessian is often not available or is too expensive to calculate. Instead, the most popular gradient-based methods construct an approximation to the Hessian using previous gradient values and iterates. This results in the so-called quasi-Newton class of methods.

The BFGS method is the most popular quasi-Newton method; it was co-discovered by Broyden, Fletcher, Goldfarb, and Shanno[2]. One of the key properties of BFGS is its super-linear convergence, which is guaranteed under reasonable assumptions on the problem [127]. More importantly, BFGS works well in practice, even when the initial iterate is far from the solution. BFGS constructs a positive-definite approximation and requires a convex curvature condition to hold on the updates. This may be a problem for indefinite, constrained problems.

Another quasi-Newton method, well-suited to trust-region methods, is the symmetric-rank-1 (SR1) update; see, for example, Conn *et al.* [26]. The SR1 method tends to build better approximations to the Hessian matrix than BFGS [127]. Moreover, it does not require a convex curvature condition to hold, and can capture an indefinite Hessian (useful for constrained problems). However, some updates to SR1 may need to be skipped to prevent ill-conditioned approximations [127]. With somewhat stronger assumptions than needed for BFGS, SR1 can also be shown to give super-linear convergence [127].

Regardless of the choice of quasi-Newton update, these methods usually provide rapid (super-linear) convergence, a property stochastic methods cannot attain. This is an important observation for problems with many design variables, e.g. aerodynamic shape optimization. However, another important observation is that any algorithm based on Newton's method will converge to local optima.

Methods for Constrained Optimization

As already mentioned, SQP and interior-point methods are the most popular approaches for constrained optimization, so I will focus this review on these methods.

Efficient methods have been developed to solve quadratic programs (QPs), optimization problems with linear constraints and a quadratic objective. Both SQP and interior-point algorithms are extensions of methods developed for quadratic programs.

[2]Davidon deserves some credit as well [27].

At each step of the SQP method, the nonlinear problem is approximated using a quadratic model, and the next iterate is obtained by solving the quadratic program. Variations of the method arise depending on how inequality constraints are handled. The equality-constrained QP (EQP) approach imposes some of the inequality constraints as equality constraints, i.e. those that are active, and ignores the rest [127]. Conversely, in inequality-constrained QP (IQP) the complete set of (linearized) inequalities is enforced during the solution of the QP [127]. Both SQP approaches can use line-search or trust-region methods. Popular SQP implementations include SNOPT [49], FILTERSQP [42], and KNITRO/ACTIVE [17].

Interior-point methods circumvent the difficulties associated with the inequality constraints by introducing a modified problem consisting of only equality and bound constraints. The modified problem depends on a continuation parameter μ, such that the original problem is recovered as $\mu \to 0$. A sequence of modified problems are then solved by introducing Lagrange multipliers and using a quasi-Newton approximation for the Lagrangian. Interior-point algorithms that use line-search methods must modify the search directions, when merit functions are used, or provide a feasibility restoration phase, when filter-based line-searches are used. These modifications are needed to avoid convergence problems [127], particularly when the initial point is infeasible (i.e. constraints are not satisfied). Interior-point algorithms have shown considerable promise on large-scale problems with more than a thousand variables. Example implementations include IPOPT [180] and KNITRO/DIRECT [181].

When coupled with a quasi-Newton approximation, both SQP and interior-point algorithms are efficient optimization algorithms for general nonlinear problems. Because they incorporate information about the constraints, they generally avoid infeasible regions of the design space, a feature that is more difficult to build into stochastic methods. These methods converge rapidly near the solution, and work well in practice. Again, their principal disadvantage is that they will converge to a local optimum, which is not necessarily the global optimum.

4.1.3 The Case for a Gradient-Based Algorithm

The choice of optimization algorithm is a choice between fast local convergence (gradient-based methods) and increased probability of finding the global optimum (stochastic methods). For this thesis I have chosen to use a gradient-based algorithm. There are three

reasons that motivated this choice. First, the efficiency of gradient-based methods makes a strong argument in their favour. Second, previous work with 2-dimensional flows suggests that the design space for aerodynamic optimization is convex [119]. Although 3-dimensional problems are more likely to be multimodal, a simple multistart procedure, i.e. starting from different initial designs, may be sufficient. Third, if it is established that the design space is highly multimodal, a hybrid approach coupling a stochastic global search and a gradient-based local search can be developed; in this case, the relatively difficult task of implementing a gradient-based algorithm will be complete.

As already mentioned, many excellent gradient-based optimization codes have been developed. Conceivably, any one of these packages could be chosen for this work. The true challenge is then clear: an efficient evaluation of the objective function gradient[3]. The following sections describe how the gradient is evaluated in this work.

4.2 Gradient Evaluation via Adjoint Variables

4.2.1 Notation

Some notation will simplify the subsequent analysis and discussion in this chapter. Let $\mathcal{J}$ denote an objective function that we wish to minimize; for example, drag or $C_{\mathrm{D}}/C_{\mathrm{L}}$. Let the vector of design variables be $\mathbf{v}$, and partition the design variables as

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_{\mathrm{geo}} \\ \alpha \end{bmatrix},$$

where $\mathbf{v}_{\mathrm{geo}}$ are the geometric design variables and α is the angle of attack.

Recall that the geometric design variables are the coordinates of the B-spline control points on the aerodynamic surface. To facilitate a formal definition of $\mathbf{v}_{\mathrm{geo}}$, the B-spline coordinates are ordered such that all internal control points are first, then non-surface boundary points, and finally the surface control points:

$$\mathbf{b} = \begin{bmatrix} \mathbf{b}_{\mathrm{int}} \\ \mathbf{b}_{\mathrm{bnd}} \\ \mathbf{b}_{\mathrm{srf}} \end{bmatrix},$$

[3]Some constraints, e.g. lift, may also need efficient gradient evaluations.

where $\mathbf{b}_{\text{int}} \in \mathbb{R}^{3N_{\text{int}}}$, $\mathbf{b}_{\text{bnd}} \in \mathbb{R}^{3N_{\text{bnd}}}$, and $\mathbf{b}_{\text{srf}} \in \mathbb{R}^{3N_{\text{srf}}}$. It follows that we can relate the geometric design variables and the B-spline coordinates using the restriction

$$\mathbf{v}_{\text{geo}} = \begin{bmatrix} 0 & 0 & I_{3N_s} \end{bmatrix} \mathbf{b}^{(m)} = \mathbf{b}_{\text{srf}}^{(m)} \tag{4.1}$$

where $I_{3N_{\text{srf}}}$ is the $3N_{\text{srf}} \times 3N_{\text{srf}}$ identity matrix. Recall that $\mathbf{b}^{(m)}$ is the vector of B-spline control-point coordinates for the final mesh-movement increment.

4.2.2 Using Intermediate Variables to Evaluate the Gradient

From the perspective of gradient evaluation, the ideal objective function is one that depends explicitly on the design variables, i.e. $\mathcal{J} = \mathcal{J}(\mathbf{v})$. Unfortunately, the objective functions of interest in typical engineering analysis codes do not depend explicitly on the design variables. Instead, the objective function depends on the design variables through a long sequence of operations and intermediate variables. For example, the equation for lift depends on geometric quantities, like the surface normal, and flow quantities, like the surface pressure. While the former is explicitly related to the design variables, the surface pressure is implicitly related to the variables through the solution of a discretized PDE.

Let $\mathbf{w}$ be a vector of intermediate variables, and let $\mathbf{c}(\mathbf{v}, \mathbf{w}) = \mathbf{0}$ denote the constraint equations that define the intermediate variables. For example, $\mathbf{w}$ could be the flow variables $\mathbf{q}$, and the constraint equations could be the Euler equations, $\boldsymbol{\mathcal{R}} = 0$. Alternatively, $\mathbf{w}$ and $\mathbf{c}$ could be associated with the individual variables and operations in the computer program. Presently, the vector $\mathbf{w}$ and equations $\mathbf{c} = \mathbf{0}$ are left unspecified to keep the analysis as general as possible.

We require a systematic procedure for evaluating the gradient when the objective depends on intermediate variables. The chain rule is one possible approach, but it requires considerable care and can quickly become tedious. Instead, suppose we include the intermediate variables in the optimization problem as follows:

$$\begin{aligned} \min \quad & \mathcal{J}(\mathbf{v}, \mathbf{w}) \\ \text{w.r.t.} \quad & \mathbf{v}, \mathbf{w} \\ \text{s.t.} \quad & \mathbf{c}(\mathbf{v}, \mathbf{w}) = \mathbf{0}. \end{aligned} \tag{4.2}$$

The intermediate variables $\mathbf{w}$ and equations $\mathbf{c}$ are ordered sequentially, such that c_i can be calculated using the design variables $\mathbf{v}$ and the intermediate variables up to w_i, i.e. the

equations are ordered as they would appear in the algorithm. Moreover, as we shall see, it may be advantageous to partition $\mathbf{w}$ and $\mathbf{c}$ according to particular tasks:

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_{N_w} \end{pmatrix}, \qquad \mathbf{c} = \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_{N_w} \end{pmatrix},$$

where N_w is the number of partitions. Note that, based on the sequential ordering, $\mathbf{c}_i = \mathbf{c}_i(\mathbf{v}, \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_i)$.

Next, we introduce the Lagrangian function, $\mathcal{L}$, for the constrained optimization problem (4.2):

$$\mathcal{L} \equiv \mathcal{J} + \boldsymbol{\Lambda}^T \mathbf{c}, \tag{4.3}$$

where $\boldsymbol{\Lambda}^T = [\boldsymbol{\lambda}_1^T \, \boldsymbol{\lambda}_2^T \, \cdots \, \boldsymbol{\lambda}_{N_w}^T]$ are the Lagrange multipliers, also called adjoint variables. The first-order (necessary) optimality conditions for the problem (4.2) are obtained by setting the partial derivatives of $\mathcal{L}$ to zero [127]:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Lambda}} = 0 = \mathbf{c}, \tag{4.4}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 = \frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \boldsymbol{\Lambda}^T \frac{\partial \mathbf{c}}{\partial \mathbf{w}}, \tag{4.5}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = 0 = \frac{\partial \mathcal{J}}{\partial \mathbf{v}} + \boldsymbol{\Lambda}^T \frac{\partial \mathbf{c}}{\partial \mathbf{v}}. \tag{4.6}$$

These optimality conditions are also called the Karush-Kuhn-Tucker (KKT) conditions [127]. The first condition, (4.4), is satisfied provided that the intermediate variables have been evaluated by the analysis algorithm. For example, if we are calculating the lift in an inviscid flow, this would mean that all operations necessary to solve the flow and evaluate the pressure have taken place. The second condition, (4.5), can be rearranged to solve for the Lagrange multipliers; it is restated below in a transposed and expanded form:

$$\begin{bmatrix} \left(\frac{\partial \mathbf{c}_1}{\partial \mathbf{w}_1}\right)^T & \left(\frac{\partial \mathbf{c}_2}{\partial \mathbf{w}_1}\right)^T & \cdots & \left(\frac{\partial \mathbf{c}_{N_w}}{\partial \mathbf{w}_1}\right)^T \\ 0 & \left(\frac{\partial \mathbf{c}_2}{\partial \mathbf{w}_2}\right)^T & \cdots & \left(\frac{\partial \mathbf{c}_{N_w}}{\partial \mathbf{w}_2}\right)^T \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\partial \mathbf{c}_{N_w}}{\partial \mathbf{w}_{N_w}}\right)^T \end{bmatrix} \begin{pmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ \vdots \\ \boldsymbol{\lambda}_{N_w} \end{pmatrix} = - \begin{pmatrix} \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}_1}\right)^T \\ \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}_2}\right)^T \\ \vdots \\ \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}_{N_w}}\right)^T \end{pmatrix} \tag{4.7}$$

The block-upper-triangular structure of the constraint Jacobian matrix is a consequence of the sequential ordering, since $\mathbf{c}_i$ is independent of $\{\mathbf{w}_j\}_{j=i+1}^{N_w}$. This structure reveals that the adjoint vectors λ_i must be solved in reverse order relative to the ordering used to calculate $\mathbf{w}$. The final condition of first-order optimality, (4.6), is non-zero in general and must be driven to zero using a numerical optimization algorithm. It follows from $\mathbf{c} = \mathbf{0}$ that $\mathcal{L} = \mathcal{J}$, so $\partial\mathcal{L}/\partial\mathbf{v}$ is the total derivative of $\mathcal{J}$ with respect to the design variables, i.e. it is the desired gradient.

The preceding analysis describes a sequential procedure that can be used to evaluate the gradient when the objective depends on intermediate variables. The procedure is highly general, since we are free to choose and partition the intermediate variables. At one extreme, we may associate each variable in the code with an intermediate variable and each operation with a constraint. This perspective defines the reverse mode of algorithmic differentiation. For aerodynamic optimization, a more efficient approach often results if we select and partition the intermediate variables based on high-level variables.

I conclude this section by mentioning an alternative to the sequential approach: the so-called one-shot methods[4] for solving the optimization problem (4.2). In these methods, the constraint equations for the intermediate variables are incorporated into the optimization process and are not enforced exactly at every iteration. In other words, the design and intermediate variables evolve and converge simultaneously in a global Newton iteration. This approach has been investigated by many authors [43, 171, 78, 40, 47, 61], and shows promise; however, as more disciplines are added to the optimization problem, the more complex the simultaneous system will become.

4.2.3 Intermediate Variables in Aerodynamic Analysis

How should we define the intermediate variables when evaluating the gradient of aerodynamic objective functions? The reverse mode of algorithmic differentiation, mentioned above, requires us to store, or recalculate, every intermediate variable used to find the objective function. This is not practical for aerodynamic analysis codes that use iterative schemes to solve the discrete equations. Instead, I associate $\mathbf{w}$ with those variables that require the solution of linear or nonlinear systems of equations. This is the most common and, arguably,

[4] one-shot methods are also referred to as all-at-once methods and simultaneous analysis and design.

efficient choice for the intermediate variables. It recognizes that some intermediate variables, like those used to converge the iterative solvers, are a means to an end and are not needed to evaluate the gradient.

In the present algorithm, the mesh movement may be broken into as many as m increments, and linear systems for the B-spline control points are solved at each increment. In addition, the nonlinear discrete Euler equations must be solved to find the flow variables; therefore, I define the intermediate variables and constraint equations as

$$\mathbf{w} = \begin{pmatrix} \mathbf{b}^{(1)} \\ \mathbf{b}^{(2)} \\ \vdots \\ \mathbf{b}^{(m)} \\ \mathbf{q} \end{pmatrix}, \qquad \text{and} \qquad \mathbf{c} = \begin{pmatrix} \boldsymbol{\mathcal{M}}^{(1)}\left(\mathbf{v}, \mathbf{b}^{(1)}\right) \\ \boldsymbol{\mathcal{M}}^{(2)}\left(\mathbf{v}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}\right) \\ \vdots \\ \boldsymbol{\mathcal{M}}^{(m)}\left(\mathbf{v}, \mathbf{b}^{(m-1)}, \mathbf{b}^{(m)}\right) \\ \boldsymbol{\mathcal{R}}\left(\mathbf{v}, \mathbf{b}^{(m)}, \mathbf{q}\right) \end{pmatrix}.$$

As in the general case, the ordering of the block vectors in $\mathbf{w}$ and $\mathbf{c}$ must follow the sequence used in the algorithm, e.g. the flow variables are solved after the final control points are available to regenerate the mesh.

In defining $\mathbf{c}$, I have indicated the independent variables for each of the constraint equations. In some cases, the source of the dependency is not immediately obvious. The design variables enter the mesh-movement residuals, since they determine the position of the surface control points via (2.8) and (4.1), which together give

$$\mathbf{b}_{\text{srf}}^{(i)} = \frac{i}{m}\left(\mathbf{v}_{\text{geo}} - \mathbf{b}_{\text{srf}}^{(0)}\right) + \mathbf{b}_{\text{srf}}^{(0)}, \qquad i = 1, \ldots, m.$$

The flow residual $\boldsymbol{\mathcal{R}}$ depends on $\mathbf{b}^{(m)}$, because these control points determine the grid coordinates, recall (2.3), and the grid coordinates are needed in the evaluation of the metric terms, see (3.16). Finally, the flow residual depends on the design variables through the dependence of the far-field boundary conditions on the angle of attack.

To find the gradient of an aerodynamic objective using the procedure described in Section 4.2.2, we need to introduce Lagrange multipliers (i.e. adjoint variables) for each of the constraint equations and solve the system (4.7). Specializing to the present algorithm, this

system becomes

$$\begin{bmatrix} \left(\frac{\partial \boldsymbol{\mathcal{M}}^{(1)}}{\partial \mathbf{b}^{(1)}}\right)^T & \left(\frac{\partial \boldsymbol{\mathcal{M}}^{(2)}}{\partial \mathbf{b}^{(1)}}\right)^T & 0 & \cdots & 0 & 0 \\ 0 & \left(\frac{\partial \boldsymbol{\mathcal{M}}^{(2)}}{\partial \mathbf{b}^{(2)}}\right)^T & \left(\frac{\partial \boldsymbol{\mathcal{M}}^{(3)}}{\partial \mathbf{b}^{(2)}}\right)^T & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \left(\frac{\partial \boldsymbol{\mathcal{M}}^{(m)}}{\partial \mathbf{b}^{(m)}}\right)^T & \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{b}^{(m)}}\right)^T \\ 0 & 0 & 0 & \cdots & 0 & \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{q}}\right)^T \end{bmatrix} \begin{pmatrix} \boldsymbol{\lambda}^{(1)} \\ \boldsymbol{\lambda}^{(2)} \\ \vdots \\ \boldsymbol{\lambda}^{(m)} \\ \boldsymbol{\psi} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{b}^{(m)}}\right)^T \\ -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{q}}\right)^T \end{pmatrix}. \tag{4.8}$$

The Lagrange multipliers $\{\boldsymbol{\lambda}^{(i)}\}_{i=1}^m$ are associated with the mesh movement equations, while the vector $\boldsymbol{\psi}$ is associated with the discrete flow equations. I have adopted this notational distinction between mesh- and flow-adjoint variables to remain consistent with the literature [81, 143].

Rather than an upper triangular matrix, the present algorithm produces an upper-bidiagonal matrix. As in the general case, we can rearrange the global system into a sequence of smaller systems that are solved in reverse with respect to the intermediate variables. Specifically, the sequence of systems is

$$\left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{q}}\right)^T \boldsymbol{\psi} = -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{q}}\right)^T, \tag{4.9}$$

$$\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(m)}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\lambda}^{(m)} = -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{b}^{(m)}}\right)^T - \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\psi}, \tag{4.10}$$

$$\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i)}}{\partial \mathbf{b}^{(i)}}\right)^T \boldsymbol{\lambda}^{(i)} = -\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i+1)}}{\partial \mathbf{b}^{(i)}}\right)^T \boldsymbol{\lambda}^{(i+1)}, \qquad i \in \{m-1, m-2, \ldots, 1\}. \tag{4.11}$$

The vectors on the right-hand-sides of these systems are known quantities, provided the equations are solved in the order given. Section 4.3 outlines the procedure used to solve (4.9) for the flow adjoint variables. Similarly, section 4.4 describes the process of constructing and solving the mesh adjoint systems, (4.10) and (4.11).

Finally, the gradient of the objective function can be evaluated using the adjoint variables and (4.6). Specializing to the case of the present algorithm we have

$$\boldsymbol{\mathcal{G}} = \frac{\partial \mathcal{L}}{\partial \mathbf{v}} = \frac{\partial \mathcal{J}}{\partial \mathbf{v}} + \sum_{i=1}^{m} \left(\boldsymbol{\lambda}^{(i)T} \frac{\partial \boldsymbol{\mathcal{M}}^{(i)}}{\partial \mathbf{v}}\right) + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{v}}, \tag{4.12}$$

where $\boldsymbol{\mathcal{G}}$ denotes the gradient of the objective, i.e. the total derivative.

4.3 Flow Adjoint Equation

The flow adjoint variables are governed by the linear equation

$$A^T \psi = -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{q}}\right)^T. \tag{4.9}$$

Recall that $A = \partial \boldsymbol{\mathcal{R}}/\partial \mathbf{q}$ is the Jacobian matrix of the flow residual. Many strategies have been proposed for the solution of (4.9), but most use algorithms based on the flow solver and take advantage of the identical eigenvalues shared by A and A^T. Indeed, this is the approach taken in the present work, since I solve (4.9) using a Krylov iterative solver.

4.3.1 Evaluating the Exact Flow Jacobian Matrix

One of the challenges presented by the adjoint equation is evaluating the transpose of the Jacobian matrix, since the exact Jacobian is required. Nielsen and Kleb [125] used the complex-step method [166, 108] with colouring to efficiently and accurately evaluate the entries of the Jacobian matrix. Mader *et al.* [105] constructed the residual on a node-by-node basis, and then evaluated each row of the transposed Jacobian by applying the reverse-mode of automatic differentiation.

I use a combination of analytical and complex-step differentiation to evaluate the Jacobian matrix. The Euler fluxes and numerical dissipation are differentiated analytically; much of this linearization is required to form the approximate Jacobian and is simply reused. Following Nemec [121], the pressure switch in the numerical dissipation is not differentiated. This approximation introduces small errors [119], but helps to reduce the stencil size; moreover, the errors are only present when second-difference dissipation is active for shock-capturing.

Recall from Chapter 3 that the SAT operators that couple blocks and impose boundary conditions have the following form:

$$\boldsymbol{\Delta}\hat{\mathbf{f}}_i^{\pm} = \frac{1}{2}\,\mathrm{diag}\left(|\hat{A}_i| \pm \hat{A}_i\right)(\hat{\mathbf{q}} - \hat{\mathbf{q}}_{\mathrm{bc}}) \tag{3.15}$$

where $\hat{\mathbf{q}}_{\mathrm{bc}}$ contains either boundary data or neighbouring block variables. Linearizing the SATs analytically, although possible, is complicated by the flux Jacobians; therefore, the SAT operators are differentiated using the complex-step method [166]. SAT terms appear only on block sides, so the application of the complex-step method does not significantly

affect the CPU time. Note, there is no benefit to using the reverse-mode here, since the number of inputs and outputs is the same.

4.3.2 Storing The Flow Jacobian

Once an entry in the Jacobian has been calculated, its storage location in the BCR format must be determined. This is more challenging than it first appears. Consider a single row in the matrix. The columns in this row are not formed sequentially and stacked into the BCR format. Instead, since Euler fluxes are evaluated separately from dissipation, for example, we may need to add or subtract values to the same column more than once. Thus, we need a mapping from row-column pairs to locations in the BCR format. One option would be to assign and store a location at the beginning. This would require an integer for each node in the stencil, or approximately $13 \times N$ integers for the discretization considered here, where N is the number of nodes. For a third-order viscous discretization, we would need $125 \times N$ integers.

To simplify the Jacobian storage process, I take advantage of the structured discretization. Each node is assigned an integer that encodes the node's stencil by turning bits on or off. For example, at node (j, k, m) the neighbouring node $(j + 1, k, m)$ is assigned a bit: if this neighbour is present the bit is turned on; if the neighbour is absent the bit is turned off. Hence, the column storage location in the BCR format can be determined by counting the number of activated bits up to the neighbouring node of interest.

4.3.3 Flow Jacobian Verification

To verify the accuracy of the Jacobian matrix, I implemented a complex-variable version of the flow residual. When applied to the flow residual, the complex-step method provides a second-order accurate approximation of the matrix-vector product $\mathsf{A}\mathbf{z}$. Unlike finite-difference approximations, the complex-step method does not experience subtractive cancellation errors as the step size is reduced. Thus, the truncation error in the complex-step approximation can be reduced to machine accuracy by choosing a sufficiently small step size.

Figure 4.1 shows the difference between matrix-vector products evaluated using the Jacobian matrix and products evaluated using the complex-step method, for various step sizes. The same random vector, $\mathbf{z}$, is used for both products, and the second-difference dissipation

coefficient is set to zero. Similar results are produced with distinct $\mathbf{z}$, so I conclude that the Jacobian matrix is accurate to machine error.

4.3.4 Iterative Solution of the Flow Adjoint Equation

Once the Jacobian is available, many linear solvers can be applied to the solution of the flow adjoints. Appendix B.1 demonstrates that, to ensure accurate gradients, the flow adjoint system must be solved to a relative tolerance of 10^{-8}. This tolerance requires a considerable number of Krylov iterations, unlike the larger tolerance used for the linear systems of the inexact-Newton flow solver. The memory requirements of GMRES, and its flexible variant FGMRES, grow linearly with the number of iterations. This can cause problems when GMRES is applied to the adjoint problem and memory is limited. One way to reduce the memory burden is to use restarted versions of GMRES or FGMRES, denoted GMRES(m) and FGMRES(m). These solvers simply restart after every m Krylov iterations, which keeps memory requirements proportional to m. Unfortunately, restarted Krylov solvers often exhibit degraded, and in some cases stalled, convergence.

To address this, I have developed a flexible variant of the Krylov method GCROT [30]. Unlike FGMRES(m), GCROT does not discard the entire Krylov subspace each time it restarts. Instead, it maintains a set of vectors from one outer iteration to the next based on which subspace was most important to convergence. GCROT has been shown to perform very well with respect to full GMRES while maintaining a Krylov subspace of fixed size, i.e. like GMRES(m), memory requirements do not grow for GCROT. I provide a more detailed description of GCROT and its proposed variant, GCROT(m,δ), in Appendix B.2.

I demonstrate the performance of GCROT(m,δ), relative to restarted FGMRES, by solving the flow adjoint equations for a rectangular planform at Mach 0.3 and angle of attack of four degrees. The grid for this geometry is a 1 million node version of the one shown in Figure 4.3(a). I consider FGMRES(m) and GCROT(m,δ) with $m = 20$ and $m = 50$. The parameter δ provides a threshold below which vectors are discarded in the proposed version of GCROT; I use $\delta = 0.01$. Both Krylov solvers are preconditioned with the transposed approximate-Schur preconditioner constructed using an ILU(2) factorization of the local block matrices.

Figure 4.2 plots the L^2-norm of the relative residual versus equivalent residual evaluations

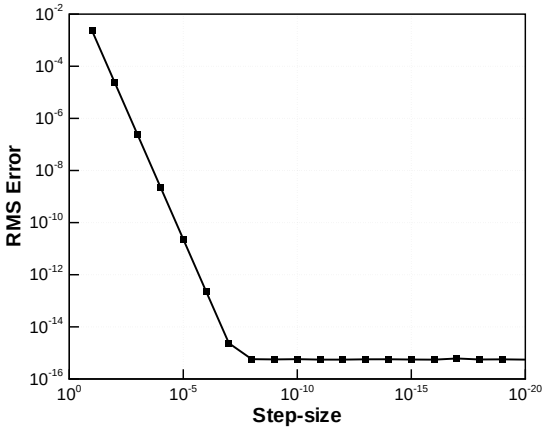


Figure 4.1: Verification of the analytical Jacobian matrix using the complex-step method

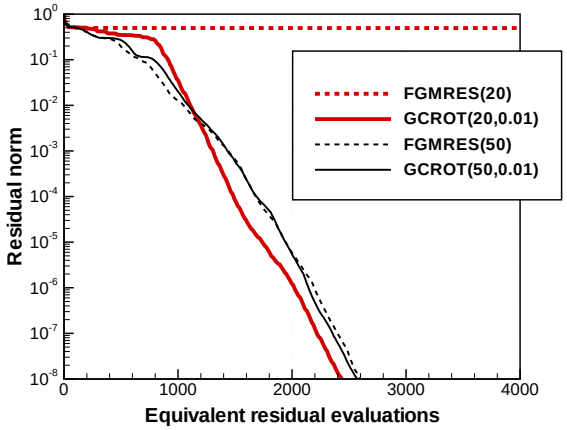


Figure 4.2: Comparison of CPU times for FGMRES and GCROT applied to the flow adjoint problem.

(recall that an equivalent residual evaluation is the CPU time needed to compute the residual $\mathcal{R}$). Typically, the flow solver requires between 5000 and 6000 equivalent residual evaluations to converge ten orders of magnitude, when the Mach number is 0.3. Figure 4.2 indicates that GCROT solves the adjoint system to a relative tolerance of 10^{-8} in 40–50% of the time needed to converge the flow solution. For the low memory case, observe that FGMRES(20) stalls while GCROT(20,0.01) is able to recover. When more memory is available, i.e. $m = 50$, the performance of the two solvers is similar. Experience suggests these results are typical: when FGMRES(m) converges, GCROT(m,δ) converges with similar CPU time; when FGMRES(m) stalls, GCROT(m,δ) is able to recover.

I conclude this section by mentioning a Krylov-based adjoint approach suggested by Griewank [53], in the general case, and by Martins [106], in the case of aerodynamic optimization. For the flow adjoint equation, Krylov iterative methods require matrix-vector products of the form $A^T\mathbf{z}$. These products can be computed using reverse-mode automatic differentiation in much the same way the forward-difference equation (3.22) is used in the flow solver. Preliminary investigations suggest that such an approach is not competitive with calculating and storing the Jacobian explicitly; however, this approach is attractive from the perspective of memory and implementation, and, as reverse-mode algorithms become more efficient, the CPU-time disadvantage should diminish.

4.4 Mesh Adjoint Equations

The analysis in Section 4.2.3 produced two types of B-spline mesh-adjoint equations, namely

$$\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(m)}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\lambda}^{(m)} = -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{b}^{(m)}}\right)^T - \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\psi} \tag{4.10}$$

and

$$\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i)}}{\partial \mathbf{b}^{(i)}}\right)^T \boldsymbol{\lambda}^{(i)} = -\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i+1)}}{\partial \mathbf{b}^{(i)}}\right)^T \boldsymbol{\lambda}^{(i+1)}, \qquad i \in \{m-1, m-2, \ldots, 1\}. \tag{4.11}$$

We shall begin with an analysis of the left-hand sides of the linear systems (4.10) and (4.11). The system matrix appearing in these equations can be found by differentiating the control-mesh movement equation (2.9) with respect to $\mathbf{b}^{(i)}$:

$$\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i)}}{\partial \mathbf{b}^{(i)}}\right)^T = K^{(i)T} = K^{(i)}, \qquad i \in \{m, m-1, \ldots, 1\},$$

where we have used the symmetry of the stiffness matrix $K^{(i)}$. The symmetry of the stiffness matrices allows the mesh-movement solution algorithm to be reused for the mesh adjoints; hence, I use the conjugate gradient method preconditioned with ILU(p) to solve both (4.10) and (4.11).

Unlike the left-hand sides, the right-hand sides of (4.10) and (4.11) are very different. To evaluate the right-hand side of the adjoint equation (4.10) I make liberal use of the chain rule:

$$-\left(\frac{\partial \mathcal{J}}{\partial \mathbf{b}^{(m)}}\right)^T - \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\psi} = -\left(\frac{\partial \mathbf{g}}{\partial \mathbf{b}^{(m)}}\right)^T \left[\left.\frac{\partial \mathcal{J}}{\partial \mathbf{g}}\right|_{\mathbf{m}} + \left(\left.\frac{\partial \mathcal{J}}{\partial \mathbf{m}}\right|_{\mathbf{g}} + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{m}}\right) \frac{\partial \mathbf{m}}{\partial \mathbf{g}}\right]^T \tag{4.13}$$

where $\mathbf{g}$ and $\mathbf{m}$ are block-column vectors of the grid coordinates and metrics, respectively. The blocks in $\mathbf{g}$ are composed of $\mathbf{x} = (x, y, z)$ at each node, with the coordinates defined by the B-spline volume mesh equation (2.3). Similarly, the blocks in $\mathbf{m}$ consist of the nine components of $\boldsymbol{\nabla}\xi_i$ at each node. The term $\partial\mathcal{J}/\partial\mathbf{g}|_{\mathbf{m}}$ denotes the partial derivative of the objective with respect to the grid coordinates while freezing the metric terms; similarly for $\partial\mathcal{J}/\partial\mathbf{m}|_{\mathbf{g}}$. Equation (4.13) provides a right-hand-side reformulation that is significantly easier to implement. Note that none of the matrices appearing in (4.13) are stored, since only the resulting vector-matrix and matrix-vector products are needed.

When the number of increments is greater than one, we must solve the additional adjoint equations (4.11). As mentioned above, these equations have identical system matrices to their corresponding movement equation. Again, the difficulty presented by these equations is evaluating their right-hand sides. The movement residual $\boldsymbol{\mathcal{M}}^{(i+1)}$ has a complicated non-linear dependence on the control points $\mathbf{b}^{(i)}$ through the Young's modulus; therefore, in the present work, I evaluate the right-hand sides of (4.11) using the complex-step method. Evaluating the right-hand sides in this way typically requires more CPU time than solving the mesh-adjoint variables. However, the relatively small control mesh implies only a small penalty in total CPU time. Clearly, the complex-step method would not be appropriate, in this context, if the equations of linear elasticity were applied to the individual grid points.

The B-spline control points $\{\mathbf{b}^{(i)}\}_{i=1}^{m}$ are used to construct the stiffness matrices and right-hand sides of the mesh adjoint equations. Recomputing these vectors is unnecessary if we store them during the forward evaluation of $\mathcal{J}$. Depending on the number of increments, storing the control points in RAM may be expensive. Alternatively, the $\mathbf{b}^{(i)}$ vectors can be saved to files on disk; this approach is used here, since I/O from disk is relatively fast for the given architecture. Moreover, read/write operations are performed only once for each vector $\mathbf{b}^{(i)}$ during a gradient evaluation.

4.5 Verification and Cost of the Gradient Calculation

4.5.1 Verification of Gradient Accuracy

Given the complexity of the present algorithm, and the use of hand differentiation, verifying the gradient accuracy is essential. My goal in this section is to demonstrate that the gradient is sufficiently accurate for gradient-based optimization algorithms.

Consider a 12 block mesh around a generic wing with no sweep; see Figure 4.3(a). Each block consists of $23\times33\times17$ nodes and is fit with B-spline volumes. The wing is parameterized using the B-spline control points corresponding to the surface. These control points are depicted as white markers in Figure 4.3(a). In total, there are 297 geometric design variables. This total excludes the y-coordinate of control points on the symmetry plane. In addition, all coordinates of one control point on the symmetry plane are frozen to prevent translation.

Each component of an objective function gradient can be verified using a finite-difference

approximation; however, this would be time consuming for the number of design variables considered here (298 including the angle of attack). Instead, I use a directional derivative to check all the gradient components simultaneously. For a given direction $\mathbf{z}$, the analytical directional derivative is given by

$$D_{\mathbf{z}}\mathcal{J} = \frac{\partial \mathcal{J}}{\partial \mathbf{v}}\mathbf{z},$$

while the second-order finite-difference approximation is

$$D_{\mathbf{z}}\mathcal{J} = \frac{\mathcal{J}(\mathbf{v} + \epsilon\mathbf{z}) - \mathcal{J}(\mathbf{v} - \epsilon\mathbf{z})}{2\epsilon} + \mathrm{O}(\epsilon^2),$$

where ϵ is the perturbation parameter.

Individual components of the gradient can differ in magnitude by 2–4 orders; therefore, it is tempting to choose a direction $\mathbf{z}$ such that each element of the gradient makes an equal contribution to $D_{\mathbf{z}}\mathcal{J}$; however, this tends to increase the step-size range over which round-off errors affect the finite-difference approximation. Instead, I use the direction

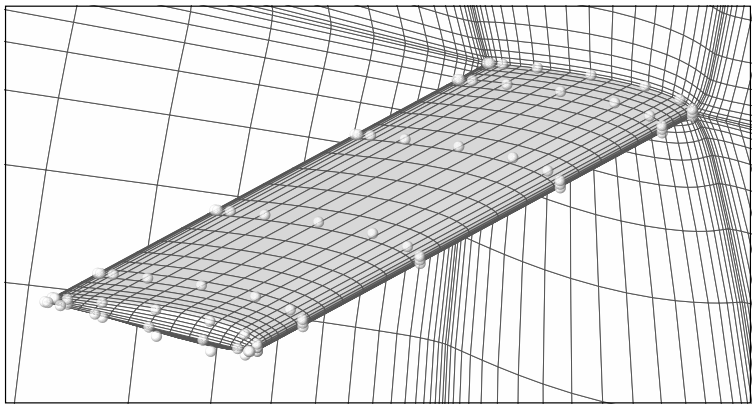
$$(\mathbf{z})_i = \operatorname{sign}\left[\left(\frac{\partial \mathcal{J}}{\partial \mathbf{v}}\right)_i\right],$$

which gives a directional derivative equal to the L^1 norm of the gradient. This direction does not eliminate the possibility that large gradient components overwhelm small gradient components, but it does prevent subtractive cancellation between gradient components.

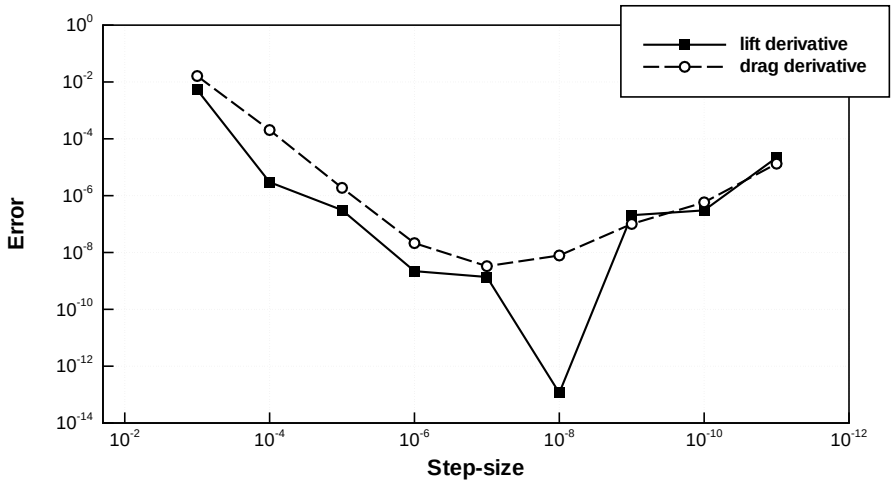
Figure 4.3(b) plots the relative error between the analytical and finite-difference values of $D_{\mathbf{z}}\mathcal{J}$, where the objective is either lift or drag. For each objective, the free-stream Mach number was fixed at 0.5 and the angle of attack at four degrees. The plot shows the expected second-order convergence of the finite-difference approximation, and its eventual contamination by round-off errors. These results suggest that the analytical gradients are at least as accurate as finite-difference approximations with optimal step sizes.

4.5.2 Efficiency of the Gradient Calculation

Table 4.1 lists a CPU-time breakdown of the objective function and gradient evaluations. The times are normalized by the total time required to calculate the objective function. Breakdowns are provided for two grid sizes to illustrate how the components of the algorithm scale.



(a) Generic wing with parameterizing control points (white spheres).



(b) Relative error between the analytical and second-order accurate finite-difference values of the directional derivative $D_{\mathbf{z}}\mathcal{J}$.

Figure 4.3: Mesh, design variables, and results for gradient verification.

The two examples in Table 4.1 use B-spline volumes with the same number of control points. Consequently, the mesh movement and mesh adjoint routines have a very weak dependence on grid size. This is reflected in the normalized CPU times, which differ by an order of magnitude between the two grids. While the mesh movement and mesh adjoint CPU times are relatively significant for the coarse grid, they represent only 3.4% of the total computational effort on the fine grid. Thus, B-spline mesh movement is very efficient for the fine grid, which is representative of the grids used in practice. For the fine grid, note that

the total time needed to compute the gradient is less than half that needed for the objective.

4.6 Optimization Algorithm

For optimization, I use the software package SNOPT [49] that solves nonlinear problems with general constraints. SNOPT uses a sequential quadratic programing algorithm, and is capable of handling both linear and nonlinear constraints. The Hessian of the Lagrangian[5] is approximated using the quasi-Newton method BFGS. I use the full memory BFGS update rather than the limited memory option, since the storage requirements are modest relative to the flow solver.

SNOPT measures convergence using criteria closely related to the KKT conditions. At convergence, the nonlinear constraints, $\mathbf{c}$, must satisfy

$$\frac{\|\mathbf{c}\|_\infty}{\|\mathbf{v}\|_2} \leq \varepsilon,$$

where $\varepsilon = 10^{-7}$ is used in this work. In addition to satisfying the constraints, optimality requires that the gradient of the Lagrangian must be sufficiently small. In particular,

$$\frac{\|\mathbf{g}\|_\infty}{\|\boldsymbol{\lambda}\|_2} \leq \varepsilon,$$

where $\boldsymbol{\lambda}$ are the SNOPT Lagrange multipliers, $\varepsilon = 10^{-7}$, and

$$(\mathbf{g})_i = g_i \equiv \begin{cases} \frac{\partial \bar{\mathcal{L}}}{\partial v_i} \min\left(v_i - v_i^{\text{low}}, 1\right), & \text{if } \frac{\partial \bar{\mathcal{L}}}{\partial v_i} \geq 0; \\ -\frac{\partial \bar{\mathcal{L}}}{\partial v_i} \min\left(v_i^{\text{up}} - v_i, 1\right), & \text{if } \frac{\partial \bar{\mathcal{L}}}{\partial v_i} < 0, \end{cases}$$

where $\bar{\mathcal{L}}$ is the SNOPT Lagrangian and $(\mathbf{v})_i = v_i$ is the i^{th} design variable. The values v_i^{low} and v_i^{up} denote the lower and upper bounds, respectively, for the variable v_i.

I report the merit function history in some cases. The merit function used in SNOPT is an augmented Lagrangian merit function. When the constraints are satisfied, the merit function is equal to the objective. For all constrained examples presented in this thesis,

[5]Note that the Lagrangian in SNOPT is not the same as the one defined in (4.3). Here, the Lagrangian is based on the objective and the constraints provided to SNOPT; the flow and mesh constraints are satisfied externally by the sequential approach.

Table 4.1: Breakdown of objective function and gradient CPU times for two grids. Relative times are normalized by the total objective function evaluation time.

		grid size (nodes)			
		$\mathbf{1.55 \times 10^5}$		$\mathbf{1.158 \times 10^6}$	
		CPU time (12 proc.)		**CPU time** (12 proc.)	
component	**solve tol.**	**relative**	**absolute**	**relative**	**absolute**
mesh movement	(10^{-12})	0.121		0.014	
flow solution	(10^{-10})	0.877		0.986	
objective function		**1.000**	**(107.0 s)**	**1.000**	**(1278.1 s)**
flow Jacobian assembly		0.005		0.003	
ILU(2) factorization		0.032		0.020	
GCROT flow adjoint solution	(10^{-8})	0.366		0.397	
mesh Jacobian assembly		0.007		0.001	
ILU(1) factorization		0.020		0.002	
PCG mesh adjoint solution	(10^{-12})	0.083		0.008	
complex-step RHS assembly		0.294		0.025	
objective function gradient		**0.812**	**(86.8 s)**	**0.457**	**(584.0 s)**

the objective function is C_D, so the merit function is equal to the coefficient of drag at convergence.

I make liberal use of SNOPT's linear-equality-constraint implementation to couple the B-spline control points. To illustrate, consider the twist optimization presented in Section 5.2.2. In this example, the design variables consist of the z-coordinates of all surface control points, 79 geometric variables in total; however, linear constraints are imposed such that each spanwise section is determined by the position of the leading edge and the (fixed) trailing edge. In this case, the number of geometric degrees of freedom is reduced to six. The linear constraints are satisfied exactly by SNOPT, so this effectively redefines the set of design variables.

If a lift constraint is imposed, then the sequential gradient evaluation must be repeated, including additional flow- and mesh-adjoint solutions for lift. As shown above, the adjoint solution process is very efficient, and adding this constraint represents an approximately 33% increase in CPU time per optimization cycle. An alternative, not explored in this work, is to impose the lift constraint as an equation in the flow solver, and add the angle of attack as a variable to the vector $\mathbf{q}$. Billing [11] has shown that this approach increases the flow solver time between 20–50%, so the total CPU time per optimization cycle is comparable to solving the lift-adjoint problem.

The optimization process may be curtailed due to time limits imposed by a queuing system. In such cases, it is desirable to warm-start the optimization using the previously calculated objectives and gradients. Although SNOPT does not provide a warm-start capability[6], I have implemented a simple strategy that is transparent to the optimization. SNOPT, like most gradient-based algorithms, is deterministic and will follow the same path if given the same information. Therefore, it is sufficient to store the objective and gradient values in the order they are produced, and read them into SNOPT in the same order during a warm-start. Once all information in the file has been exhausted, new objectives and gradients are evaluated and stored.

[6]SNOPT does have a limited version of warm-starting that restarts the optimization with the current design variables and estimates for the Lagrange multipliers; however, the BFGS Hessian approximation is discarded

Chapter 5

Studies of Induced-Drag Minimization

In Chapters 2, 3, and 4, I described the individual components of an aerodynamic optimization algorithm. The purpose of this chapter is, in part, to demonstrate that the complete optimizer produces reliable results, and that the algorithm is ready to be incorporated as a module into more realistic aircraft optimizations. Hence, the first few studies presented in this chapter, inverse design and twist optimization in particular, seek to verify and validate the optimizer.

Beyond verification and validation, I also use this chapter to study induced drag for its own sake. The study of induced drag is motivated by the work of Smith [164], who has shown that nonlinear interactions between the wake and the wing can have a significant impact on induced drag. The present algorithm, an Euler-based aerodynamic optimizer, provides a tool to investigate optimal aerodynamic shapes in flows that include nonlinear physics.

Some of the results in this chapter were presented previously in [68].

5.1 Details Regarding the Studies

5.1.1 Canonical Grids

Table 5.1 summarizes the initial grids used for the case studies in this chapter. The grids are canonical in the sense that their grid lines are rectilinear, or almost rectilinear. The geometries that the grids conform to are also rectilinear. Figure 2.7(b), on page 32, depicts a similar canonical grid.

For the optimization studies, an appropriate grid is chosen from Table 5.1 and fit using B-spline volumes. At the beginning of the optimization, an initial geometry, e.g. a rectangular

Table 5.1: Dimensions and spacing parameters for the grids used in the studies.

blocks	**grid size**	**spacing** (root-chord units)				
	(nodes)	off-wall	leading-edge	trailing-edge	wingtip	off-symmetry
12	1 158 300	0.0025	0.0025	0.0025	0.002	0.01
42	3 827 250	0.001	0.001	0.001	0.002	0.05
32	2 916 000	0.001	0.001	0.001	0.002	0.05
6	602 258	0.002	0.002	0.002	—	0.01

wing with NACA 0012 sections, is prescribed by setting the B-spline control points on the surface. The initial and subsequent geometries result in perturbed meshes. Consequently, during the optimization, the mesh spacing parameters will differ slightly from those listed in Table 5.1.

5.1.2 Default Parameter Values

The default parameters for the mesh movement, flow solver, and optimizer are listed in Table C.1 of Appendix C.3. These parameter values are adopted for the optimizations in this chapter, unless otherwise indicated.

5.1.3 Remarks on Induced Drag

When investigating induced drag, it is useful to consider the following expression from classical lifting-line theory [97, 4]:

$$D_i = \frac{L^2}{q_\infty \pi b^2 e}, \tag{5.1}$$

where D_i is the induced drag, L is the lift, $q_\infty = \frac{1}{2}\rho_\infty U_\infty^2$ is the free-stream dynamic pressure, and b is the span. The efficiency factor e accounts for non-elliptical loading; it is equal to unity for an elliptical lift distribution. When the lift, span, and Mach number are fixed, (5.1) reveals that only the efficiency factor can be used to reduce D_i. Thus, in all the optimization cases considered, the span and Mach number are fixed, and the lift is constrained.

Euler codes produce tip vortices that release off the side of the wing tip, rather than at the trailing edge. This edge separation deserves some discussion, since it influences the induced drag by creating a nonplanar wake [164]. While its physicality is debatable in an inviscid

flow, edge separation certainly exists in real flows [22], even on wings with rounded tips [25, 34]. Thus, the perspective taken here is to study the role of edge separation on induced drag, while acknowledging that the separation location and vortex size may not correspond with the true flow. Note, the geometries considered here have sharp edges along their wing tips, which should increase the likelihood that the predicted edge-separation location agrees with the physical flow.

5.1.4 Reference Area

Recall that the coefficients of lift and drag are defined as

$$C_L = \frac{L}{q_\infty S} \quad \text{and} \quad C_D = \frac{D}{q_\infty S},$$

respectively, where S is the reference area. In this thesis, the reference area is always fixed or constrained, so I can report coefficients of lift and drag, rather than lift and drag, without ambiguity. Note that the reference areas reported in this chapter are based on the half-span geometry.

For most of the induced-drag studies, the reference area is the projected area of the initial shape. The one exception is the planform optimization study, where surface area is used as the reference. The surface area is calculated using the metric terms. Suppose the surface coincides with the surface $\zeta = 0$. Then the surface area is given by

$$\begin{aligned} S &= \frac{1}{2}\iint \|\mathbf{x}_\xi \times \mathbf{x}_\eta\| d\eta d\xi \\ &= \frac{1}{2}\iint \|(y_\xi z_\eta - z_\xi y_\eta, z_\xi x_\eta - x_\xi z_\eta, x_\xi y_\eta - y_\xi x_\eta)\| d\eta d\xi \\ &= \frac{1}{2}\iint \frac{\|\boldsymbol{\nabla}\zeta\|}{J} d\eta d\xi. \end{aligned}$$

The integration includes contributions from both sides of a geometry, so the factor of 1/2 ensures that the resulting surface area is consistent with the projected area for planar geometries. Using the trapezoid rule, a second-order accurate approximation of the surface area is

$$S = \frac{1}{2}\sum_i \left(\frac{\|\boldsymbol{\nabla}\zeta\|}{J}\right)_i \Delta\eta_i \Delta\xi_i,$$

where the sum is taken over all nodes i on the surface. The values of $\Delta\xi_i$ and $\Delta\eta_i$ are equal to unity, except at the ends of the ξ and η intervals, where they are equal to one half. The gradient of the surface area is calculated analytically.

5.2 Verification and Validation

In this section, I verify and validate the optimizer using inverse design and twist optimization. Inverse design is a common verification for aerodynamic optimization. The intention with twist optimization is to recover the elliptical lift distribution predicted by linear theory. Twist optimization is a simple, yet important, validation case; oddly, there is no evidence in the literature of its use to validate high-fidelity aerodynamic optimization algorithms.

5.2.1 Inverse Design

As a simple verification, I consider an inverse design based on surface pressure. The design variables consist of the angle of attack and the 3 coordinates of a control point on the upper surface of the wing shown in Figure 4.3(a), on page 92. The initial angle of attack is four degrees. The target design is produced by randomly perturbing the four design variables. The optimizer is given the unperturbed wing and angle of attack as the initial design, and the goal is to recover the perturbed shape and angle of attack based on a target pressure distribution.

To obtain the target pressure distribution, I solve for the flow around the perturbed wing and angle of attack at a Mach number of 0.5. For the inverse design problem, the objective is defined by

$$\mathcal{J} = \frac{1}{2}\sum_{i=1}^{N_{\text{surf}}}(p_i - p_{i,\text{targ}})^2 \Delta A_i$$

where N_{surf} is the total number of surface nodes, and ΔA_i is the surface area element at node i. The pressure and target pressure at node i are denoted by p_i and $p_{i,\text{targ}}$ respectively.

Figure 5.1 shows the convergence history for the inverse design problem. The gradient converges 10 orders and the objective converges 20 orders in 25 objective function and gradient evaluations.

5.2.2 Twist Optimization

According to lifting-line theory, induced drag for a planar wake is minimized by an elliptical spanwise lift distribution; thus, classical theory provides a useful benchmark for optimization algorithms. This is also a challenging benchmark: the analysis in Appendix C.1 shows that a perturbation of order ϵ in the lift distribution produces an order ϵ^2 perturbation in the induced drag. Hence, obtaining an optimal lift distribution close to elliptical requires sufficient accuracy in the drag prediction.

As is well known, elliptical lift distributions are not unique to one geometry. The same distribution can be obtained using planform shape, twist, sectional lift, or some combination of these. For this validation, I vary the twist. In Section 5.3, I will discuss why planform variations are a poor choice for recovering the optimal distribution predicted by lifting-line theory.

The initial geometry for the twist optimization consists of a rectangular wing with NACA 0012 sections, a chord length of 2/3, and a semi-span of 2. The reference area is the projected area of the planform, $S = 4/3$. The 12 block grid from Table 5.1 is adopted for this study. The blocks are fit with B-spline volumes such that the upper and lower wing surfaces are parameterized with 9 control points in the streamwise direction and 7 control points in the spanwise direction. The parameterization is similar to the one in Figure 4.3(a). The free-stream Mach number is 0.5, and the angle of attack is fixed at 4.2416° to avoid non-unique designs. This particular angle of attack ensures that the initial geometry meets the C_L constraint of 0.375.

The trailing edge control points are fixed, and linear constraints are used to couple the twist of each spanwise station to the z-coordinate of the leading edge control point. Fixing the trailing edge helps reduce non-planar effects, although edge separation makes a completely planar wake difficult to achieve with Euler-based flow solvers. Finally, the twist of the wingtip edge is constrained to have the same twist as the neighbouring section; this constraint is necessary to prevent mini-winglets.

The convergence history for the twist optimization is shown in Figure 5.2. After 21 function evaluations the optimality measure has been reduced 4 orders of magnitude and the absolute constraint violation has been reduced below 10^{-12}. The coefficient of drag has been improved approximately 2.1% from 0.00755 to 0.00739. The optimal induced drag predicted

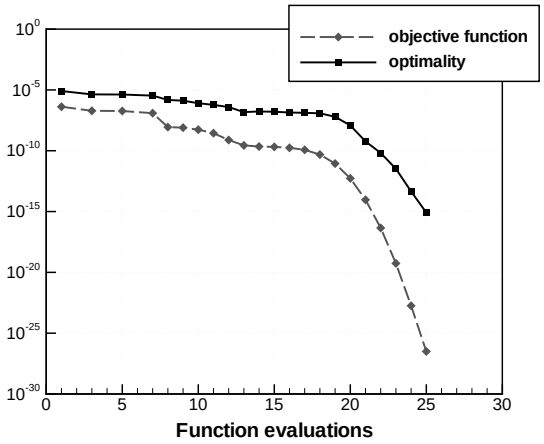


Figure 5.1: Convergence history for the inverse design verification.

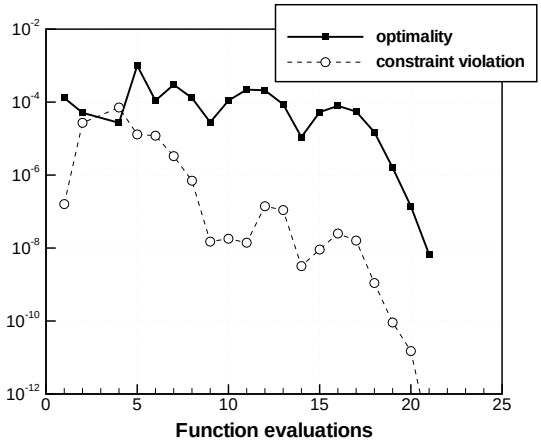


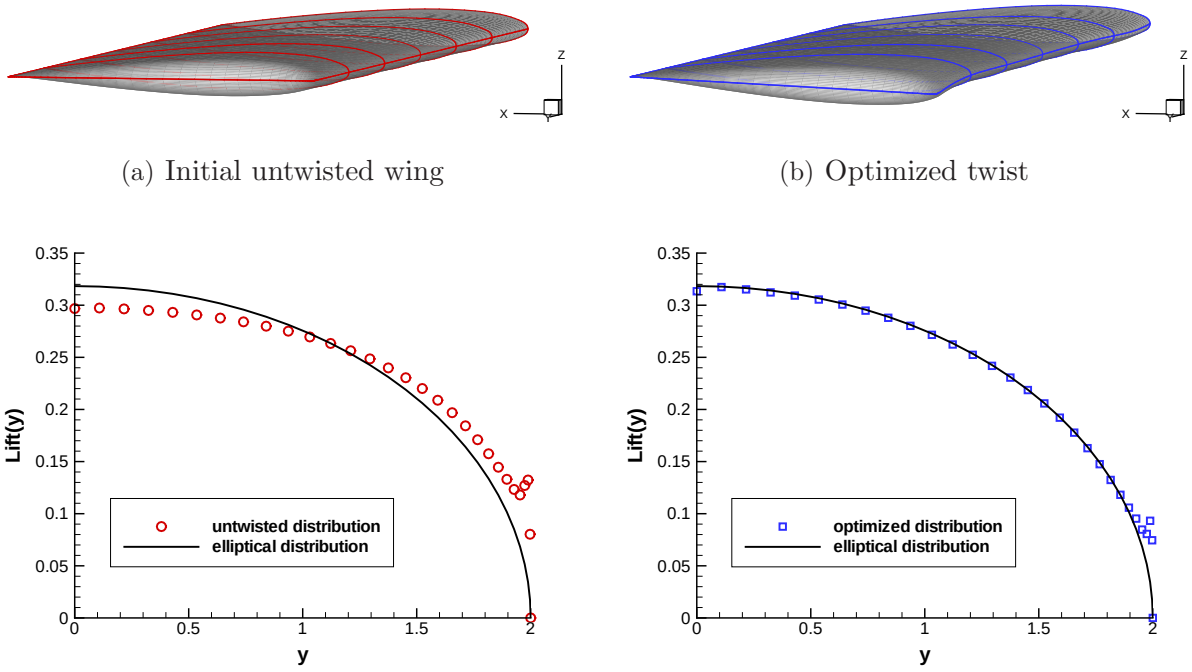
Figure 5.2: Convergence history for the twist optimization case.

by lifting line theory, based on $C_L = 0.375$, is $C_{D,i} = 0.00746$. Edge separation is likely responsible for the slightly lower drag produced by the optimal design, although numerical errors may also play a role.

Figure 5.3 shows the geometries and lift distributions for the initial and optimized designs. In contrast to the initial shape, the optimized shape exhibits a lift distribution close to elliptical. A small discrepancy between the elliptical and optimized distribution is visible at the wingtip. The increased sectional lift is caused by the tip vortex. The edge separation induces a non-planar wake, which implies that the elliptical lift distribution is no longer optimal [164]. In Section 5.3, the influence of edge separation is discussed further in the context of planform optimization.

5.3 Planform Optimization

The planform shape can also be used to produce an elliptical lift distribution; therefore, according to classical lifting-line theory for planar wakes, optimizing the planform should produce an elliptical shape. By including the effects of wake geometry, Smith and Kroo [165] have shown that the planform may not be exactly elliptical, depending on the shape of the trailing edge. Their results suggest we should expect some deviation from lifting-line theory in terms of planform shape, but not necessarily lift distribution. However, the elliptical lift



(a) Initial untwisted wing

(b) Optimized twist

(c) Initial distribution

(d) Optimized distribution

Figure 5.3: Initial and optimized designs and their lift distributions.

distribution itself may fail to be optimal if the effects of edge separation are included, since this will lead to a nonplanar wake [164].

As with the twist optimization, the initial geometry is a rectangular planform with a root chord of 2/3, semi-span of 2, and NACA 0012 sections. The 12 block grid from Table 5.1 is again used. The upper and lower surfaces of the wing are parameterized using 9 control points in the streamwise direction and 6 control points in the spanwise direction. Again, the trailing edge control points are fixed to reduce the effects of a nonplanar wake. The leading edge control points are free to move in the x-direction, and the remaining control points in each section are scaled based on the chord length; hence, the effective design variables are the chord lengths at six spanwise stations.

For this problem, it is necessary to use some form of root chord or area constraint to prevent non-unique optima, since there is an optimal planform for each angle of attack. For this example I adopt a surface/reference area constraint of $S = 1.36176$, the wetted area of

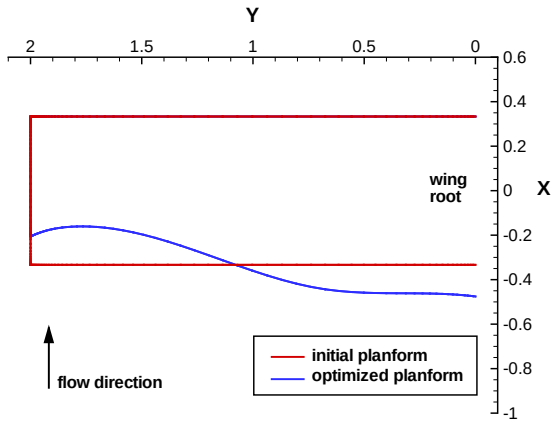


Figure 5.4: Initial and optimal planform shapes.

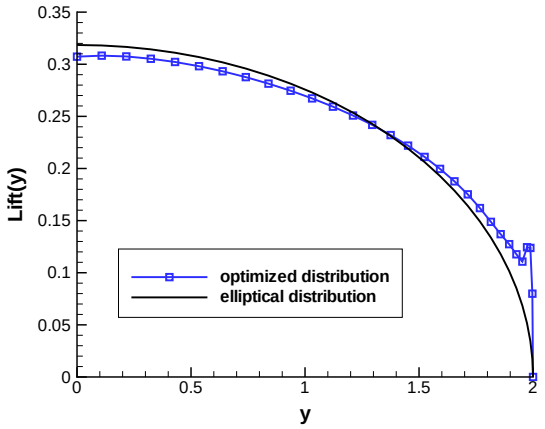


Figure 5.5: Elliptical lift distribution and distribution for optimal shape.

the initial planform shape.

The Mach number is fixed at 0.5, and the target lift coefficient is 0.36717. The initial angle of attack is set such that the lift constraint is satisfied at the first iteration. Plots showing the optimality, constraint violation, and merit function history can be found in Figure C.2 of Appendix C.

The optimal planform, which is clearly not elliptical, is plotted in Figure 5.4. As Figure 5.5 demonstrates, the lift distribution is not elliptical either, yet the coefficient of drag is 2% lower than predicted by lifting line theory: $C_D = 0.00715$ for the optimal geometry and $C_{D,i,\text{ellip}} = 0.00730$ for an elliptical lift distribution. There are two possible explanations for this result: either this is the optimal planform, or numerical errors are overwhelming the physics.

The 12 block grid has off-wall and edge spacing on the order of 10^{-3}. This resolution usually provides sufficient accuracy for subsonic inviscid flows. Nevertheless, I perform a grid refinement to investigate the role of numerical errors in the observed drag reduction. A refined grid for the optimal geometry is created by taking advantage of the analytical B-spline volume mappings, which permit changes in mesh resolution while conforming to the shape. The resolution is increased by a factor of 1.25 in each direction, relative to the initial grid, which produces a grid with 2.23×10^6 nodes. The flow analysis is repeated on the refined grid using the same dissipation coefficients and SAT parameters. The span

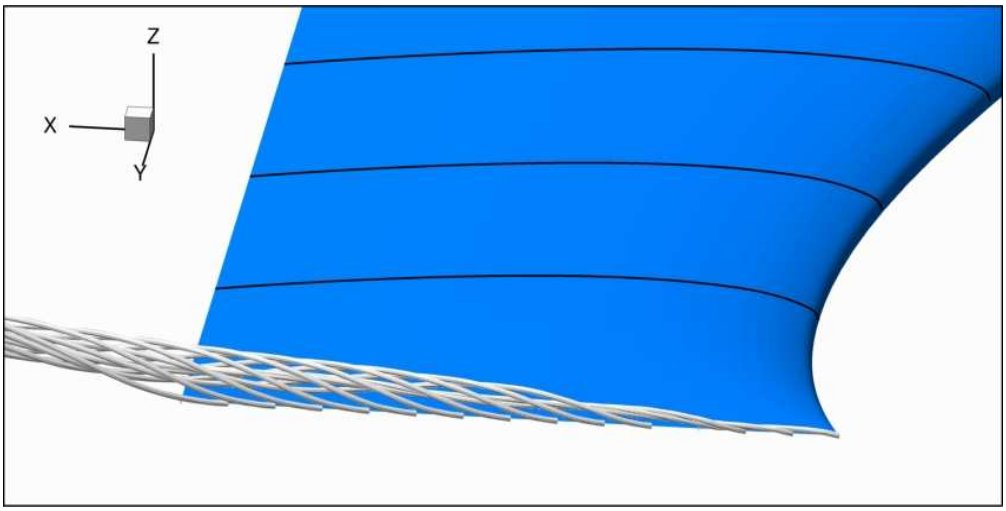


Figure 5.6: Wingtip view of the optimized planform with the tip vortex visualized with streamlines.

efficiencies for the original and refined grids are $e = 1.022$ and $e = 1.020$, respectively. This suggests that numerical errors are not responsible for the optimal planform shape.

A possible physical explanation is based on the edge separation and the resulting tip vortex. The tip vortex releases along the wing tip and curls onto the upper side of the wing, as shown in Figure 5.6. The flow field created by the vortex lowers the pressure on the upper surface near the tip, similar to the vortices on a delta-wing. Evidence for this effect is visible in the lift distribution in Figure 5.5, as well as the experimental results of [22]. As the wingtip chord decreases, the region of reduced pressure also decreases; hence, there appears to be a compromise between maintaining this vortex-induced low pressure region and establishing an elliptical lift distribution. The shape of the wake offers an alternative perspective. The edge separation produces a nonplanar wake, and the vertical extent of this wake is increased by extending the chord at the tip (a tip with finite chord has a vertical component when the wing is inclined at an angle of attack).

5.4 Spanwise Vertical Shape Optimization: Winglet Generation

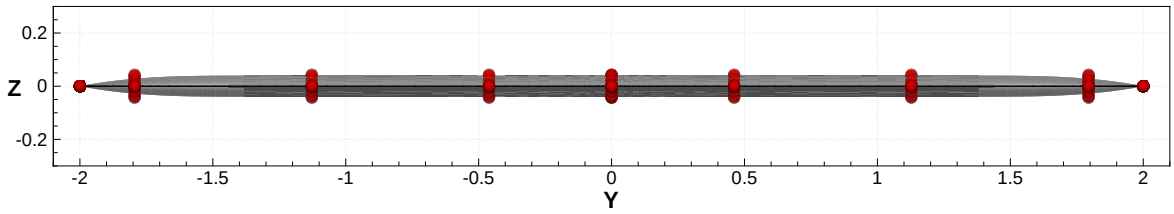
For a fixed span, nonplanar configurations can produce much lower induced drag than those with planar wakes. In the next three sections, I consider examples that exploit nonplanar wakes. In this section, I study the effects of spanwise vertical shape on induced drag.

Again, the initial geometry is a rectangular wing with NACA 0012 sections, a semi-span of 2, and chord length of 2/3. The 12 block grid listed in Table 5.1 is fit using B-spline volumes, with 9×5 control points on the upper and lower surfaces of the wing, in the streamwise and spanwise directions, respectively. The 5 spanwise control point sections are free to move in the vertical direction provided no control point exceeds the bounds $-0.2 \leq z \leq 0.2$. Permitting all spanwise stations to move introduces non-uniqueness in the design space, since many designs can be translated within the box constraints; however, the final designs are unique, because both the upper and lower bound constraints are active.

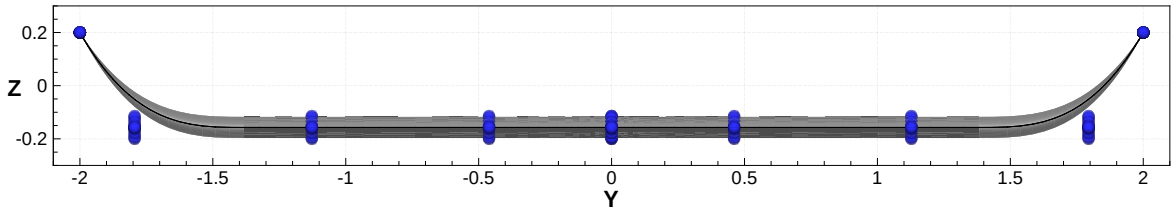
The Mach number is fixed at 0.5, and a C_L constraint of 0.375 is imposed, based on a reference (projected) area of 4/3. The initial angle of attack is set to satisfy the lift constraint. Convergence plots of optimality, constraint violation, and merit function are provided in Appendix C.2.2, Figure C.3.

The initial and optimized designs are shown in Figures 5.7(a) and 5.7(b), respectively. The optimized design has maximized the vertical extent near the tip, which is the "critical parameter" for nonplanar systems [97]. The initial geometry has an induced drag of $C_D = 0.00752$, while the final design produces $C_D = 0.00690$, a reduction of approximately 8%. Note that further reductions in drag would be possible if twist or planform changes were permitted to optimally load the configuration.

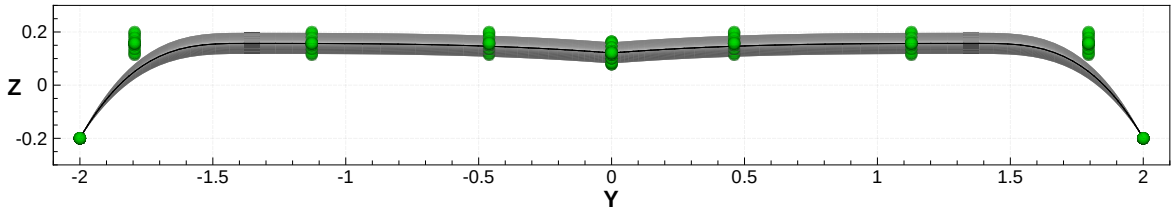
To identify possible local optima for this problem, I reran the optimization with the control points at the tip of the initial shape positioned at $z = -0.2$, thus creating a negative dihedral winglet. Indeed, this initial shape did lead to the distinct optimum shown in Figure 5.7(c); however, this configuration produces a drag reduction of only 3.9% ($C_D = 0.00723$). Eppler [36], using lifting-surface theory with induced-lift contributions, also concluded that "winglets up are much better than winglets down, whereas classical theories with rigid wake yield exactly the same (drag)." He attributed this difference to the horizontal component of the bound vortex increasing (decreasing) the distance between the



(a) Initial flat wing: $\mathbf{C_D = 0.00752}$



(b) Optimized spanwise vertical shape: $\mathbf{C_D = 0.00690}$



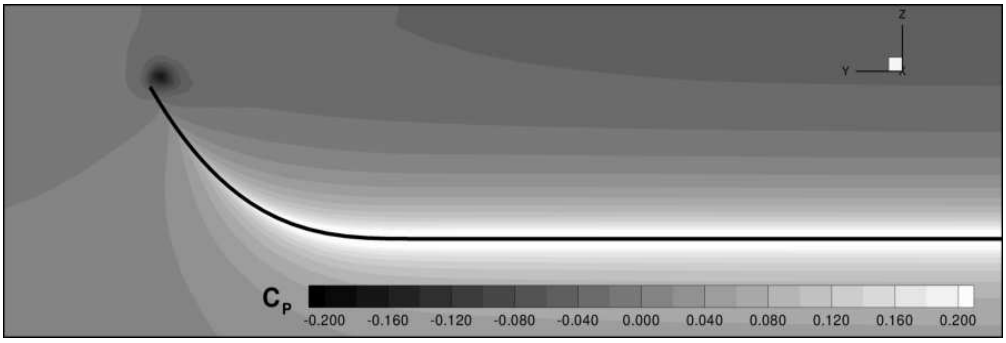
(c) Local optimum of spanwise vertical shape: $\mathbf{C_D = 0.00723}$

Figure 5.7: Initial and optimized designs for spanwise vertical shape.

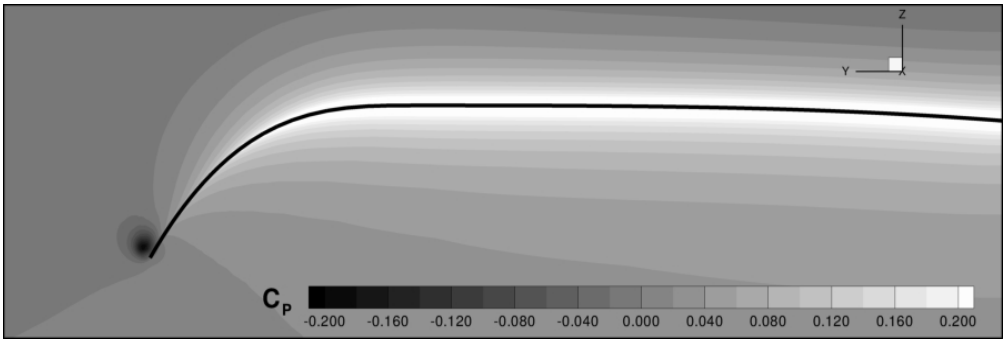
tip vortices for the winglet-up (-down) case, thus increasing (decreasing) the effective span.

Another possible mechanism is the vertical distance that the vortex is moved by the free-stream as the wake is shed from the tip. In the winglet-up case, the free-stream tends to increase the distance between the vortex and the wing. In contrast, the vortex shed from the winglet-down shape is swept closer to the wing in the vertical direction. Figure 5.8 demonstrates this asymmetry in the wake shape, by plotting the pressure contours at the trailing edge of the two configurations. The tip vortex locations, indicated by the concentrated regions of low pressure, are clearly not symmetric, and the vertical extent of the wake is clearly larger in the case of the winglet-up configuration.

There is some evidence that the optimal winglet dihedral is strongly influenced by viscous



(a) Winglet-up, pressure contours at the trailing edge



(b) Winglet-down, pressure contours at the trailing edge

Figure 5.8: Pressure contours on vertical spanwise planes at the trailing edge of the winglet-up and winglet-down configurations. The same contour levels are used in both figures.

effects. For example, Gerontakos and Lee [48] varied wingtip dihedral in an experimental investigation and found that the winglet-down case produced lower induced drag than the corresponding winglet-up case; however, they noted that there was an order of magnitude discrepancy between the induced drag predicted by lifting-line theory and the experimental results obtained using the Maskell wake survey method [109].

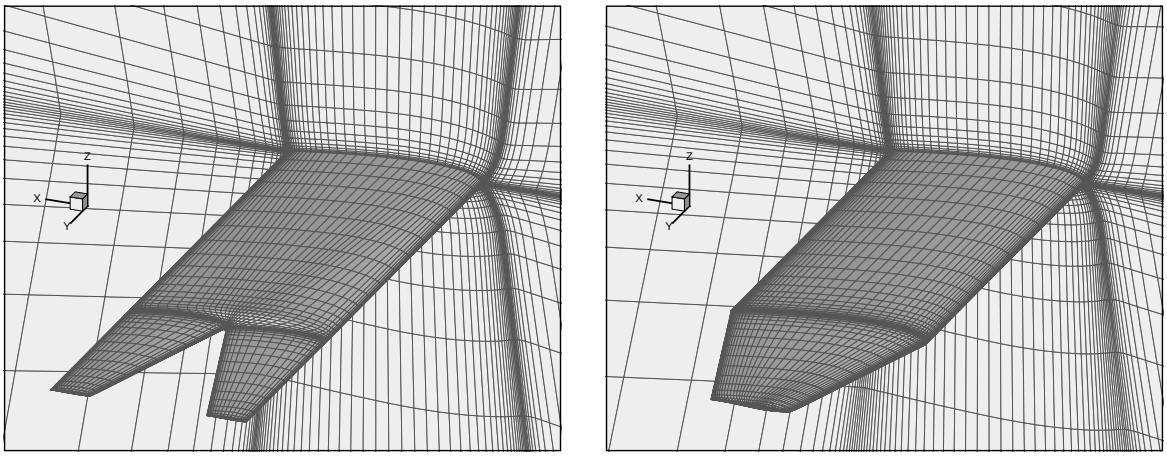
5.5 Split-tip Dihedral Optimization

In [164], Smith studied a split-tip configuration consisting of a main wing with two tip wings. The tip wings were staggered in the streamwise direction, and the rear tip wing was swept back. Although the split-tip configuration has a planar wake at zero angle of attack, the streamwise separation of the tip wings produces a nonplanar wake when the wing is inclined to the flow. Smith showed that a linear discrete-vortex method predicts that the split-tip configuration has a span efficiency of approximately $e = 1.066$, at an angle of attack of 9 degrees, i.e. a drag reduction of about 6%. He also analyzed the configuration using a nonlinear, force-free wake calculation. Remarkably, when nonlinear effects were included, the split-tip reduced the drag by 12% relative to an elliptical lift distribution. Inspired by Smith's results, I investigate the effect of dihedral on the induced drag of a split-tip configuration.

When dihedral is included, the wake from the split-tip is nonplanar, even at zero angle of attack. Benchmarking such a split-tip configuration with an optimally loaded planar wing is no longer appropriate. Instead, I compare the split-tip with a wing composed of two spanwise sections, a main wing and a single tip; essentially the split-tip configuration with the two tip wings replaced with one.

The initial split-tip geometry and grid are shown in Figure 5.9(a). The split-tip configuration has a semi-span of 3, and a chord of 1. The main wing has a rectangular planform, while the tip wings have taper ratios of 0.4. The reference area is the projected area of $S = 2.7$. The junction between the main wing and tip wings is located at two-thirds of the semi-span. The main wing and tip wings have NACA 0012 sections throughout most of the span, with some fairing required at the junction and at the wing tips.

The canonical grid for the split-tip configuration is the 42 block grid listed in Table 5.1. Each block in the grid is fit with a B-spline volume consisting of $6 \times 6 \times 5$ control points. The geometry is composed of 4 B-spline patches on the upper surface (2 for the main wing and 1 each for the tip wings), and 4 patches on the lower surface. Each patch is parameterized with 6×6 control points. Linear constraints couple the 212 control-point coordinates, and reduce the number of geometric design variables to 7: specifically, linear twist and dihedral of the main wing and two tip wings, plus the vertical translation of the configuration. The leading edge, trailing edge, and tip edges are constrained by the box constraint $|z| \leq 0.2$. As with



(a) Split-tip (b) Single-tip

Figure 5.9: Geometries and grids for the initial split-tip and single-tip geometries. Every other grid line is removed for clarity.

the spanwise vertical shape optimizations, allowing the configuration to translate introduces nonunique designs, but any design that activates the box constraints will be unique. The coefficient of lift is constrained to be 0.37.

Figure 5.9(b) shows the initial single-tip geometry and its corresponding grid. The single-tip geometry also has a semi-span of 3, and a chord of 1. Again, the main wing has a rectangular planform, while the tip wing has a taper ratio of 0.4. The junction between the main wing and the tip wing is at $y = 2$, and the wing is composed of NACA 0012 sections. The 32 block grid from Table 5.1 is used for the single-tip configuration. Consistent with the split-tip, the geometry is composed of 4 B-spline patches on the upper and lower surfaces, each parameterized using 6×6 control points. The 5 effective geometric design variables are the (linear) twist and dihedral of the main and tip wings, together with vertical translation. As with the split-tip, the edges of the wing are constrained vertically by $|z| \leq 0.2$, and a C_L constraint of 0.37 is imposed.

Figure 5.10 shows two local optima obtained from the split-tip optimization; note that additional local optima for this geometry may exist. The convergence and merit function histories for the optimizations leading to these optima can be found in Appendix C.2.3. I compare the split-tip optima with the winglet-up single-tip optimum, also shown in Figure 5.10, since the preceding study indicated that this would produce a lower drag than the winglet-down configuration.

The split-tip optimum in Figure 5.10(a) was obtained from the initial configuration shown in Figure 5.9(a). I will refer to this optimum as the up-down configuration, as a mnemonic for the positions of the forward and rear tip-wing dihedrals. Similarly, I will refer to the local optimum shown in Figure 5.10(b) as the down-up configuration. The optimization leading to the down-up configuration was initiated with the rear tip wing at maximum dihedral.

Both the split-tip geometries have maximized the vertical distance between the tip wings. The up-down configuration produces an induced drag value of $C_D = 0.00589$, while the down-up configuration gives $C_D = 0.00596$. The small difference in drag between the two split-tip optima can be explained by the horizontal separation of the tip wings. With positive angle of attack, the up-down configuration has a slightly larger height-to-span ratio than the down-up configuration.

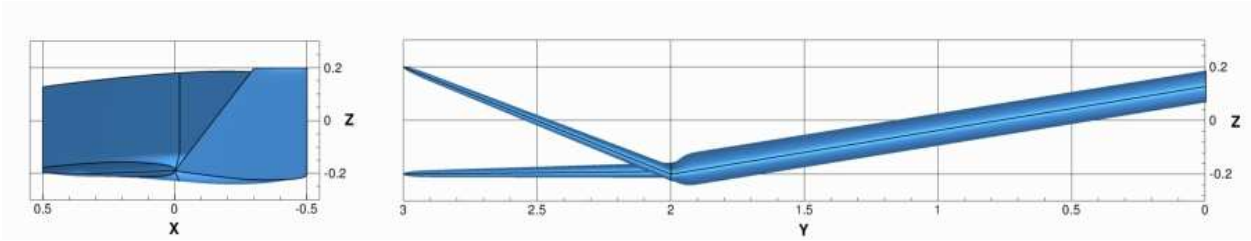
There is a striking difference in drag between the split-tip and single-tip optima. Relative to an elliptically loaded planar wing, the split-tip configurations have 9–10% lower drag, while the single-tip has only 2% lower drag. We can conclude that, for the same height-to-span ratio, a split-tip configuration has a larger span efficiency than a single-tip wing. Indeed, in terms of span efficiency, the split-tip wings outperform the winglet-up case from the previous study, despite that case having a larger height-to-span ratio. To be fair, including twist in the winglet-up case would diminish this performance gap.

5.6 Box-Wing Optimization

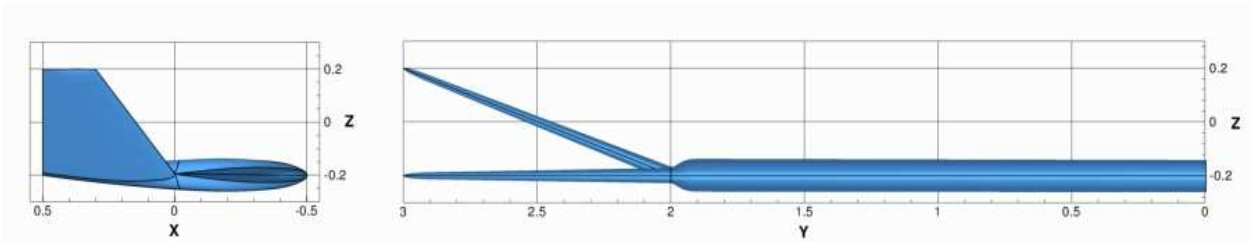
In this final example, I optimize the horizontal loading for a box-wing configuration. For closed systems, like the box-wing, the optimal loading is not unique, since a vortex loop of constant circulation can be added to the bound vortex [97]. The addition of the vortex loop does not alter the drag or lift, but it can be used to change the lift distribution. For this reason, box-wing configurations offer considerable design flexibility.

The initial box-wing geometry has a semi-span of 3.065 and chord length of 1. The initial height to span ratio is 0.1. The 6 block grid from Table 5.1 surrounds the box-wing geometry with approximately 6.02×10^5 nodes. The surface is parameterized using 9 control points in the streamwise direction and 5 control points in the spanwise and vertical directions. The Poisson's ratio used in the mesh movement algorithm was set to $\nu = -0.2$.

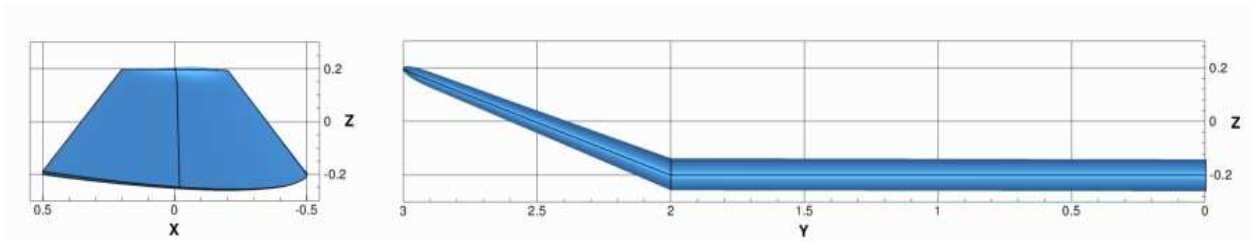
The vertical surfaces are linearly constrained by the upper and lower horizontal surfaces.



(a) Optimized split-tip configuration: $\mathbf{C_D = 0.00589}$ and $\mathbf{e = 1.11}$.



(b) Local optimum of split-tip configuration: $\mathbf{C_D = 0.00596}$ and $\mathbf{e = 1.10}$.



(c) Optimized single-tip configuration: $\mathbf{C_D = 0.00641}$ and $\mathbf{e = 1.02}$.

Figure 5.10: Optimal split-tip and single-tip geometries together with their drag coefficients and span efficiencies.

Along the horizontal surfaces, the leading and trailing edge control points are free to move vertically within the box constraint $|z| \leq 0.315$. A translation constraint is imposed by forcing the upper and lower leading edges at the root to have an average z-coordinate of 0.

Based on the above constraints, the effective design variables are the twist and vertical position of the 5 spanwise sections along the upper and lower surfaces. Accounting for the translation constraint, this provides 19 degrees of freedom; however, the gradient tends to push the sections to their upper and lower bounds, so only about half of these degrees of freedom are useful in practice.

The lift coefficient constraint is 0.5, based on a projected area of 3. This constraint is satisfied initially using an angle of attack of 4.13486 degrees. The free-stream Mach number is 0.5.

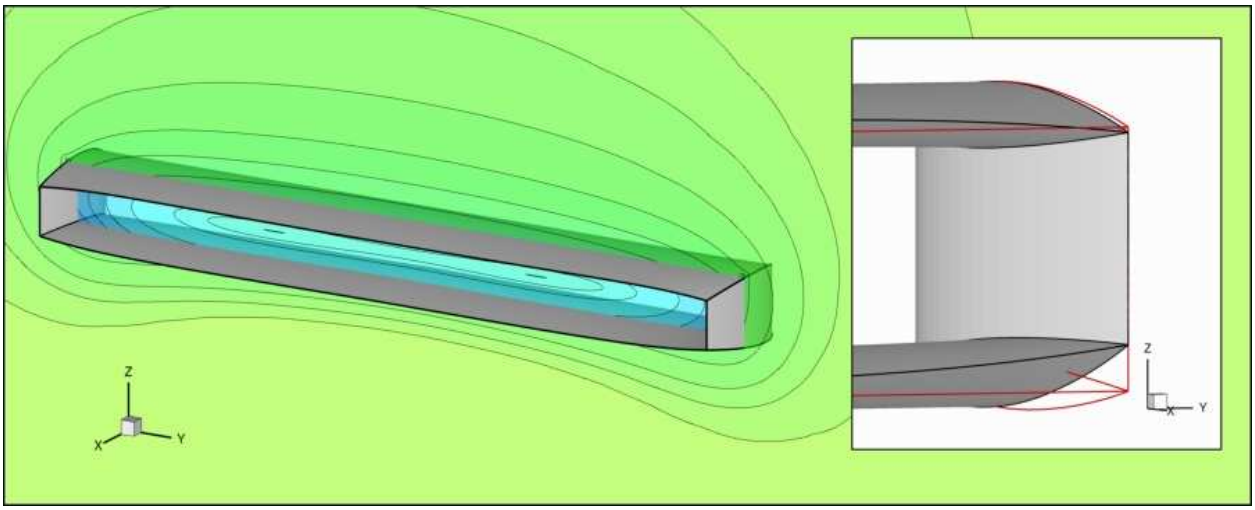


Figure 5.11: Optimized box-wing configuration with pressure contours on $x = 0$ slice. Inset: tip detail showing trailing edge shape of the optimized configuration. The edges of the initial geometry are shown as red lines in the inset.

The convergence history for this problem is provided in Figure C.7 of Appendix C.2.4. SNOPT has converged the optimality conditions by approximately 3.5 orders over 53 function evaluations. However, the optimizer was unable to reduce the optimality below the requested tolerance of 10^{-7}; SNOPT stalled at an optimality value of 7.1×10^{-7}. The flow adjoint tolerance was decreased to 10^{-9}, but this did not eliminate the stagnation. This convergence difficulty may be related to the non-unique design space producing a singular Hessian.

Figure 5.11 shows the pressure contours around the final design. The drag coefficient has been reduced 7% from 0.0125 to 0.0116. For a planar configuration with the same lift, lifting-line theory predicts an optimal drag coefficient of 0.0127; hence, relative to a planar system, the optimized box-wing has reduced the drag by 8.7%.

Remarkably, the optimized split-tip geometries have lower induced drags than the box-wing (see previous section), despite the larger height-to-span ratio of the box-wing. Clearly, nonlinear wake-wing interactions play an important role in induced drag reduction.

Chapter 6

Conclusions, Contributions, and Recommendations

I use this final chapter to summarize Chapters 2 through 5, and review major contributions and conclusions. These summaries can be found in the following section. In addition, I provide a list of recommendations for future work in Section 6.2, and some concluding remarks in Section 6.3.

6.1 Conclusions and Contributions

6.1.1 Integrated Parameterization and Mesh Movement

I have shown that B-spline mesh movement produces grids with quality comparable to those obtained using a node-based mesh movement, while requiring two to three orders less CPU time. In addition, the integrated approach significantly reduces the cost of the mesh-adjoint solution, relative to a node-based linear elasticity mesh movement. For typical-size grids, the mesh-movement and mesh-adjoint solutions represent less than 5% of the total computational work, despite considerable use of complex-step differentiation.

B-spline volume mappings have many potential applications beyond parameterization and mesh movement. For example, they may be of use in mesh adaptation and refinement. The control points can be moved according to error estimates; this approach should be faster and more robust than moving individual points, given the properties of B-splines. When applied to convergence studies, these mappings provide an explicit transformation from parameter space to computational space, permitting a smooth refinement strategy. Using the same strategy, hierarchical grids can be constructed for multigrid. More generally, one can apply a Cartesian adaptive strategy in parameter space to obtain boundary-fitted unstructured

meshes. Finally, by adjusting the order of the B-spline basis functions, we can produce grids with sufficient smoothness for high-order discretizations.

Contributions: I have used B-spline volumes to integrate geometry parameterization with mesh movement. Specifically, each block is associated with a B-spline volume, and the control points on the geometry surface become design variables. When the geometry changes, the B-spline control grid is perturbed using a robust mesh movement algorithm (linear elasticity), and the grid used for flow analysis is regenerated algebraically. In short, mesh movement is made more efficient by reducing the number of degrees of freedom associated with the grid.

6.1.2 Numerical Solution of the Euler Equations

I have implemented a second-order SBP-SAT discretization for the Euler equations. The SBP-SAT discretization can accommodate grids with C^0 continuity at the block interfaces. Thus, multi-block grids generated algebraically, including B-spline volume grids, can be used without global smoothing of the grid. The SAT interface treatment also simplifies the stencil for nodes along block edges and vertices, which, in turn, reduces the complexity of the implicit algorithm.

I conclude that a SBP-SAT discretization of the Euler equations is well suited to a parallel Newton-Krylov solution strategy. In addition, for the Euler equations, the approximate-Schur preconditioner outperforms the additive-Schwarz preconditioner in terms of Krylov iterations, but the preconditioners have similar CPU time requirements. The Schur preconditioner may become more attractive for high-order SBP operators, since the size of the interface system will remain the same. The Schur preconditioner may also perform better for viscous discretizations when using a Jacobian-free matrix-vector product, since the higher cost of evaluating the product will favour the preconditioner that requires fewer Krylov iterations. The Schwarz preconditioner remains a good choice when memory is limited, since the approximate-Schur preconditioner must use a flexible iterative solver (e.g. FGMRES), which often requires more memory.

Contributions: I have developed a novel parallel Newton-Krylov algorithm to solve the discrete equations. To my knowledge, all prior work with SBP-SAT discretizations

has used explicit time-marching methods, so this thesis represents the first example of an implicit solution strategy applied to this discretization. This is significant, because SAT penalties, when used at block interfaces, reduce the maximum stable CFL number of explicit schemes.

An efficient parallel preconditioner was required as part of the Newton-Krylov algorithm. To this end, I modified the approximate-Schur preconditioner of Saad and Sosonkina [153] and, consequently, improved its performance. This preconditioner is an attractive choice for the SBP-SAT approach, since the size of the Schur complement is independent of the discretization order.

6.1.3 Gradient-Based Optimization

I have implemented a sequential approach to evaluate the objective function gradient using adjoint variables. The algorithm is sequential in the sense that the flow and mesh adjoints must be solved in a reverse sequence, relative to the forward objective evaluation. The flow adjoint equation is solved using a variant of GCROT, while the mesh adjoint equations are solved using CG. The optimization process is governed by the SNOPT software package, which uses an SQP framework.

An accurate Jacobian matrix is required for both the flow- and mesh-adjoint equations. In the case of the flow adjoint, the matrix is calculated using a combination of analytical and complex-step differentiation. I have shown that the accuracy of the flow Jacobian can be easily verified by comparing with a complex-step matrix-vector product, since this product only requires a complex version of the residual evaluation. The complex-step method is also useful for differentiating the complicated nonlinear terms in the mesh adjoint equations; however, in this context, the use of complex-step differentiation is only practical due to the small size of the B-spline mesh.

The gradient of the objective function, evaluated using the sequential-adjoint approach, has been verified by comparing with finite-difference approximations of a directional derivative. The gradient has been shown to be as accurate as the finite-difference approximation with optimal step sizes.

Contributions: To facilitate the solution of the flow adjoint equations, I have developed a variant of the Krylov iterative solver GCROT. This variant, called GCROT(m,δ),

is flexible and can use the approximate-Schur preconditioner; this is an important extension of the solver given the number of matrix-vector products required to solve the adjoint equations. I have shown that GCROT(m,δ) is at least as effective as restarted FGMRES when the same amount of memory is available to both solvers. In addition, I have demonstrated that in some cases FGMRES(m) may stall while GCROT(m,δ) does not.

6.1.4 Studies of Induced Drag Minimization

I have demonstrated that high-fidelity Euler-based optimizers can be validated using twist optimization to recover an elliptical lift distribution. This is a simple, yet frequently overlooked, case. It is at least as important as inverse design.

In contrast to twist optimization, I have shown that planform optimization cannot be used to recover an elliptical lift distribution. The tip vortex creates a nonplanar wake, and the vertical extent of this wake is controlled by the tip chord. The optimizer balances the benefits of a finite tip chord with the lift distribution, producing a planform with lower induced drag than the elliptical planform.

I have recovered a winglet-up configuration as a local optimum of spanwise vertical shape optimization. A winglet-down configuration was also found to be a local optimum, but its drag reduction was less than half that of the winglet-up configuration. According to classical lifting-line theory, these configurations produce the same drag, so this result provides further evidence that high-fidelity optimization is useful in preliminary design.

A split-tip configuration produces lower induced drag than a single-tip configuration, for a given height-to-span ratio. This confirms the findings in [164], that non-linear wake interactions have a significant impact on induced drag, beyond that predicted by streamwise wake models.

Finally, I have applied the optimizer to the box-wing, which demonstrates that it can be used on complex configurations.

Contributions: I have shown that an Euler-based solver can be used for high-fidelity optimization of induced drag. Unlike classical streamwise-wake models, an Euler solver accounts for nonlinear wake-wing interactions and side-edge separation. Thus, induced-drag minimization based on the Euler equations can provide insights that might oth-

erwise require significant expertise. This is exemplified by the planform optimization case.

The optimized configurations of Chapter 5 contribute to our understanding of induced drag. Table 6.1 summarizes some of the statistics for these configurations. The split-tip is particularly remarkable, because it outperforms the box-wing. Recall that the box-wing is predicted by lifting-line theory to have the largest span efficiency for a given height-to-span ratio.

Table 6.1: Summary of the induced-drag minimization studies.

study	**span**	$\|\Delta z\|^{\dagger}$	**area**	C_L	C_D	e
twist	4	—	2.667	0.375	0.00739	1.010
planform‡	4	—	2.724	0.368	0.00718	1.020
winglet-down	4	0.4	2.667	0.375	0.00723	1.032
winglet-up	4	0.4	2.667	0.375	0.00690	1.081
split-tip down-up	6	0.4	5.400	0.370	0.00596	1.099
split-tip up-down	6	0.4	5.400	0.370	0.00589	1.111
box-wing	6.13	0.63	6.000	0.500	0.01159	1.096

† vertical height constraint

‡ values from refined grid

6.2 Recommendations

Block decomposition: For large problems, subdividing blocks to allow for adequate load balancing will become essential. This can easily be accomplished using B-spline meshes, since these provide an explicit mapping between computational and physical space.

Nonlinear Mesh Movement: The mesh movement algorithm is relatively robust, but may fail in some circumstances. The optimization process often produces large changes in the design, which is inconsistent with the infinitesimal perturbations assumed by

linear elasticity. While breaking the movement into increments provides a partial solution, nonlinear elasticity and the theory of finite deformations should be investigated.

Multi-level Preconditioner: The preconditioners considered in this thesis do not scale ideally as the number of unknowns is increased. To achieve better scaling, multi-level preconditioners such as multi-grid, multi-level Schwarz, or multi-level ILU should be considered.

Viscous SATs: A flow solver that includes viscous effects and turbulence modelling should be developed as the next step in the research programme. For the viscous SATs, derivatives of the flow variables will be required, which may increase the number of internal-interface variables. I recommend solving for these derivatives at the interface, i.e. by adding additional equations to the residual. This will avoid increasing the size of the reduced system in the Schur-complement preconditioner for high-order viscous discretizations.

High-Order SBP Operators: High-order discretizations may improve the efficiency of the flow solver, and should be investigated for the Euler and RANS equations. In particular, I suspect there are significant advantages to increasing the accuracy of the one-sided discretizations used by SBP operators at block boundaries.

Jacobian-Free Adjoint: Using a Jacobian-free adjoint approach may become necessary for high-order optimization due to memory considerations. Such an approach can be achieved using Krylov solvers, since they require only transposed-Jacobian-vector products. As already mentioned, the product $A^T\mathbf{v}$ can be evaluated by applying reverse-mode algorithmic differentiation to the residual calculation; however, following Barth and Linton [8], a hand-coded transposed product may be more efficient in the short term.

Adaptive Tolerances: The mesh movement, flow solver, and adjoint equations are solved to relatively small tolerances throughout the optimization. As discussed in Chapter 4, this is required for accurate gradient evaluations. However, during the early iterations of optimization, small tolerances may be inefficient and an approach based on inexact-Newton methods may be advantageous. Incorporating adaptive tolerances into an

existing optimizer may be difficult, since black-box optimizers often expect accurate gradients, even during early iterations. In contrast, a one-shot method may be well-suited to such an inexact-Newton approach.

One-Shot Method: A one-shot method could be constructed using the algorithms described in this thesis as preconditioners. Such an approach may be worth pursuing given the convergence properties of one-shot methods.

Hybrid Optimizer: The induced-drag studies have demonstrated that the optimizer must handle multi-modal design spaces. Rather than abandoning gradient-free optimization in favour of stochastic methods, I recommend developing a hybrid algorithm. There appears to be significant scope for improving such hybrid algorithms.

Unsteady Flows: Implicit time-marching schemes can be coupled with a Newton-Krylov strategy, resulting in efficient algorithms for time-dependent problems [172]. Using this approach, the steady solver described in this thesis could easily be extended to handle unsteady flows. This would be worth pursuing in order to investigate, for example, acoustic waves and direct/large-eddy simulation of turbulent flows. The possibility of implementing Rumpfkeil's [148, 147] unsteady optimization framework in 3-dimensions could also be explored.

Multi-point Optimization: Aircraft operate over a range of conditions, and must be efficient at multiple Mach numbers and lift coefficients. Therefore, the present optimizer must be extended to handle multi-point optimization.

Multi-disciplinary Design Optimization: The flow solver, mesh movement, and aerodynamic sensitivities (gradients) should be coupled with other disciplines to investigate unconventional configurations in a high-fidelity MDO framework [107]. This is necessary to complete the research programme introduced by this thesis.

6.3 Epilogue

Over the next few decades, the aviation industry will experience an unprecedented rise in fuel costs, triggered by peak oil and emissions-based trading or taxes. Alternative fuels and

operational changes will offer only partial answers; ultimately, new aircraft configurations will be needed. This leads to the question motivating this thesis: to what extent can numerical optimization be used to automatically design unconventional aircraft?

As a first step toward answering this question, I have developed a robust and efficient optimizer for the Euler equations, and used this optimizer to study induced drag minimization. The components of the optimizer — the integrated parameterization and mesh movement, the inviscid flow solver, and the gradient evaluation — can be used as modules in multidisciplinary shape optimization. In particular, the components are intended to study the possibility of using optimization in whole aircraft, high-fidelity aircraft design, and to permit sufficient flexibility to explore novel aircraft configurations.

References

[1] M. J. Aftosmis, *Lecture notes for the 28th computational fluid dynamics lecture series: solution adaptive Cartesian grid methods for aerodynamic flows with complex geometries*, tech. rep., von Kármán Institute for Fluid Dynamics, Rhode-Saint-Genèse, Belgium, Mar. 1997.

[2] C. B. Allen and T. C. S. Rendall, *Unified approach to CFD-CSD interpolation and mesh motion using radial basis functions*, in 25th AIAA Applied Aerodynamics Conference, no. AIAA–2007–3804, Miami, Florida, United States, June 2007.

[3] S. Aly, M. Ogot, and R. Pelz, *Stochastic approach to optimal aerodynamic shape design*, Journal of Aircraft, 33 (1996), pp. 956–961.

[4] J. D. Anderson, *Fundamentals of Aerodynamics*, McGraw–Hill, Inc., New York, NY, third ed., 2001.

[5] W. K. Anderson and D. L. Bonhaus, *Airfoil design on unstructured grids for turbulent flows*, AIAA Journal, 37 (1999), pp. 185–191.

[6] W. K. Anderson, R. D. Rausch, and D. L. Bonhaus, *Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids*, Journal of Computational Physics, 128 (1996), pp. 391–408, doi:10.1006/jcph.1996.0219.

[7] P. Z. Bar-Yoseph, S. Mereu, S. Chippada, and V. J. Kalro, *Automatic monitoring of element shape quality in 2-D and 3-D computational mesh dynamics*, Computational Mechanics, 27 (2001), pp. 378–395.

[8] T. J. Barth and S. W. Linton, *An unstructured mesh Newton solver for compressible fluid flow and its parallel implementation*, in 33rd AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–95–0221, Reno, Nevada, 1995.

[9] J. T. Batina, *Unsteady Euler airfoil solutions using unstructured dynamic meshes*, AIAA Journal, 28 (1990), pp. 1381–1388.

[10] O. Baysal and M. E. Eleshaky, *Aerodynamic sensitivity analysis methods for the compressible Euler equations*, Journal of Fluids Engineering, 113 (1991), pp. 681–688.

[11] L. K. Billing, *On the development of an improved lift-constrained aerodynamic optimization algorithm*, Master's thesis, University of Toronto, Toronto, Ontario, Canada, 2006.

[12] M. Blanco and D. W. Zingg, *Fast Newton-Krylov method for unstructured grids*, AIAA Journal, 36 (1998), pp. 607–612.

[13] F. J. Blom, *Considerations on the spring analogy*, International Journal for Numerical Methods in Fluids, 32 (2000), pp. 647–688, doi:10.1002/(SICI)1097-0363(20000330)32:6<647::AID-FLD979>3.0.CO;2-K.

[14] P. Bogers, *The role of alternative fuels in air transport — options and challenges*, in UTIAS-MITACS International Workshop on Aviation and Climate Change, Toronto, Canada, May 2008, http://goldfinger.utias.utoronto.ca/~IWACC/Program.html.

[15] O. P. Brunoa, Y. Hana, and M. M. Pohlmanb, *Accurate, high-order representation of complex three-dimensional surfaces via Fourier continuation analysis*, Journal of Computational Physics, 227 (2007), pp. 1094–1125.

[16] G. W. Burgreen and O. Baysal, *Three-dimensional aerodynamic shape optimization using discrete sensitivity analysis*, AIAA Journal, 34 (1996), pp. 1761–1770.

[17] R. H. Byrd, N. I. Gould, J. Nocedal, and R. A. Waltz, *An algorithm for nonlinear optimization using linear programming and equality constrained subproblems*, Mathematical Programming, 100 (2004), pp. 27–48, doi:10.1007/s10107-003-0485-4.

[18] X.-C. Cai, W. D. Gropp, D. E. Keyes, and M. D. Tidriri, *Newton-Krylov-Schwarz methods in CFD*, in International Workshop on Numerical Methods for the Navier-Stokes Equations, F. Hebeker and R. Rannacher, eds., Notes in Numerical Fluid Mechanics, Braunschweig, 1993, Vieweg Verlag, pp. 17–30.

[19] M. H. Carpenter, D. Gottlieb, and S. Abarbanel, *Time-stable boundary conditions for finite-difference schemes solving hyperbolic systems: methodology and application to high-order compact schemes*, Journal of Computational Physics, 111 (1994), pp. 220–236, doi:10.1006/jcph.1994.1057.

[20] M. H. Carpenter, J. Nordström, and D. Gottlieb, *A stable and conservative interface treatment of arbitrary spatial accuracy*, Journal of Computational Physics, 148 (1999), pp. 341–365, doi:10.1006/jcph.1998.6114.

[21] P. Castonguay and S. K. Nadarajah, *Effect of shape parameterization on aerodynamic shape optimization*, in The 45th AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–2007–0059, Reno, Nevada, 2007.

[22] N. A. Chigier and V. R. Corsiglia, *Tip vortices — velocity distributions*, Tech. Rep. NASA/TM X-62087, National Aeronautics and Space Administration, Ames Research Center, Moffett Field, CA, 94035–1000, Sept. 1971.

[23] T. T. Chisholm, *A Fully Coupled Newton-Krylov Solver With a One-Equation Turbulence Model*, PhD thesis, University of Toronto, Toronto, Ontario, Canada, 2007.

[24] T. T. Chisholm and D. W. Zingg, *A Newton-Krylov algorithm for turbulent aerodynamic flows*, in 41st AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA 2003–0071, Reno, Nevada, 2003.

[25] J. S. Chow, G. G. Zilliac, and P. Bradshaw, *Mean and turbulence measurements in the near field of a wingtip vortex*, AIAA Journal, 35 (1997), pp. 1561–1567.

[26] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Mathematics of Computation, 50 (1988), pp. 39–430.

[27] W. C. Davidon, *Variable metric method for minimization*, SIAM Journal on Optimization, 1 (1991), pp. 1–17.

[28] S. De Rango and D. W. Zingg, *Higher-order spatial discretization for turbulent aerodynamic computations*, AIAA Journal, 39 (2001), pp. 1296–1304.

[29] E. de Sturler, *Nested Krylov methods based on GCR*, Journal of Computational and Applied Mathematics, 67 (1996), pp. 15–41.

[30] ——, *Truncation strategies for optimal Krylov subspace methods*, SIAM Journal on Numerical Analysis, 36 (1999), pp. 864–889.

[31] K. S. Deffeyes, *Hubbert's peak — the impending world oil shortage*, Princeton University Press, Princeton, NJ, 2001.

[32] C. Degand and C. Farhat, *A three-dimensional torsional spring analogy method for unstructured dynamic meshes*, Computers and Structures, 80 (2002), pp. 305–316, doi:10.1016/S0045-7949(02)00002-0.

[33] J. Driver and D. W. Zingg, *Numerical aerodynamic optimization incorporating laminar-turbulent transition prediction*, AIAA Journal, 45 (2007), pp. 1810–1818.

[34] M. V. Dyke, ed., *Album of Fluid Motion*, The Parabolic Press, Stanford, California, tenth ed., 1982.

[35] S. C. Eisenstat and H. F. Walker, *Choosing the forcing terms in an inexact Newton method*, SIAM Journal on Scientific Computing, 17 (1996), pp. 16–32.

[36] R. Eppler, *Induced drag and winglets*, Aerospace Science and Technology, 1 (1997), pp. 3–15.

[37] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne, *Torsional springs for two-dimensional dynamic unstructured fluid meshes*, Computer Methods in Applied Mechanics and Engineering, 163 (1998), pp. 231–245.

[38] G. E. Farin, *Curves and surfaces for computed aided geometric design: a practical guide*, Academic Press, Inc., London, UK, fourth ed., 1997.

[39] M. Farmery, *Future aviation fuels*, in ICAO/Transport Canada Workshop, Montreal, Canada, Sept. 2006.

[40] D. Feng and T. H. Pulliam, *An all-at-once reduced hessian SQP scheme for aerodynamic design optimization*, Tech. Rep. RIACS-TR-95.19, NASA Ames Research Center, Moffett Field, California, Oct. 1995.

[41] K. J. Fidkowski and D. L. Darmofal, *A triangular cut-cell adaptive method for high-order discretizations of the compressible Navier-Stokes equations*, Journal of Computational Physics, 225 (2007), pp. 1653–1672.

[42] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Mathematical Programming, 91 (2002), pp. 239–269, doi:10.1007/s101070100244.

[43] P. D. Frank and G. R. Shubin, *A comparison of optimization-based approaches for a model computational aerodynamics design problem*, Journal of Computational Physics, 98 (1992), pp. 74–89.

[44] D. M. Fudge, D. W. Zingg, and R. Haimes, *A CAD-free and CAD-based geometry control system for aerodynamic shape optimization*, in The 43rd AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–2005–0451, Jan. 2005.

[45] D. Funaro and D. Gottlieb, *A new method of imposing boundary conditions in pseudospectral approximations of hyperbolic equations*, Mathematics of Computation, 51 (1988), pp. 599–613.

[46] P. J. Gage, I. M. Kroo, and I. P. Sobieski, *Variable-complexity genetic algorithm for topological design*, AIAA Journal, 33 (1995), pp. 2212–2217.

[47] J. Gatsis and D. W. Zingg, *A fully-coupled Newton-Krylov algorithm for aerodynamic optimization*, in 16th AIAA Computational Fluid Dynamics Conference, no. AIAA–2003–3956, Orlando, Florida, United States, June 2003.

[48] P. Gerontakos and T. Lee, *Effects of winglet dihedral on a tip vortex*, Journal of Aircraft, 43 (2006), pp. 117–124.

[49] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: an SQP algorithm for large-scale constrained optimization*, SIAM Journal on Optimization, 12 (2002), pp. 979–1006.

[50] D. Goodstein, *Out of Gas: the End of the Age of Oil*, W. W. Norton, 2004.

[51] J. E. Green, *Civil aviation and the environmental challenge*, The Aeronautical Journal, (2003), pp. 281–299.

[52] ——, *Civil aviation and the environment — the next frontier for the aerodynamicist*, The Aeronautical Journal, (2006), pp. 469–486.

[53] A. Griewank, *Evaluating Derivatives*, SIAM, Philadelphia, PA, 2000.

[54] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, *High-performance parallel implicit CFD*, Parallel Computing, 27 (2001), pp. 337–362, doi:10.1016/S0167-8191(00)00075-2.

[55] C. P. Groth and S. A. Northrup, *Parallel implicit adaptive mesh refinement scheme for body-fitted multi-block mesh*, in 17th AIAA Computational Fluid Dynamics Conference, no. AIAA–2005–5333, Toronto, Ontario, Canada, June 2005.

[56] B. Gustafsson, *The convergence rate for difference approximations to mixed initial boundary value problems*, Mathematics of Computation, 29 (1975), pp. 396–406.

[57] ——, *On the implementation of boundary conditions for the method of lines*, BIT, 38 (1998), pp. 293–314.

[58] P. Hajela, *Genetic search — an approach to the nonconvex optimization problem*, AIAA Journal, 28 (1990), pp. 1205–1210.

[59] P. Hajela, *Nongradient methods in multidisciplinary design optimization — status and potential*, Journal of Aircraft, 36 (1999), pp. 255–265.

[60] J. G. Hayes, *Numerical Analysis*, Springer Berlin/ Heidelberg, 1982, ch. Curved knot lines and surfaces with ruled segments, pp. 140–156.

[61] S. B. Hazra, *Direct treatment of state constraints in aerodynamic shape optimization using simultaneous pseudo-time-stepping*, AIAA Journal, 45 (2007), pp. 1988–1997.

[62] J. S. Hesthaven, *A stable penalty method for the compressible Navier-Stokes equations: III. multidimensional domain decomposition schemes*, SIAM Journal on Scientific Computing, 20 (1998), pp. 62–93.

[63] J. S. Hesthaven and D. Gottlieb, *A stable penalty method for the compressible Navier-Stokes equations: I. open boundary conditions*, SIAM Journal on Scientific Computing, 17 (1996), pp. 579–612.

[64] J. E. Hicken and D. W. Zingg, *An assessment of parallel preconditioners for a Newton-Krylov algorithm*, in 12$^{\text{th}}$ Aerodynamics Symposium, Canadian Aeronautics and Space Institute, Toronto, Ontario, Apr. 2007.

[65] ——, *Facilitating multi-block discretizations via simultaneous approximation terms: a Newton-Krylov perspective*, in 15$^{\text{th}}$ Annual Conference of the CFD Society of Canada, Toronto, Ontario, May 2007.

[66] ——, *A parallel Newton-Krylov flow solver for the Euler equations on multi-block grids*, in 18th AIAA Computational Fluid Dynamics Conference, no. AIAA–2007–4333, Miami, Florida, United States, June 2007.

[67] ——, *Integrated parametrization and grid movement using B-spline meshes*, in The 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, no. AIAA–2008–6079, Victoria, British Columbia, Canada, 2008.

[68] ——, *An investigation of induced drag minimization using a parallel Newton-Krylov algorithm*, in The 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, no. AIAA-2008-5807, Victoria, British Columbia, Canada, 2008.

[69] ——, *A parallel Newton-Krylov solver for the Euler equations discretized using simultaneous approximation terms*, AIAA Journal, 46 (2008), pp. 2773–2786.

[70] R. M. Hicks and P. A. Henne, *Wing design by numerical optimization*, Journal of Aircraft, 15 (1978), pp. 407–412.

[71] R. M. Hicks, E. M. Murman, and G. N. Vanderplaats, *An assessment of airfoil design by numerical optimization*, Tech. Rep. NASA/TM X-3092, National Aeronautics and Space Administration, Ames Research Center, Moffett Field, CA, 94035–1000, July 1974.

[72] C. Hirsch, *Numerical Computation of Internal and External Flows*, John Wiley & Sons, Chichester, England, 1990.

[73] R. Hixon, *Numerically consistent strong conservative grid motion for finite difference schemes*, AIAA Journal, 38 (2000), pp. 1586–1593.

[74] J. H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, Michigan, 1975.

[75] J. Hoschek, *Intrinsic parametrization for approximation*, Computer Aided Geometric Design, 5 (1988), pp. 27–31.

[76] J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design*, A. K. Peters Ltd., Wellesley, Massachusetts, 1993. translated by Larry L. Schumaker.

[77] M. K. Hubert, *Nuclear energy and the fossil fuels*, Tech. Rep. 95, Shell Development Company, Houston, Texas, June 1956.

[78] A. Iollo, M. D. Salas, and S. Ta'asan, *Shape optimization governed by the Euler equations using an adjoint method*, Tech. Rep. ICASE 93-78, NASA Langley Research Center, Hampton, Virginia, Nov. 1993.

[79] *Climate change 2007: IPCC synthesis report*, tech. rep., Valencia, Spain, Nov. 2007, `http://www.ipcc.ch/index.htm`.

[80] S. Jakobsson and O. Amoignon, *Mesh deformation using radial basis functions for gradient- based aerodynamic shape optimization*, Computers and Fluids, 36 (2007), pp. 1119–1136.

[81] A. Jameson, *Aerodynamic design via control theory*, Journal of Scientific Computing, 3 (1988), pp. 233–260.

[82] ——, *Computational Fluid Dynamics Review*, Wiley, 1995, ch. Optimum aerodynamic design using control theory, pp. 495–528.

[83] A. Jameson, L. Martinelli, and N. A. Pierce, *Optimum aerodynamic design using the Navier-Stokes equations*, Theoretical and Computational Fluid Dynamics, 10 (1998), pp. 213–237.

[84] A. Jameson and J. Reuther, *Control theory based airfoil design using Euler equations*, in AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City Beach, Sept. 1994.

[85] A. Jameson, W. Schmidt, and E. Turkel, *Numerical solution of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes*, in 14th Fluid and Plasma Dynamics Conference, Palo Alto, CA, 1981. AIAA Paper 81–1259.

[86] Z. Johan, T. J. R. Hughes, and F. Shakib, *A globally convergent matrix-free algorithm for implicit time-marching schemes arising in finite element analysis in fluids*, Computer Methods in Applied Mechanics and Engineering, 87 (1991), pp. 281–304, doi:10.1016/0045-7825(91)90009-U.

[87] A. A. Johnson and T. E. Tezduyar, *Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces*, Computer Methods in Applied Mechanics and Engineering, 119 (1994), pp. 73–94.

[88] W. T. Jones, *A grid generation system for multi-disciplinary design optimization*, in 12th AIAA Computational Fluid Dynamics Conference, no. AIAA–1995–1689, San Diego, California, United States, June 1995.

[89] D. K. Kaushik, D. E. Keyes, and B. F. Smith, *Newton-Krylov-Schwarz methods for aerodynamics problems: compressible and incompressible flows on unstructured grids*, in 11th International Conference on Domain Decomposition Methods, C.-H. Lai, P. E. Bjørstad, M. Cross, and O. Widlund, eds., 1998, pp. 513–520.

[90] C. T. Kelley, *Solving Nonlinear Equations with Newton's Method*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003.

[91] C. T. Kelley and D. E. Keyes, *Convergence analysis of pseudo-transient continuation*, SIAM Journal on Numerical Analysis, 35 (1998), pp. 508–523.

[92] D. E. Keyes, *Aerodynamic applications of Newton-Krylov-Schwarz solvers*, in Proceedings of the 14th International Conference on Numerical Methods in Fluid Dynamics, New York, 1995, Springer, pp. 1–20.

[93] D. B. Kim and P. D. Orkwis, *Jacobian update strategies for quadratic and near-quadratic convergence of Newton and Newton-like implicit schemes*, in 31st AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–93–0878, Reno, Nevada, 1993.

[94] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science, 220 (1983), pp. 671–680.

[95] D. Knoll and D. Keyes, *Jacobian-free Newton-Krylov methods: a survey of approaches and applications*, Journal of Computational Physics, 193 (2004), pp. 357–397, doi:10.1016/j.jcp.2003.08.010.

[96] H.-O. Kreiss and G. Scherer, *Finite element and finite difference methods for hyperbolic partial differential equations*, in Mathematical Aspects of Finite Elements in Partial Differential Equations, C. de Boor, ed., Mathematics Research Center, the University of Wisconsin, Academic Press, 1974.

[97] I. Kroo, *Drag due to lift: concepts for prediction and reduction*, Annual Review of Fluid Mechanics, 33 (2001), pp. 587–617.

[98] B. M. Kulfan, *A universal paramtric geometry representation method — "CST"*, in The 45th AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–2007–0062, Reno, Nevada, 2007.

[99] J. Lépine, F. Guilbault, J.-Y. Trépanier, and F. Pépin, *Optimized nonuniform rational b-spline geometrical representation for aerodynamic design of wings*, AIAA Journal, 39 (2001), pp. 2033–2041.

[100] J. Lépine, J.-Y. Trépanier, and F. Pépin, *Wing aerodynamic design using an optimized NURBS geometrical representation*, in The 38th AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–2000–699, Reno, Nevada, 2000.

[101] R. Liebeck, *Design of the blended wing body subsonic transport*, Journal of Aircraft, 41 (2004), pp. 10–25.

[102] R. H. Liebeck, *A class of airfoils designed for high lift in incompressible flow*, Journal of Aircraft, 10 (1973), pp. 610–617.

[103] X. Liu, N. Qin, and H. Xia, *Fast dynamic grid deformation based on Delaunay graph mapping*, Journal of Computational Physics, 211 (2006), pp. 405–423.

[104] H. Lomax, T. H. Pulliam, and D. W. Zingg, *Fundamentals of Computational Fluid Dynamics*, Springer–Verlag, Berlin, Germany, 2001.

[105] C. A. Mader, J. R. R. A. Martins, J. J. Alonso, and E. van der Weide, *ADjoint: an approach for rapid development of discrete adjoint solvers*, AIAA Journal, 46 (2008), pp. 863–873.

[106] J. R. R. A. Martins. personal communication, Feb. 2007.

[107] J. R. R. A. Martins, J. J. Alonso, and J. J. Reuther, *High-fidelity aerostructural design optimization of a supersonic business jet*, Journal of Aircraft, 41 (2004), pp. 863–873.

[108] J. R. R. A. Martins, P. Sturdza, and J. J. Alonso, *The complex-step derivative approximation*, ACM Transactions on Mathematical Software, 29 (2003), pp. 245–262, doi:10.1145/838250.838251.

[109] E. Maskell, *Progress towards a method for the measurement of the components of the drag of a wing of finite span*, Tech. Rep. RAE 72232, Royal Aircraft Establishment, 1973.

[110] C. W. Mastin, *Truncation error on structured grids*, in Handbook of Grid Generation, J. F. Thompson, B. K. Soni, and N. P. Weatherill, eds., CRC Press Inc., 1999, ch. 32.

[111] K. Mattsson, *Boundary procedures for summation-by-parts operators*, Journal of Scientific Computing, 18 (2003), pp. 133–153, doi:10.1023/A:1020342429644.

[112] K. Mattsson, M. Svärd, and J. Nordström, *Stable and accurate artificial dissipation*, Journal of Scientific Computing, 21 (2004), pp. 57–79.

[113] K. I. M. McKinnon, *Convergence of the Nelder–Mead simplex method to a nonstationary point*, SIAM Journal on Optimization, 9 (1998), pp. 148–158, doi:10.1137/S1052623496303482.

[114] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of Computation, 31 (1977), pp. 148–162.

[115] B. Mohammadi, *A new optimal shape design procedure for inviscid and viscous turbulent flows*, International Journal for Numerical Methods in Fluids, 25 (1997), pp. 183–203.

[116] A. Mousavi, P. Castonguay, and S. K. Nadarajah, *Survey of shape parameterization techniques and its effect on three-dimensional aerodynamic shape optimization*, in 18th AIAA Computational Fluid Dynamics Conference, no. AIAA–2007–3837, Miami, Florida, United States, June 2007.

[117] W. A. Mulder and B. van Leer, *Experiments with implicit upwind methods for the Euler equations*, Journal of Computational Physics, 59 (1985), pp. 232–246, doi:10.1016/0021-9991(85)90144-5.

[118] J. A. Nelder and R. Mead, *A simplex method for function minimization*, The Computer Journal, 7 (1965), pp. 308–313, doi:10.1093/comjnl/7.4.308.

[119] M. Nemec, *Optimal Shape Design of Aerodynamic Configurations: A Newton-Krylov Approach*, PhD thesis, University of Toronto, Toronto, Ontario, Canada, 2003.

[120] M. Nemec and M. J. Aftosmis, *Aerodynamic shape optimization using a Cartesian adjoint method and CAD geometry*, in 24th AIAA Applied Aerodynamics Conference, no. AIAA–2006–3456, San Francisco, California, United States, June 2006.

[121] M. Nemec, D. W. Zingg, and T. H. Pulliam, *Multipoint and multi-objective aerodynamic shape optimization*, AIAA Journal, 42 (2004), pp. 1057–1065.

[122] J. Nichols and D. W. Zingg, *A three-dimensional multi-block Newton-Krylov flow solver for the Euler equations*, in 17th AIAA Computational Fluid Dynamics Conference, no. AIAA–2005–5230, Toronto, Canada, June 2005.

[123] J. C. Nichols, *A three-dimensional multi-block Newton–Krylov flow solver for the Euler equations*, Master's thesis, University of Toronto, Toronto, Ontario, Canada, 2004.

[124] E. J. Nielsen and W. K. Anderson, *Recent improvements in aerodynamic design optimization on unstructured meshes*, AIAA Journal, 40 (2002), pp. 1155–1163.

[125] E. J. Nielsen and B. Kleb, *Efficient construction of discrete adjoint operators on unstructured grids by using complex variables*, in The 43rd AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–2005–0324, Reno, Nevada, 2005.

[126] E. J. Nielsen, R. W. Walters, W. K. Anderson, and D. E. Keyes, *Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code*, in 12th AIAA Computational Fluid Dynamics Conference, San Diego, CA, 1995. AIAA Paper 95–1733.

[127] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer–Verlag, Berlin, Germany, second ed., 2006.

[128] J. Nordström and M. H. Carpenter, *Boundary and interface conditions for high-order finite-difference methods applied to the Euler and Navier-Stokes equations*, Journal of Computational Physics, 148 (1999), pp. 621–645, doi:10.1006/jcph.1998.6133.

[129] ——, *High-order finite difference methods, multidimensional linear problems, and curvilinear coordinates*, Journal of Computational Physics, 173 (2001), pp. 149–174, doi:10.1006/jcph.2001.6864.

[130] P. Olsson, *Summation by parts, projections, and stability. I*, Mathematics of Computation, 64 (1995), pp. 1035–1065.

[131] ——, *Summation by parts, projections, and stability. II*, Mathematics of Computation, 64 (1995), pp. 1473–1493.

[132] P. D. Orkwis, *Comparison of Newton's and quasi-Newton's method solvers for the Navier-Stokes equations*, AIAA Journal, 31 (1993), pp. 832–836.

[133] E. Perry, R. Balling, and M. Landon, *A new morphing method for shape optimization*, in 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, no. AIAA–1998–4907, St. Louis, Missouri, Sept. 1998.

[134] L. A. Piegl and W. Tiller, *Computing the derivative of NURBS with respect to a knot*, Computer Aided Geometric Design, 15 (1998), pp. 925–934.

[135] O. Pironneau, *On optimum design in fluid mechanics*, Journal of Fluid Mechanics, 64 (1974), pp. 97–110.

[136] M. J. D. Powell, *A method for minimizing a sum of squares of non-linear functions with calculating derivatives*, The Computer Journal, 7 (1965), pp. 303–307, doi:10.1093/comjnl/7.4.303.

[137] A. Pueyo and D. W. Zingg, *Efficient Newton-Krylov solver for aerodynamic computations*, AIAA Journal, 36 (1998), pp. 1991–1997.

[138] T. Pulliam, M. Nemec, T. Holst, and D. Zingg, *Comparison of evolutionary (genetic) algorithm and adjoint methods for multi-objective viscous airfoil optimizations*, in The 41st AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–2003–0298, Jan. 2003.

[139] T. H. Pulliam, *Efficient solution methods for the Navier-Stokes equations*, tech. rep., Lecture Notes for the von Kármán Inst. for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Rhode-Saint-Genèse, Belgium, Jan. 1986.

[140] T. H. Pulliam and J. L. Steger, *Implicit finite-difference simulations of three-dimensional compressible flow*, AIAA Journal, 18 (1980), pp. 159–167.

[141] D. Quagliarella and A. D. Cioppa, *Genetic algorithms applied to the aerodynamic design of transonic airfoils*, Journal of Aircraft, 32 (1995), pp. 889–891.

[142] E. Quak and J. Gravesen, *Design for change*, SIAM News, 41 (2008).

[143] J. Reuther, A. Jameson, J. Farmer, L. Martinelli, and D. Saunders, *Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation*, in The 34rd AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–1996–0094, Reno, Nevada, 1996.

[144] J. J. Reuther, A. Jameson, J. J. Alonso, M. J. Rimlinger, and D. Saunders, *Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 1*, AIAA Journal, 36 (1999), pp. 51–60.

[145] ——, *Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 2*, AIAA Journal, 36 (1999), pp. 61–74.

[146] D. F. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics*, McGraw–Hill, Inc., New York, NY, 1990.

[147] M. Rumpfkeil, *Airfoil optimization for unsteady flows with application to high-lift noise reduction*, PhD thesis, University of Toronto, Toronto, Ontario, Canada, 2008.

[148] M. P. Rumpfkeil and D. W. Zingg, *The optimal control of unsteady flows with a discrete adjoint method*, Optimization and Engineering, (2008), p. 18, doi:10.1007/s11081-008-9035-5.

[149] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM Journal on Scientific and Statistical Computing, 14 (1993), pp. 461–469.

[150] ——, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, second ed., 2003.

[151] Y. Saad and A. V. Malevsky, *P-sparslib: a portable library of distributed memory sparse iterative solvers*, Tech. Rep. UMSI-95-180, Minnesota Supercomputer Institute, May 1995,
http://www-users.cs.umn.edu/~saad/reports.html.

[152] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[153] Y. Saad and M. Sosonkina, *Distributed Schur complement techniques for general sparse linear systems*, SIAM Journal of Scientific Computing, 21 (1999), pp. 1337–1357.

[154] J. A. Samareh, *Novel multidisciplinary shape parameterization approach*, Journal of Aircraft, 38 (2001), pp. 1015–1024.

[155] ——, *Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization*, AIAA Journal, 39 (2001), pp. 877–884.

[156] B. Saynor, A. Bauen, and M. Leach, *The potential for renewable energy sources in aviation*, tech. rep., Imperial College Centre for Energy Policy and Technology, London, England, Aug. 2003.

[157] V. Schmitt and F. Charpin, *Pressure distributions on the ONERA-M6-wing at transonic mach numbers*, tech. rep., Office National d'Etudes et Recherches Aerospatiales, 92320, Chatillon, France, 1979.

[158] V. Schramm, J. Denecke, S. Kim, and S. Wittig, *Shape optimization of a labyrinth seal applying the simulated annealing method*, Internation Journal of Rotating Machinery, 10 (2004), pp. 365–371, doi:10.1080/10236210490474485.

[159] G. Schrauf, *Status and perspectives of laminar flow*, The Aeronautical Journal, (2005), pp. 639–644.

[160] L. L. Schumaker, *Spline functions*, John Wiley & Sons, Inc., New York, NY, 1981.

[161] T. W. Sederberg and S. R. Parry, *Free-form deformation of solid geometric models*, in SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, New York, NY, USA, 1986, ACM, pp. 151–160, doi:10.1145/15922.15903.

[162] C.-W. Shu, *High-order finite difference and finite volume WENO schemes and discontinuous Galerkin methods for CFD*, International Journal of Computational Fluid Dynamics, 17 (2003), pp. 107–118.

[163] R. Slingerland, *Aircraft with LH2 propulsion*, in Innovative Configurations and Advanced Concepts for Future Civil Aircraft, E. Torenbeek and H. Deconinck, eds., von Kármán Inst. Lecture Series, Bressels, Belgium, June 2005.

[164] S. C. Smith, *A computational and experimental study of nonlinear aspects of induced drag*, Tech. Rep. NASA TP 3598, National Aeronautics and Space Administration, Ames Research Center, Moffett Field, CA, 94035–1000, 1996.

[165] S. C. Smith and I. M. Kroo, *Computation of induced drag for elliptical and crescent-shaped wings*, Journal of Aircraft, 30 (1993), pp. 446–452.

[166] W. Squire and G. Trapp, *Using complex variables to estimate derivatives of real functions*, SIAM Review, 40 (1998), pp. 110–112.

[167] B. Strand, *Summation by parts for finite difference approximations for d/dx*, Journal of Computational Physics, 110 (1994), pp. 47–67, doi:10.1006/jcph.1994.1005.

[168] P. Sturdza, *Extensive supersonic natural laminar flow on the Aerion business jet*, in The 45th AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–2007–685, Reno, Nevada, 2007.

[169] M. Svärd, *Stable High-Order Finite Difference Methods for Aerodynamics*, PhD thesis, Uppsala University, Uppsala, Sweden, 2004.

[170] R. C. Swanson and E. Turkel, *On central-difference and upwind schemes*, Journal of Computational Physics, 101 (1992), pp. 292–306, doi:10.1016/0021-9991(92)90007-L.

[171] S. Ta'asan, G. Kuruvila, and M. D. Salas, *Aerodynamic design and optimization in one shot*, in The 30th AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–1992–25, Jan. 1992.

[172] M. Tabesh and D. W. Zingg, *Efficient implicit time-marching methods using a Newton-Krylov algorithm*, in 47th AIAA Aerospace Science Meeting and Exhibit, no. AIAA–2009–0164, Orlando, Florida, Jan. 2009.

[173] S. Takahashi, S. Obayashi, and K. Nakahashi, *Inverse design optimization of transonic wings based on multi-objective genetic algorithms*, AIAA Journal, 37 (1999), pp. 1656–1662.

[174] P. D. Thomas and C. K. Lombard, *Geometric conservation law and its application to flow computations on moving grids*, AIAA Journal, 17 (1979), pp. 1030–1037.

[175] A. H. Truong, C. A. Oldfield, and D. W. Zingg, *Mesh movement for a discrete-adjoint Newton-Krylov algorithm for aerodynamic optimization*, AIAA Journal, 46 (2008), pp. 1695–1704.

[176] M. J. Vachon, R. J. Ray, K. R. Walsh, and K. Ennix, *F/a-18 performance benefits measured during the autonomous formation flight project*, Tech. Rep. NASA/TM-2003-210734, NASA Dryden Flight Research Center, Edwards, California, Sept. 2003.

[177] A. Vicini and D. Quagliarella, *Inverse and direct airfoil design using a multiobjective genetic algorithm*, AIAA Journal, 35 (1997), pp. 1499–1505.

[178] T. von Kármán and J. M. Burgers, *General aerodynamic theory — perfect fluids*, in Aerodynamic Theory: A General Review of Progress, vol. 2, Julius Springer, Berlin, Germany, 1935.

[179] A. Vretblad, *Fourier analysis and its applications*, Springer–Verlag, New York, NY, 2003.

[180] A. Wächter and L. T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Mathematical Programming, 106 (2006), pp. 25–57, doi:10.1007/s10107-004-0559-y.

[181] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban, *An interior algorithm for nonlinear optimization that combines line search and trust region steps*, Mathematical Programming, 107 (2006), pp. 391–408, doi:10.1007/s10107-004-0560-5.

[182] C. Walz, *A unified approach to B-spline recursions and knot insertion, with application to new recursion formulas*, Advances in Computational Mathematics, 3 (1995), pp. 89–100.

[183] S. R. Weart, *The Discovery of Global Warming*, Harvard University Press, Cambridge, Massachusetts, 2003.

[184] Z. Yang and D. J. Mavriplis, *Unstructured dynamic meshes with higher-order time integration schemes for the unsteady Navier-Stokes equations*, in 43rd AIAA Aerospace Sciences Meeting and Exhibit, no. AIAA–2005–1222, Reno, Nevada, 2005.

[185] T.-Y. Yu and B. K. Soni, *Application of NURBS in numerical grid generation*, Computer-Aided Design, 27 (1995), pp. 147–157.

[186] T.-Y. Yu and B. K. Soni, *NURBS in structured grid generation*, in Handbook of grid generation, J. F. Thompson, B. K. Soni, and N. P. Weatherill, eds., CRC Press Inc., 1999, ch. 30.

[187] D. Zeng and C. R. Ethier, *A semi-torsional spring analogy model for updating unstructured meshes in 3D moving domains*, Finite Elements in Analysis and Design, 41 (2005), pp. 1118–1139, doi:10.1016/j.finel.2005.01.003.

[188] D. W. Zingg and T. T. Chisholm, *Jacobian-free Newton-Krylov methods: issues and solutions*, in Proceedings of The Fourth International Conference on Computational Fluid Dynamics, Ghent, Belgium, July 2006.

Appendices

Appendix A

Flow Solver

A.1 Numerical Dissipation

The scalar dissipation model is based on a symmetric positive definite (SPD) operator. In an unsteady simulation, such an operator will drain energy from $\mathbf{q}$, ensuring stability. The actual dissipation operator is nonlinear, so an unequivocal statement regarding its stability properties is not possible; nevertheless, the model has been used successfully by many authors.

I describe the one-dimensional dissipation operator in the direction ξ for one variable only. The operator is extended to 3-dimensions and multiple variables using Kronecker products. Suppose there are L nodes in the ξ directions. Define the forward difference operator $\Delta_\xi \in \mathbb{R}^{(L-1)\times L}$ and diagonal matrix $B \in \mathbb{R}^{L\times L}$ as follows.

$$\Delta_\xi = \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & -1 & 1 & \\ & & & \ddots & \ddots \\ & & & & -1 \quad 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ & & & & & 0 \end{bmatrix}.$$

The rectangular matrix Δ_ξ forms undivided differences of the nodal variables at the half-node positions, i.e. the positions geometrically centered between nodes in computational space. In addition, we need the nonlinear diagonal matrix operator $C_\xi^{(4)}(\mathbf{q}) \in \mathbb{R}^{(L-1)\times(L-1)}$;

$$C_\xi^{(4)}(\mathbf{q}) = \begin{bmatrix} \kappa_4 \sigma^{(\xi)}_{\frac{1}{2}} & & & \\ & \kappa_4 \sigma^{(\xi)}_{\frac{3}{2}} & & \\ & & \ddots & \\ & & & \kappa_4 \sigma^{(\xi)}_{L-\frac{3}{2}} \end{bmatrix},$$

where $\sigma^{(\xi)}_{i+\frac{1}{2}} = \frac{1}{2}\left(\sigma^{(\xi)}_i + \sigma^{(\xi)}_{i+1}\right)$ and

$$\sigma^{(\xi)}_i = \left[\frac{|U_1| + a\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}}{J}\right]_i$$

is the spectral radius of the flux Jacobian $\hat{A}_1 = \partial\hat{\mathbf{F}}_1/\partial\hat{\mathbf{Q}}$, evaluated at node i and scaled by the metric Jacobian, J.

Using the above operators, we can define the fourth-difference dissipation operator $Z_\xi \in \mathbb{M}^{L\times L}$.

$$Z_\xi = \Delta_\xi^T C_\xi^{(4)}(\mathbf{q})\Delta_\xi B \Delta_\xi^T \Delta_\xi.$$

For constant $\sigma^{(\xi)}$, it is easily verified that the above operator is SPD. In addition, the operator is third-order accurate in the interior, and first-order accurate at the boundaries.

For transonic flows, the dissipation operator must be modified to eliminate oscillations near shocks. This is accomplished by adding second-difference dissipation and modifying the fourth-difference coefficient κ_4. In particular, Z_ξ becomes

$$Z_\xi = \Delta_\xi^T C_\xi^{(2)}(\mathbf{q})\Delta_\xi + \Delta_\xi^T C_\xi^{(4)}(\mathbf{q})\Delta_\xi B \Delta_\xi^T \Delta_\xi,$$

where the diagonal matrix $C_\xi^{(2)}(\mathbf{q}) \in \mathbb{R}^{(L-1)\times(L-1)}$ is defined by

$$C_\xi^{(2)}(\mathbf{q}) = \begin{bmatrix} \kappa_2\sigma^{(\xi)}_{\frac{1}{2}}\varepsilon^{(\xi)}_{\frac{1}{2}} & & & \\ & \kappa_2\sigma^{(\xi)}_{\frac{3}{2}}\varepsilon^{(\xi)}_{\frac{3}{2}} & & \\ & & \ddots & \\ & & & \kappa_2\sigma^{(\xi)}_{L-\frac{3}{2}}\varepsilon^{(\xi)}_{L-\frac{3}{2}} \end{bmatrix},$$

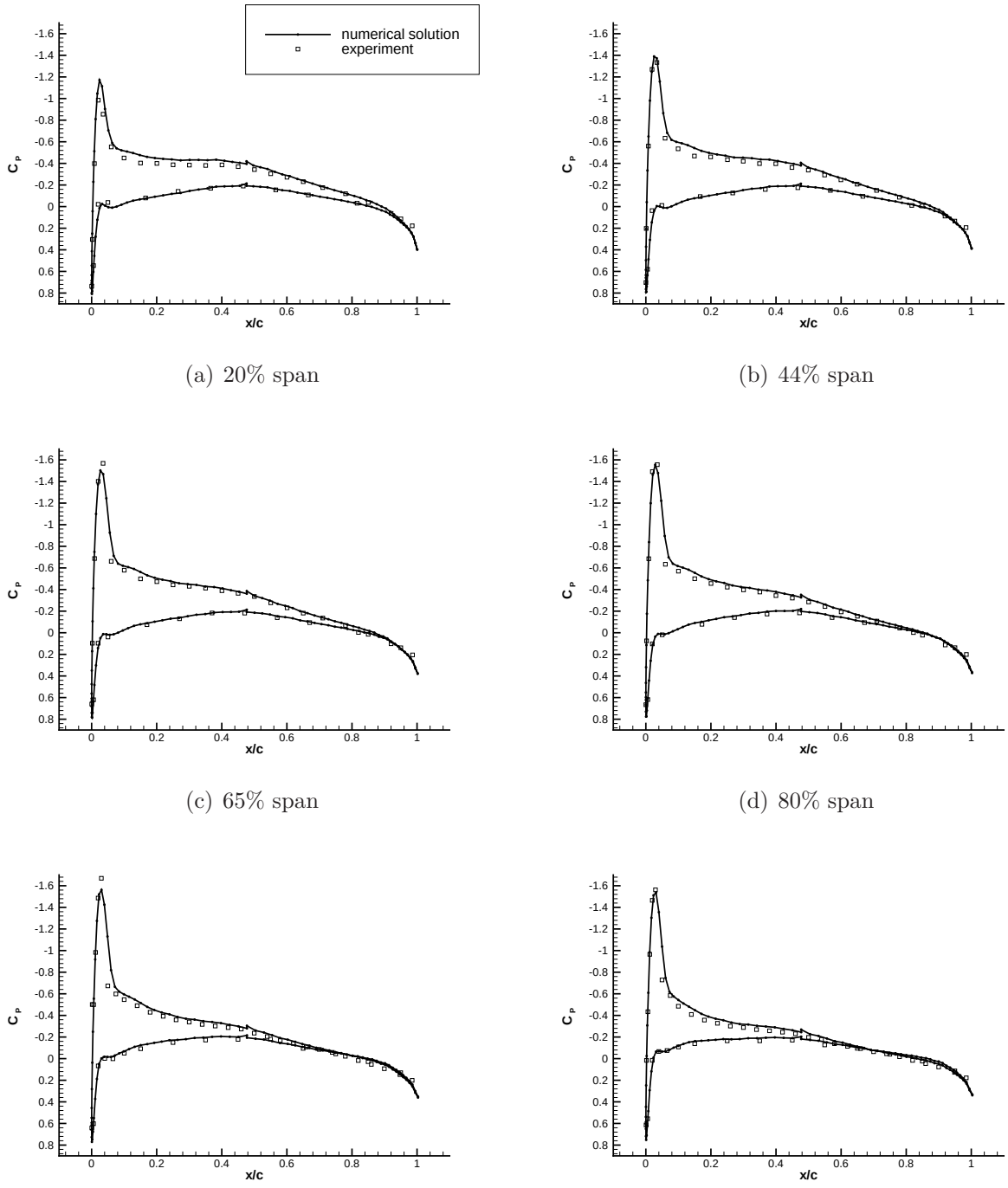
and $\varepsilon^{(\xi)}_{i+\frac{1}{2}} = \varepsilon^{(\xi)}_i + \varepsilon^{(\xi)}_{i+1}$ is a pressure switch that activates near shocks. It is defined at the nodes by the following equations.

$$\begin{aligned} \varepsilon^{(\xi)}_i &= \frac{1}{4}\left(\Upsilon^\star_{i-1} + 2\Upsilon^\star_i + \Upsilon^\star_{i+1}\right), \\ \Upsilon^\star_i &= \max\left(\Upsilon_{i-1}, \Upsilon_i, \Upsilon_{i+1}\right), \\ \Upsilon_i &= \frac{|p_{i-1} - 2p_i + p_{i+1}|}{|p_{i-1} + 2p_i + p_{i+1}|}. \end{aligned}$$

Values for $\varepsilon^{(\xi)}_i$, $\Upsilon^\star_i$ and Υ_i at block sides are extrapolated from the adjacent interior nodes. Finally, as mentioned above, the parameter κ_4 is modified; it is replaced in $C_\xi^{(4)}$ with

$$\hat{\kappa}_{4,i+\frac{1}{2}} = \max\left(0, \kappa_4 - \kappa_2\varepsilon^{(\xi)}_{i+\frac{1}{2}}\right).$$

A.2 ONERA M6: Pressure Distributions at Mach 0.699



(a) 20% span

(b) 44% span

(c) 65% span

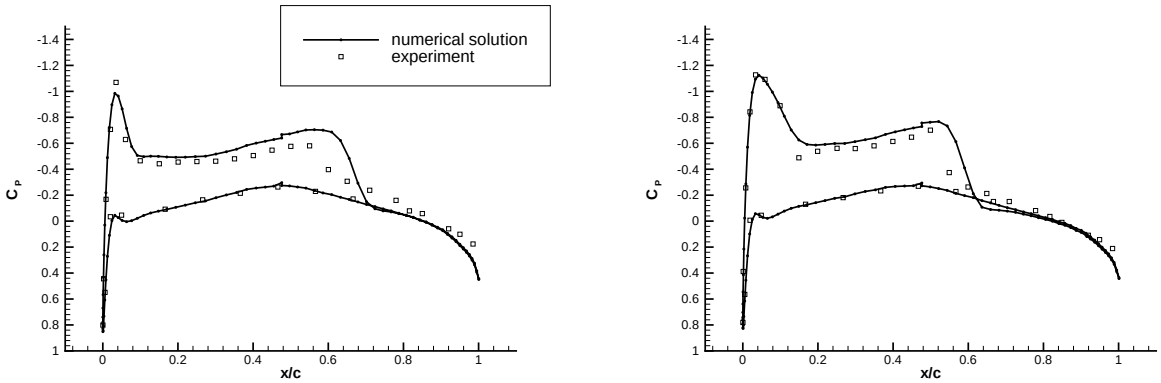
(d) 80% span

(e) 90% span

(f) 96% span

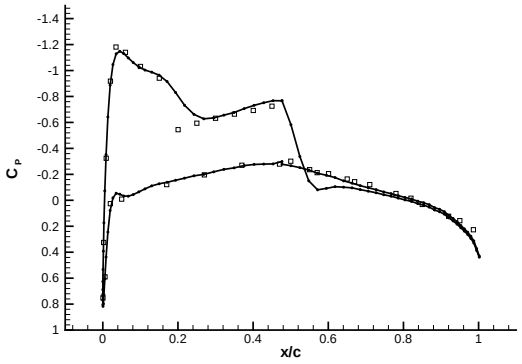
Figure A.1: Comparison of the coefficient of pressure at a Mach number of 0.699 around the ONERA M6 wing. The grid has 96 blocks and 1.168×10^6 nodes.

A.3 ONERA M6: Pressure Distributions at Mach 0.84

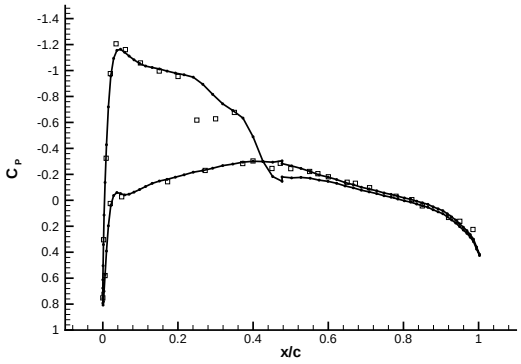


(a) 20% span

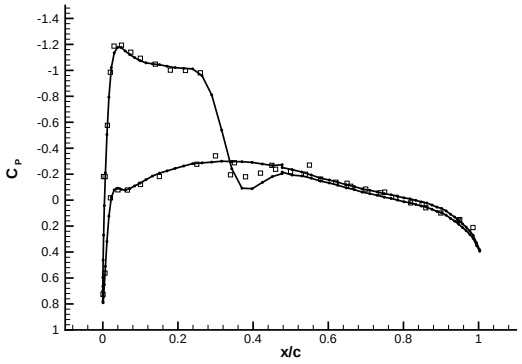
(b) 44% span



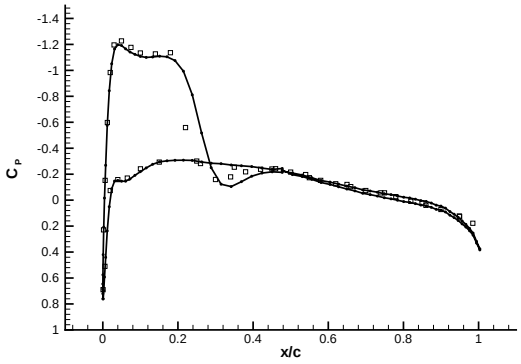
(c) 65% span



(d) 80% span



(e) 90% span



(f) 96% span

Figure A.2: Comparison of the coefficient of pressure at a Mach number of 0.84 around the ONERA M6 wing. The grid has 96 blocks and 1.168×10^6 nodes.

A.4 Approximate-Schur Preconditioner

Algorithm 1: Approximate-Schur Preconditioner

Data: m, η, $\mathbf{u}$
Result: $\mathbf{v}$

$\mathbf{v} = U_{S_i}^{-1} L_i^{-1} \mathbf{u}$ `// get residual of reduced system for a guess of zero`
$\mathbf{w}_{(i)}^{(1)} = P\mathbf{v}$ `// restrict v to internal-interface unknowns using` P
set $\beta = \|\mathbf{w}_{(i)}^{(1)}\|_2$, $\mathbf{w}_{(i)}^{(1)} \leftarrow \mathbf{w}_{(i)}^{(1)}/\beta$, and $H = 0$
for $j = 1, m$ **do**
    obtain external interface values, $\mathbf{w}_{(i,\text{ext})}^{(j)}$
    $\mathbf{w}_{(i)}^{(j+1)} = E_i \mathbf{w}_{(i,\text{ext})}^{(j)}$ `// perform external matrix-vector product`
    $\mathbf{w}_{(i)}^{(j+1)} \leftarrow U_{S_i}^{-1} L_{S_i}^{-1} \mathbf{w}_{(i)}^{(j+1)}$ `// apply diagonal block of Schur complement`
    $\mathbf{w}_{(i)}^{(j+1)} \leftarrow \mathbf{w}_{(i)}^{(j)} + \mathbf{w}_{(i)}^{(j+1)}$ `// finish the matrix-vector product` (3.33)
    for $k = 1, j$ **do** Gram-Schmidt orthogonalization
        $H_{k,j} = (\mathbf{w}_{(i)}^{(j+1)})^T \mathbf{w}_{(i)}^{(k)}$
        $\mathbf{w}_{(i)}^{(j+1)} \leftarrow \mathbf{w}_{(i)}^{(j+1)} - H_{k,j} \mathbf{w}_{(i)}^{(k)}$
    end
    $H_{j+1,j} = \|\mathbf{w}_{(i)}^{(j+1)}\|_2$
    $\mathbf{w}_{(i)}^{(j+1)} \leftarrow \mathbf{w}_{(i)}^{(j+1)}/H_{j+1,j}$
    if *reduced system residual tolerance* $\leq \eta$ **then**
        set $m = j$ and exit
    end
end
Define $W_m = [\mathbf{w}_{(i)}^{(1)}, \ldots, \mathbf{w}_{(i)}^{(m)}]$
Compute $\mathbf{y}_{(i)} = W_m \mathbf{z}^{(m)}$ where $\mathbf{z}^{(m)} = \min_z \|\beta e_1 - Hz\|_2$ and $e_1 = [1, 0, \ldots, 0]^T$
obtain external interface preconditioned values, $\mathbf{y}_{(i,\text{ext})}$
$\mathbf{v} \leftarrow \mathbf{v} + P^T P(\mathbf{u} - E_i \mathbf{y}_{(i,\text{ext})} - \mathbf{v})$ `// updates internal-interface only`
$\mathbf{v} \leftarrow U_i^{-1} \left[I + P^T P(L_i^{-1} - I) \right] \mathbf{v}$ `// apply forward solve to interface, then backsolve`

Appendix B

Gradient Evaluation

B.1 Optimal Adjoint Tolerance

In this section, I investigate the appropriate tolerance to use for the 3-dimensional Euler-adjoint equations. Consider an unswept generic wing geometry with symmetric cross sections. The design variables consist of the coordinates of the 79 B-spline control points defining the upper and lower surfaces of the wing, as well as the angle of attack. The adjoint equation for lift-over-drag ratio was solved using tolerances from 10^{-12} to 10^{-5}. The gradient was evaluated, and the RMS gradient error was calculated based on an adjoint solution with a tolerance of 10^{-13} as the 'exact' solution. To prevent confounding effects, the mesh adjoint was solved to a tolerance of 10^{-14}.

Figure B.1 plots the RMS gradient error versus the adjoint-system tolerance. For example, to achieve a gradient accuracy of 10^{-6}, the adjoint tolerance should be approximately 10^{-8}. Although I fix the tolerance for the cases described in this thesis, one could consider decreasing the tolerance adaptively based on the previous norm of the gradient.

B.2 Flexible Variant of GCROT

GCROT is a truncated version of the Krylov iterative solver GCRO (generalized conjugate residual with inner orthogonalization) proposed by de Sturler [29]. GCRO uses an inner-outer approach. On the k^{th} outer iteration, the method produces a residual correction, denoted c_k, which is appended to the matrix C_k containing the k previous residual corrections. In the inner iterations a standard Krylov method, such as GMRES, is used; however, instead of solving $Ax = b$, the inner method makes use of the vectors from previous outer iterations and attempts to solve $(I - C_k C_k^T)Ax = b$. When GMRES is adopted as the inner method, GCRO finds the minimal residual solution over the range of the outer and inner vectors, and

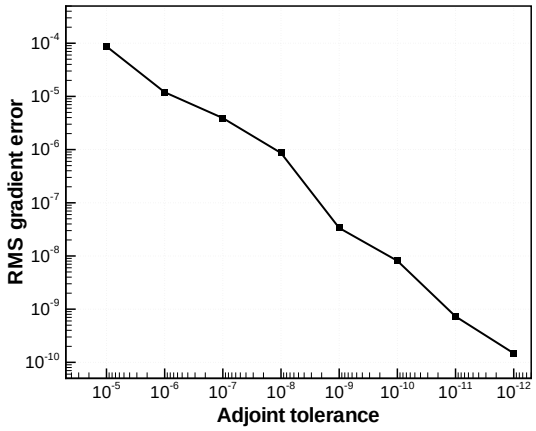


Figure B.1: Gradient accuracy versus adjoint tolerance.

thereby helps avoid stagnation.

The number of vectors in C_k grows linearly with each outer iteration, so GCRO, like GMRES, may need to be truncated. To address this issue, de Sturler developed GCRO truncated, or GCROT [30]. In this method, the subspaces that contributed the most to the residual reduction can be determined and retained, while those that were not useful can be discarded.

The original versions of GCRO and GCROT are not flexible; hence, they can not accommodate the approximate-Schur preconditioner. This shortcoming is significant, since numerical experiments indicate that the Schur preconditioner is more efficient than the Schwarz preconditioner for the adjoint problem. With minor changes, GCRO is easily adapted to accept FGMRES as the inner method, thus producing a flexible version. A general, flexible version of GCROT is more difficult to obtain, due to some results that apply to GMRES but not FGMRES. As a compromise, the proposed variant of GCROT is flexible, but not as general as the one presented by de Sturler.

Algorithm 2: GCROT(m,δ)

Data: $\mathbf{x}_0$, m, δ
Result: $\mathbf{x}$

set $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, and $k = 0$
for $l = 0, 1, 2, \ldots$ **do**
  compute $\beta = \|\mathbf{r}_l\|_2$ and $\mathbf{v}_1 = \mathbf{r}_l/\beta$ `// begin inner FGMRES method`
  for $j = 1, m$ **do**
    compute $\mathbf{z}_j = M_j^{-1}\mathbf{v}_j$ `// M denotes the preconditioner`
    compute $\mathbf{w} = A\mathbf{z}_j$
    for $i = 1, k$ **do** `// orthogonalize w against` C_k
      $b_{i,j} = \mathbf{w}^T\mathbf{c}_i$
      $\mathbf{w} := \mathbf{w} - b_{i,j}\mathbf{c}_i$
    end
    for $i = 1, j$ **do** `// orthogonalize w against` $V_{1:j}$
      $h_{i,j} = \mathbf{w}^T\mathbf{v}_i$
      $\mathbf{w} := \mathbf{w} - h_{ij}\mathbf{v}_i$
    end
    compute $h_{j+1,j} = \|\mathbf{w}\|_2$ and $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$
  end
  define $Z_m := [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m]$
  compute $\mathbf{y}_m = \min_{\mathbf{y}} \|\beta\mathbf{e}_1 - \bar{H}_m\mathbf{y}\|$
  $\hat{\mathbf{u}} = (Z_m - U_k B_m)\mathbf{y}_m$ `// calculate new outer vectors,` $\hat{\mathbf{u}}$ `and` $\hat{\mathbf{c}}$
  $\hat{\mathbf{c}} = \mathbf{r}_l - \mathbf{r}_{\text{inner}} = V_{m+1}\bar{H}_m\mathbf{y}_m$
  compute $\alpha = \|\hat{\mathbf{c}}\|$, $\hat{\mathbf{c}} := \hat{\mathbf{c}}/\alpha$, and $\hat{\mathbf{u}} := \hat{\mathbf{u}}/\alpha$
  $\mathbf{r}_{l+1} := \mathbf{r}_l - (\hat{\mathbf{c}}^T\mathbf{r}_l)\hat{\mathbf{c}}$ `// update residual and solution`
  $\mathbf{x}_{l+1} := \mathbf{x}_l + (\hat{\mathbf{c}}^T\mathbf{r}_l)\hat{\mathbf{c}}$
  $N = 0$
  if $(k > 0)$ **then** `// find the number,` N`, of vectors to discard`
    compute $D = B_m R_m^{-1}$
    find sing. value decomp. $D = \Lambda_D \Sigma_D \Gamma_D^T$, where $\Sigma_D = \text{diag}\,(\sigma_i)$
    for $i = 1, k$ **do**
      if $\sigma_i/\sqrt{1+\sigma_i^2} < \delta$ **then** $N := N + 1$
    end
    if $k \geq m$ **then** $N = \max(1, N)$ `// prevent` k `from growing too big`
  end
  for $i = 1, N$ **do**
    discard subspace associated with σ_i using Givens rotation; see ref. [30]
  end
  set $k := k - N + 1$ and $m := m + N - 1$
  set $C_k := \begin{bmatrix} C_{k-1} & \hat{\mathbf{c}} \end{bmatrix}$ and $U_k := \begin{bmatrix} U_{k-1} & \hat{\mathbf{u}} \end{bmatrix}$
end

Appendix C

Ancillary Optimization Data and Results

C.1 Perturbation of the Lift Distribution and its Effect on Induced Drag

Consider a planar wing with an elliptical lift distribution, defined by its circulation $\Gamma(y)$, where y is the spanwise coordinate. Using the transformation $y = -\frac{b}{2}\cos(\theta)$, where $0 \leq \theta \leq \pi$ and b is the span, the elliptical distribution can be expressed as (see, for example, reference [4])

$$\Gamma_{\text{ellip}}(\theta) = 2bVA_1 \sin(\theta).$$

V denotes the free-stream velocity magnitude. The lift coefficient is determined by the coefficient A_1; specifically, $C_L = A_1 \pi \frac{b^2}{S}$, where S is the reference area.

Next, consider an arbitrary C^2 perturbation of the elliptical lift distribution. This smoothness assumption is reasonable for a subsonic steady flow and a smooth geometry. Using a Fourier sine series, the perturbed lift distribution can be written as

$$\Gamma(\theta) = 2bVA_1 \sin(\theta) + \epsilon \left[2bV \sum_{n=2}^{\infty} A_n \sin(n\theta) \right],$$

where ϵ controls the magnitude of the perturbation. The elliptical and perturbed distributions produce the same lift, since the leading coefficient A_1 is the same. In addition, it is easy to show that the L^2 norm of the perturbation is bounded by ϵ:

$$\|\Gamma - \Gamma_{\text{ellip}}\|_2 = \mathrm{O}(\epsilon). \tag{C.1}$$

We are interested in the relationship between ϵ and the induced drag of the perturbed lift distribution. For the subsequent analysis, we need the following bound on the coefficients

A_n, which is a consequence of $\Gamma \in C^2$ and Fourier theory (see, for example, theorem 4.4 from Vretblad[179]):

$$|A_n| \leq \frac{c}{n^2} \tag{C.2}$$

for some constant $c \in \mathbb{R}$. Now, the induced drag for the elliptical lift distribution is simply

$$C_{D,i,\text{ellip}} = \frac{\pi b^2 A_1^2}{S},$$

and for the perturbed distribution we have[4],

$$\begin{aligned}
C_{D,i} &= \left(\frac{\pi b^2 A_1^2}{S}\right)\left[1 + \epsilon^2 \sum_{n=2}^{\infty} n \left(\frac{A_n}{A_1}\right)^2\right] \\
&\leq \left(\frac{\pi b^2 A_1^2}{S}\right)\left[1 + \left(\frac{\epsilon c}{A_1}\right)^2 \sum_{n=2}^{\infty} \frac{1}{n^3}\right] && \text{(using inequality (C.2))} \\
&\leq \left(\frac{\pi b^2 A_1^2}{S}\right)\left[1 + \left(\frac{\epsilon c}{A_1}\right)^2 \left(\frac{\pi^2}{6} - 1\right)\right]. && \text{(sum of bounding } p\text{-series, } p = 2)
\end{aligned}$$

Thus, we have shown that the difference between the induced drags of the perturbed and elliptical distributions has the following asymptotic behaviour:

$$|C_{D,i} - C_{D,i,\text{ellip}}| = \mathrm{O}(\epsilon^2). \tag{C.3}$$

On the one hand, equations (C.1) and (C.3) suggest that a fairly large perturbation of the elliptical lift distribution will have a small effect on the induced drag. On the other hand, these equations underscore the difficulty of recovering the elliptical lift distribution through optimization: if we want to obtain a lift distribution that is within ϵ of the elliptical distribution, the induced drag must be accurate to within ϵ^2.

This effect is illustrated in Figure C.1. An elliptical circulation distribution, $\Gamma_{\text{ellip}} = \sin(\theta)$, is given a perturbation of the form $\epsilon \sin(3\theta)$ where $\epsilon = 0.075$. Despite this relatively large perturbation in the lift distribution, the resulting change in the drag is only 1.69%.

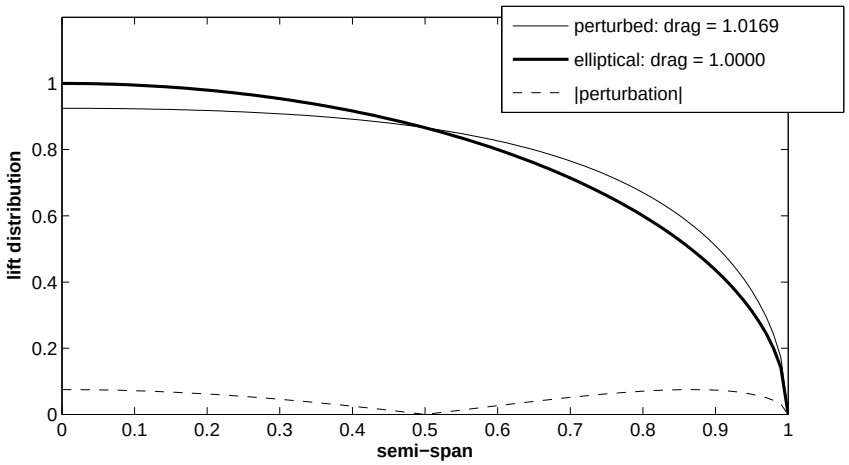
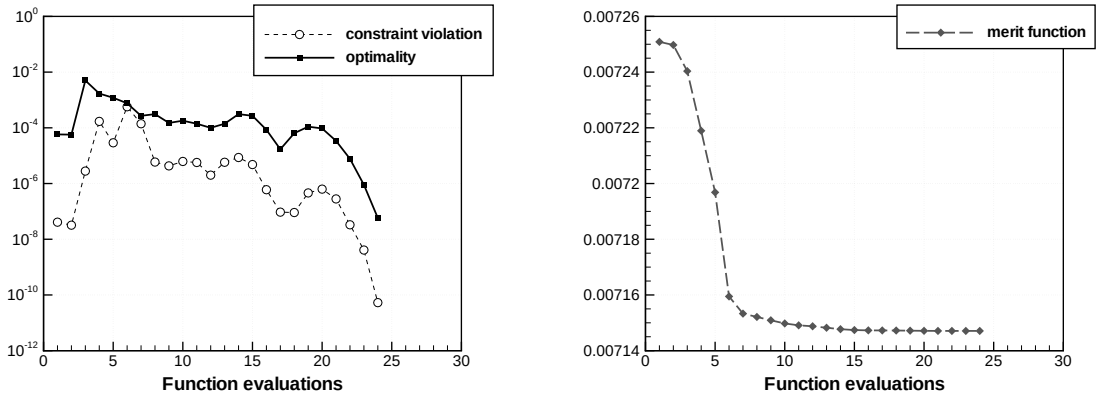


Figure C.1: Example illustrating the effect on drag when the lift distribution is perturbed. A relatively large change in the lift distribution has a small effect on the drag.

C.2 Optimization Convergence Histories

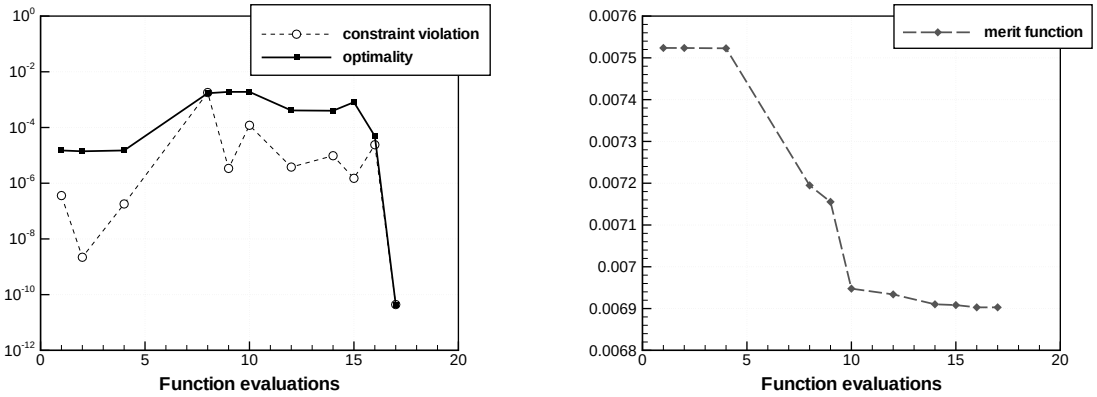
C.2.1 Planform Optimization



(a) optimality and constraint violation (b) merit function

Figure C.2: Convergence history for the planform shape optimization. Note that the merit function becomes the objective (drag coefficient) as the constraint violation goes to zero.

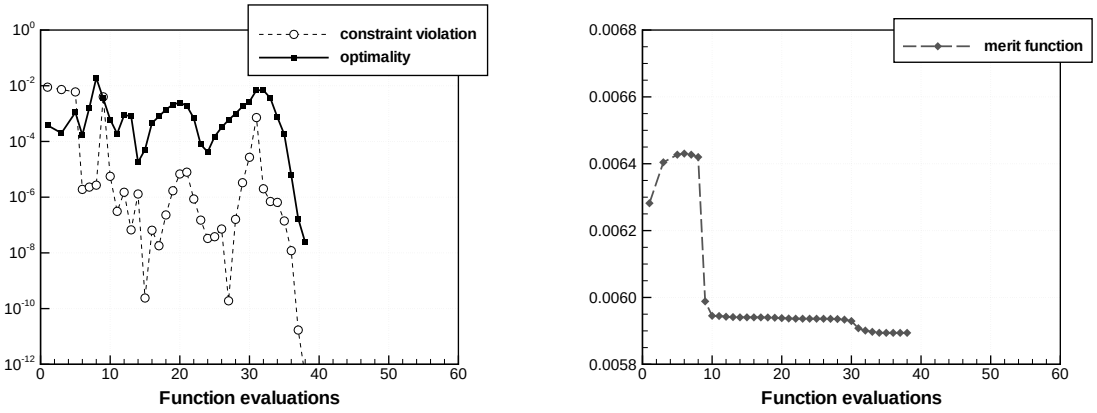
C.2.2 Spanwise Vertical Shape Optimization



(a) optimality and constraint violation (b) merit function

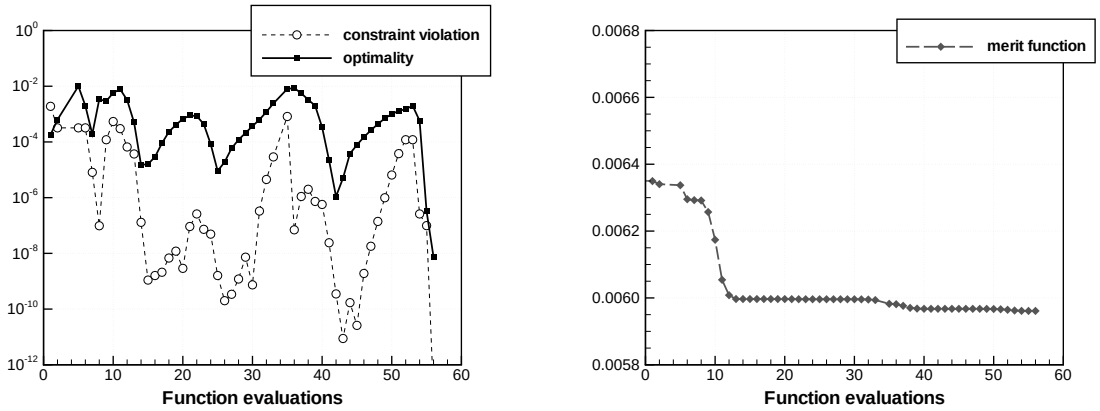
Figure C.3: Convergence history for the spanwise vertical shape optimization. Note that the merit function becomes the objective (drag coefficient) as the constraint violation goes to zero.

C.2.3 Split-tip Optimization



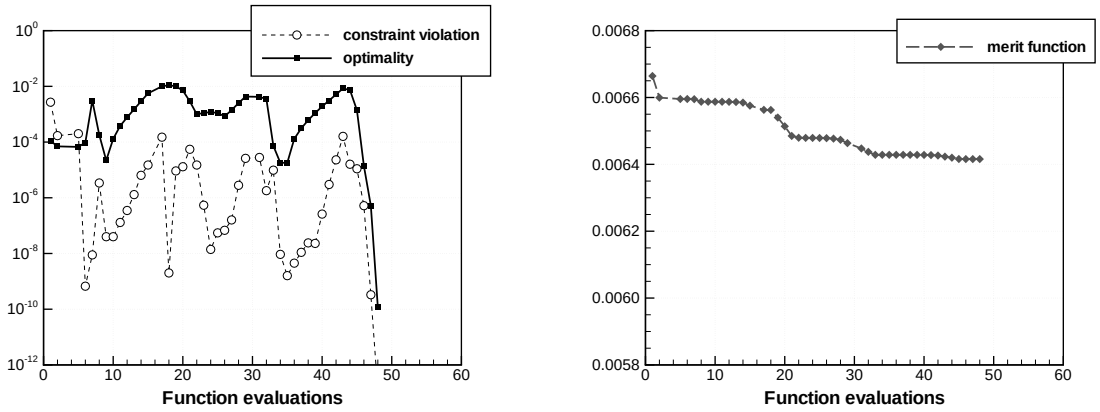
(a) optimality and constraint violation (b) merit function

Figure C.4: Convergence history for the split-tip up-down shape optimization. Note that the merit function becomes the objective (drag coefficient) as the constraint violation goes to zero.



(a) optimality and constraint violation (b) merit function

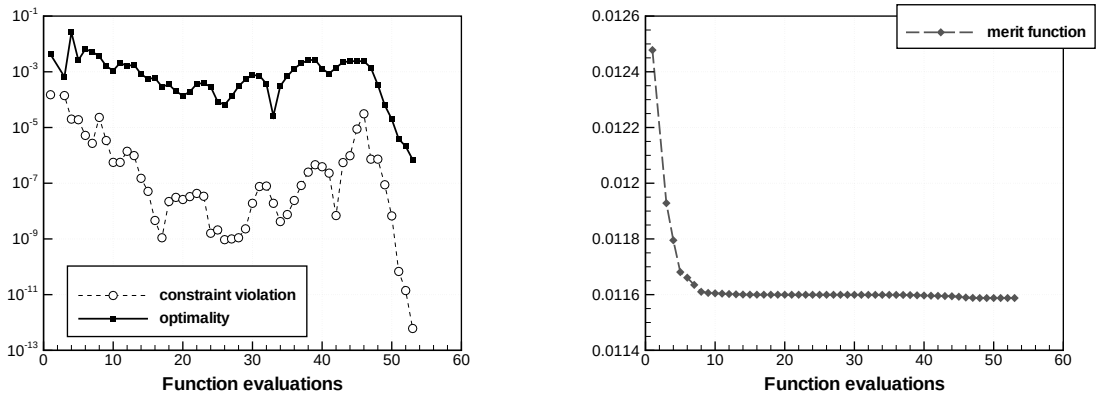
Figure C.5: Convergence history for the split-tip down-up shape optimization. Note that the merit function becomes the objective (drag coefficient) as the constraint violation goes to zero.



(a) optimality and constraint violation (b) merit function

Figure C.6: Convergence history for the single-tip shape optimization. Note that the merit function becomes the objective (drag coefficient) as the constraint violation goes to zero.

C.2.4 Box-Wing Optimization



(a) optimality and constraint violation

(b) merit function

Figure C.7: Convergence history for the box-wing configuration shape optimization. Note that the merit function becomes the objective (drag coefficient) as the constraint violation goes to zero.

C.3 Default Parameter Values

Table C.1: List of default parameters used in the induced-drag studies of Chapter 5.

parameter		value
mesh movement		
ν	Poisson's ratio	0.33333
p	ILU fill level	1
—	CG tolerance	10^{-12}
m	number of increments	5
flow solver		
κ_2	dissipation coefficient	0.0
κ_4	dissipation coefficient	0.03
σ	dissipation lumping factor	5.0
η	Schur tolerance	0.01
m	Schur maximum iterations	5
—	Block-RCM reordering	True
p	ILU fill level	1
a	Δt_{ref} parameter	0.1
b	Δt_{ref} parameter	1.7
β	Δt_{ref} parameter	2.0
m	A_1 update period	4
η	GMRES tolerance (approx.-Newton)	0.5
η	GMRES tolerance (inexact-Newton)	0.01
—	maximum GMRES iterations	80
τ	tolerance to start inexact-Newton	0.1
—	relative residual norm tolerance	10^{-10}
—	absolute residual norm tolerance	10^{-12}
optimization		
p	flow adjoint ILU fill level	2
—	flow adjoint tolerance	10^{-8}
p	mesh adjoint ILU fill level	1
—	mesh adjoint tolerance	10^{-12}
ε	optimality tolerance	10^{-7}