An investigation of induced drag minimization using a Newton-Krylov algorithm

Jason E. Hicken[*] and David W. Zingg [†]

Institute for Aerospace Studies, University of Toronto, Toronto, Ontario, M3H 5T6, Canada

We present an optimization algorithm for the study of induced drag minimization, with applications to unconventional aircraft design. The algorithm is based on a discrete-adjoint formulation and uses an efficient parallel-Newton-Krylov solution strategy. We validate the optimizer by recovering an elliptical lift distribution using twist optimization; we believe this an important, and under-appreciated, benchmark for aerodynamic optimization. The algorithm is further illustrated using several design examples, including planform, spanwise vertical shape, and box-wing optimization.

I. Introduction

The aircraft industry faces two challenges in the twenty-first century: climate change and peak oil production. These problems may eventually be solved by alternative fuels such as hydrogen, bio-kerosene, and coal-based kerosene (provided that carbon from the coal is sequestered); however, when production emissions are included, these alternative fuels presently produce more greenhouse gases than traditional kerosene.[1] Alternative fuels must, therefore, be considered a long term solution.

In the near term, reduced emissions and improved fuel efficiency may be possible by considering unconventional configurations, such as the often cited blended-wing-body; see, for example, Liebeck.[2] The evolution of the blended-wing-body concept and advent of aerodynamic shape optimization motivate the following question: can we use numerical optimization to "discover" radically new concepts in aircraft design? In this paper we present an algorithm which we hope can begin addressing this question.

Our preliminary goal is to develop an Euler-based optimization code capable of minimizing induced drag. While viscous and turbulence effects cannot be neglected in practical applications, it is important that the optimizer accurately minimize drag due to lift. This requirement can be motivated by considering a typical commercial aircraft, where induced drag represents 40% of the total drag.[3]

In addition to presenting the optimization algorithm, we hope to shed some light on a useful, but overlooked, benchmark for aerodynamic optimization. An elliptical lift distribution is known to minimize induced drag for planar configurations at low Mach numbers. Recovering an elliptical distribution as a (local) optimum should be a reasonable validation for any aerodynamic optimization algorithm; nevertheless, we have found no evidence in the literature that such a benchmark has ever been used explicitly for Euler-based optimizations.

Broadly, the paper is divided into a description of the algorithm and its illustration with examples. The algorithm description, Sections II–IV, follows the sequence of operations as they would be encountered in a typical design iteration. In particular, Section II describes the design space parametrization and integrated mesh movement algorithm. Subsequently, Section III reviews the Euler flow solver used for analysis. Gradient evaluation is covered in Section IV, including details regarding the flow and mesh adjoint equations. The second half of the paper includes verification and validation, presented in Section V, and design examples, provided in Section VI. A summary and our conclusions are given in Section VII

[*]PhD Candidate, AIAA Student Member
[†]Professor and Director, Tier 1 Canada Research Chair in Computational Aerodynamics, Associate Fellow AIAA

II. Design Parametrization and Mesh Movement

The choice of parametrization places implicit constraints on the design space. The design variables should be sufficiently powerful, in the sense of approximation theory, to minimize the effects of these constraints. In turn, the mesh movement algorithm should be capable of accepting any geometrically physical design produced by the parametrization. For the present work, we use the control points from B-spline meshes for both parametrization and mesh movement. The complete details of the this integrated approach are given in a companion paper.[4] We provide only a brief outline of the method below.

A. B-spline Meshes

Consider a B-spline tensor product volume mapping the cubic computational domain $\mathcal{D} = \{\boldsymbol{\xi} = (\xi, \eta, \zeta) \in \mathbb{R}^3 | \xi, \eta, \zeta \in [0, 1]\}$ to the physical domain $\mathcal{P} \in \mathbb{R}^3$:

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

The set of points $\{\mathbf{B}_{ijk}\}$ defines the control mesh which can be used to manipulate the volume. The functions $\mathcal{N}_i$, $\mathcal{N}_j$, and $\mathcal{N}_k$, described below, are the modified B-spline basis functions. A B-spline mesh is obtained by discretizing the cubic domain $\mathcal{D}$.

The basis functions $\mathcal{N}_i$, $\mathcal{N}_j$, and $\mathcal{N}_k$ are modified versions of the standard B-spline basis; for definitions of the standard basis see de Boor.[5] The modified bases differ in that they use spatially varying knot vectors, also called curved knot lines.[6] For example, the knots defining $\mathcal{N}_i(\xi; \eta, \zeta)$ are bilinear functions of η and ζ; hence, for fixed (η, ζ) the function $\mathcal{N}_i(\xi; \eta, \zeta)$ reduces to the standard basis. The spatially varying knot vectors permit better approximations of the initial grid and geometry.

Suppose an initial design surface is given, together with a conforming computational mesh. A preprocessing stage approximates each block of the grid using B-spline volumes. The approximation algorithm is based on the parameter correction method proposed by Hoschek.[7] The initial (ξ, η, ζ) parameters and knot vectors are given a chord length spacing, which produces a control mesh that mimics a coarse grid. Thus, the control points corresponding to the surface patches provide a low dimensional approximation of the design. Bézier or B-spline control points have been used as design variables by many authors; see, for example, Burgreen and Baysal[8] or Nemec et al.;[9] the difference here is that the design variables are a subset of a volume control mesh.

To formalize the design variable definition, let $\mathbf{b}$ be a block column vector of all control points. Furthermore, let $\mathbf{b}$ be ordered such that all internal control points are first, then non-surface boundary points, and finally the N_s surface control points. The vector of (geometric) design variables is defined by the restriction operation

$$\mathbf{v} \equiv \begin{bmatrix} 0 & 0 & I \end{bmatrix} \mathbf{b} \tag{2}$$

where I is the $3N_s \times 3N_s$ identity matrix.

B. Semi-Algebraic Mesh Movement

When the geometric design variables change, the mesh must be updated to conform to the new surface. Here, the surface control points are the design variables, so the surface mesh automatically conforms to the design. Moreover, by moving the internal control points in response to the surface control points, we can maintain the quality of the volume grid.

As noted above, the control mesh obtained by fitting the initial grid is a coarse approximation of the actual mesh. By applying different mesh movement algorithms to the control points, rather than the grid, a family of mesh movement algorithms is possible. For example, we use a linear elasticity algorithm, described by Truong et al.,[10] to perturb the internal control mesh based on the surface control points. Subsequently, the fine grid is regenerated algebraically using equation (1).

We briefly describe the mesh movement algorithm of Truong et al.[10] to help elucidate the details of the gradient calculation in Section IV. The algorithm breaks the movement into m equal increments to improve robustness — the linear elasticity equation assumes small displacements. The residual vectors, obtained from a trilinear finite-element discretization of the governing equations, are

$$\boldsymbol{\mathcal{M}}^{(i)}\left(\mathbf{b}^{(i)},\mathbf{b}^{(i-1)}\right) = K^{(i)}\left(\mathbf{b}^{(i-1)}\right)\left[\mathbf{b}^{(i)}-\mathbf{b}^{(i-1)}\right] - \mathbf{f}^{(i)} = 0, \qquad i = 1,\ldots,m. \tag{3}$$

In the above equation, $\mathbf{b}^{(i)}$ is the vector of B-spline control points at increment stage i, and $\mathbf{f}^{(i)}$ is the vector of forces. The forces are defined implicitly to account for the boundary movement. The stiffness matrix $K^{(i)}$ is symmetric positive definite with respect to the unknown degrees of freedom; thus, we use the conjugate gradient method with ILU(p)[11] preconditioning to solve (3) for each increment i. These equations can be expensive to solve when applied to the actual computational mesh; however, since the control mesh has 2 to 3 orders fewer degrees of freedom, solutions to (3) are obtained in negligible time relative to the flow solver. For example, consider a 12 block mesh fitted using $9 \times 9 \times 9$ control points per block. Then solving equations (3) with $m = 5$, to a relative tolerance of 10^{-12}, requires 128 seconds on a 1500 MHz Itanium 2 processor.

Notice that the stiffness matrix in (3) is a function of the B-spline control points at the previous increment. This functional dependence arises because the Young's modulus is assumed to depend on the control mesh cell volumes and quality. In other words, for each control mesh element $\mathcal{E}$ we have

$$E_{\mathcal{E}}^{(i)} = E_{\mathcal{E}}^{(i)}(\mathbf{b}^{(i-1)}).$$

This (nonlinear) dependence on the previous increment is important to consider when differentiating the mesh movement algorithm.

We now summarize the design parametrization and mesh movement. The grid around an initial design is approximated using B-spline volumes. The coordinates of the surface control points become the design variables. When the design changes, the internal control points are updated using linear elasticity, and the grid is regenerated algebraically.

III. Flow Analysis

The flow solver incorporated into the optimization algorithm uses a second-order accurate finite-difference discretization and a Newton-Krylov solution strategy. The solver is described briefly below, and in detail in reference 12.

A. Governing Equations and Discretization

In this study we consider the 3-dimensional Euler equations on multi-block structured grids. Applying a diffeomorphism from physical to computational space, the Euler equations become

$$\partial_t \hat{\mathbf{Q}} + \partial_{\xi_i} \hat{\mathbf{E}}_i = \mathbf{0}, \tag{4}$$

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

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

The scalar J denotes the Jacobian of the mapping, and the U_i are the contravariant velocities defined by $U_i = u_j \partial_{x_j} \xi_i$.

The convective fluxes in (4) are discretized using second-order accurate summation-by-parts (SBP) operators.[13] Boundary conditions are imposed and blocks are coupled using simultaneous approximation terms (SATs).[14] The SBP-SAT discretization is linearly time-stable, requires only C^0 mesh continuity at block interfaces, accommodates arbitrary block topologies, and has low inter-block communication overhead. To suppress high frequency modes, the discretization is augmented with scalar dissipation,[15,16] and, in some cases, with matrix dissipation.[17] The resulting nonlinear set of algebraic equations is represented by the vector equation

$$\boldsymbol{\mathcal{F}}(\mathbf{q}) = \mathbf{0}, \tag{5}$$

where $\mathbf{q}$ is a block column vector of the conservative flow variables.

B. Inexact-Newton Algorithm

We solve the discretized Euler equations using a Newton-Krylov strategy. Applying Newton's method to (5) produces the following linear update for each Newton (outer) iteration n:

$$\mathcal{A}^{(n)}\boldsymbol{\Delta}\mathbf{q}^{(n)} = -\boldsymbol{\mathcal{F}}^{(n)}, \tag{6}$$

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

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

Successful convergence of Newton's method depends on the initial iterate, $\mathbf{q}^{(0)}$, which must be sufficiently close to the solution of (5).[18] For this reason, our solution algorithm is broken into two phases: 1) a start-up phase to find a suitable "initial" iterate, and; 2) an inexact-Newton phase. We describe these phases below, starting with the second.

The inexact-Newton phase gets its name from solving (6) inexactly with, for example, a Krylov linear solver. This strategy is useful, because an inexpensive inexact-Newton update may reduce the nonlinear residual by the same magnitude as the exact update. In addition, Krylov solvers do not need the Jacobian matrix explicitly; they only need Jacobian-vector products of the form $\mathcal{A}^{(n)}\mathbf{z}$. This matrix-vector product can be calculated to first-order with the forward-difference equation

$$\mathcal{A}^{(n)}\mathbf{z} \approx \frac{\boldsymbol{\mathcal{F}}(\mathbf{q}^n + \epsilon\mathbf{z}) - \boldsymbol{\mathcal{F}}(\mathbf{q}^n)}{\epsilon}. \tag{7}$$

The perturbation parameter must be chosen carefully to minimize truncation error and avoid round-off errors.[19] We use a perturbation suggested by Nielsen et al.:[20]

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

where $\delta = 10^{-13}$ and N is the number of unknowns.

The start-up phase is invoked to find an initial iterate for the inexact-Newton phase. The start-up phase is similar to the implicit Euler method with some important modifications which we now describe. The time step varies spatially with mesh spacing to roughly approximate a constant CFL.[16] The Jacobian $\mathcal{A}^{(n)}$ is replaced with a first-order approximation obtained by lumping the fourth-difference dissipation into the second-difference dissipation. This lumping reduces the stencil size and improves diagonal dominance. During the start-up phase the approximate Jacobian is updated and incompletely factored every four iterations[21] — factorization is used in preconditioning the linear update equation, see below. This lagged update strategy reduces factorization time while maintaining robustness.[12]

C. Parallel Krylov Linear System Solution

At each outer iteration of the inexact-Newton algorithm, we must solve a linear equation of the form

$$\mathcal{A}\mathbf{x} = \mathbf{r}. \tag{8}$$

While the matrix $\mathcal{A}$ is different during the start-up and inexact-Newton phases, the solution strategy for the linear equation remains the same. Specifically, we solve (8) in parallel using a preconditioned Krylov iterative method. Experience suggests that the generalized minimal residual method (GMRES)[22] is an efficient Krylov method for aerodynamic applications. We use flexible GMRES (FGMRES)[23] to accommodate iterative preconditioners.

Krylov methods need to be preconditioned to be effective on ill-conditioned problems. Two parallel preconditioners are implemented in the solver: an additive Schwarz preconditioner[24,25] with no overlap (block Jacobi), and an approximate Schur preconditioner.[26] Both preconditioners require an incomplete lower-upper factorization of the approximate local Jacobian matrices $\mathcal{A}_k$. The matrix $\mathcal{A}_k$ is local because it contains only those columns and rows corresponding to variables assigned to processor k; it is approximate, because it lumps the fourth-difference dissipation into the second-difference dissipation. ILU(p)[11] with a fill-level of 1 is used for the incomplete factorizations.

IV. Lagrangian Formulation and Gradient Evaluation

Let $\mathcal{J}$ denote an objective function that we wish to minimize; for example, drag or $C_{\mathrm{D}}/C_{\mathrm{L}}$. We require that a valid optimal point satisfy the mesh movement and flow equations; therefore, the optimization problem can be posed mathematically as follows:

$$\begin{aligned}
\text{min} \quad & \mathcal{J}\left(\mathbf{q}, \mathbf{b}^{(m)}, \mathbf{v}\right) \\
\text{w.r.t.} \quad & \mathbf{q}, \mathbf{b}^{(m)}, \mathbf{v}, \\
\text{s.t.} \quad & \boldsymbol{\mathcal{M}}^{(i)}\left(\mathbf{b}^{(i)}, \mathbf{b}^{(i-1)}\right) = 0, \qquad i \in \{1, 2, \ldots, m\} \\
& \boldsymbol{\mathcal{F}}\left(\mathbf{q}, \mathbf{b}^{(m)}\right) = 0
\end{aligned} \tag{9}$$

Recall that $\mathbf{v}$ are the design variables, $\mathbf{b}^{(i)}$ are the B-spline control points, and $\mathbf{q}$ are the flow variables.

Following the standard approach, we introduce the Lagrangian function, $\mathcal{L}$, for the constrained optimization problem (9):

$$\mathcal{L} = \mathcal{J} + \sum_{i=1}^{m} \boldsymbol{\lambda}^{(i)T} \boldsymbol{\mathcal{M}}^{(i)} + \boldsymbol{\psi}^T \boldsymbol{\mathcal{F}}. \tag{10}$$

The Lagrange multipliers $\boldsymbol{\lambda}^{(i)}$ and $\boldsymbol{\psi}$ are the mesh and flow adjoint variables, respectively. The first-order (necessary) optimality conditions for the problem (9) are obtained by setting the partial derivatives of $\mathcal{L}$ to zero:[27]

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\lambda}^{(i)}} = 0 = \boldsymbol{\mathcal{M}}^{(i)}, \qquad i \in \{1, 2, \ldots, m\} \tag{11}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\psi}} = 0 = \boldsymbol{\mathcal{F}} \tag{12}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}} = 0 = \frac{\partial J}{\partial \mathbf{q}} + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{\mathcal{F}}}{\partial \mathbf{q}} \tag{13}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(m)}} = 0 = \frac{\partial \mathcal{J}}{\partial \mathbf{b}^{(m)}} + \boldsymbol{\lambda}^{(i)T} \frac{\partial \boldsymbol{\mathcal{M}}^{(m)}}{\partial \mathbf{b}^{(m)}} + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{\mathcal{F}}}{\partial \mathbf{b}^{(m)}} \tag{14}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(i)}} = 0 = \boldsymbol{\lambda}^{(i)T} \frac{\partial \boldsymbol{\mathcal{M}}^{(i)}}{\partial \mathbf{b}^{(i)}} + \boldsymbol{\lambda}^{(i+1)T} \frac{\partial \boldsymbol{\mathcal{M}}^{(i+1)}}{\partial \mathbf{b}^{(i)}}, \qquad i \in \{m-1, m-2, \ldots, 1\} \tag{15}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = 0 = \frac{\partial \mathcal{J}}{\partial \mathbf{v}} + \sum_{i=1}^{m} \left(\boldsymbol{\lambda}^{(i)T} \frac{\partial \boldsymbol{\mathcal{M}}^{(i)}}{\partial \mathbf{b}^{(i)}} \frac{\partial \mathbf{b}^{(i)}}{\partial \mathbf{v}} \right) + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{\mathcal{F}}}{\partial \mathbf{v}} \tag{16}$$

The optimality conditions are also called the Karush-Kuhn-Tucker (KKT) conditions.[27] Note the use of a bold-font notation for gradients and Jacobians; for example, $\left(\partial J/\partial \mathbf{q}\right)_i = \partial J/\partial q_i$. We hope this notation will clarify which terms in the KKT conditions are vectors and which are matrices.

We follow the approach outlined in Truong et al.,[10] and drive the KKT conditions to zero using a sequential approach. We begin with the design variables, $\mathbf{v}$, which define the surface B-spline control points; see equation (2). These control points are then used as boundary conditions to solve for the control mesh equations (11). Once the control mesh is determined and the mesh is generated, the flow equations (12) can be solved to find $\mathbf{q}$. Subsequently, the adjoint variables can be determined using equations (13)–(15), in that order. Finally, the gradient of the Lagrangian is calculated using (16). Using this sequential approach, all of the KKT conditions are satisfied at each step, with the exception of $\partial \mathcal{L}/\partial \mathbf{v} = 0$. The gradient of the Lagrangian, $\partial \mathcal{L}/\partial \mathbf{v}$, is driven to zero using the sequential quadratic programming (SQP) algorithm in the software SNOPT.[28]

A. Flow Adjoint Equation

The flow adjoint variables are governed by the linear equation

$$\mathcal{A}^T \boldsymbol{\psi} = -\left(\frac{\partial J}{\partial \mathbf{q}}\right)^T. \tag{17}$$

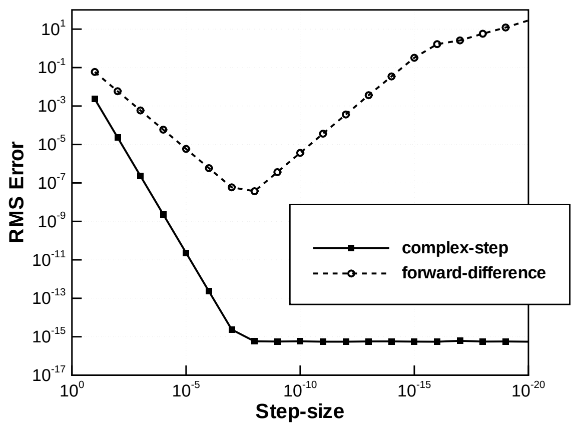


Figure 1. Verification of the analytical Jacobian matrix using the complex-step method

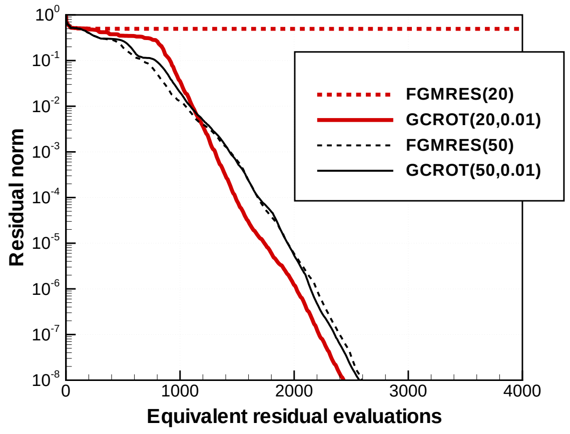


Figure 2. Comparison of CPU times for FGMRES and GCROT applied to the flow adjoint problem.

Recall that $\mathcal{A} = \partial\mathcal{F}/\partial\mathbf{q}$ is the Jacobian matrix of the flow residual. Many strategies have been proposed for the solution of equation (17), but most use algorithms based on the flow solver and take advantage of the identical eigenvalues shared by $\mathcal{A}$ and $\mathcal{A}^T$. Indeed, this is the approach taken in the present work: we solve (17) using a Krylov iterative solver.

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[29] use the complex-step method[30] with colouring to efficiently and accurately evaluate the entries of the Jacobian matrix. Mader et al.[31] construct the residual on a node-by-node basis, and then evaluate each row of the transposed Jacobian by applying the reverse-mode of automatic differentiation.

In the present work, we use a combination of hand-differentiation and complex-step to evaluate the Jacobian matrix. The Euler fluxes and numerical dissipation are differentiated by hand; much of this work was already completed for the approximate Jacobian, which is used to form the preconditioner required by the flow solver. Following Nemec,[9] the pressure switch used in the numerical dissipation is not differentiated. This approximation introduces small errors,[32] but helps to reduce the stencil size; moreover, the errors are only present when second-difference dissipation is active for shock-capturing. The SAT operators that couple blocks and impose boundary conditions have the following form:

$$\mathbf{\Sigma}_{\text{SAT}} = -\frac{1}{2}\left(|A| \pm A\right)\Delta\mathbf{Q}$$

where $(|A| \pm A)$ is the flux Jacobian for left- or right-traveling waves, and $\Delta\mathbf{Q}$ is the flow variable difference between nodes on a block interface or between a node and the boundary value. The SAT operators are differentiated using the complex-step method.[30] SAT terms appear only on block sides, so the application of the complex-step method does not significantly affect the CPU time.

To verify the accuracy of the Jacobian matrix, a complex-variable version of the flow residual was developed. When applied to the flow residual, the complex-step method[30] provides an accurate evaluation of the matrix-vector product $\mathcal{A}\mathbf{z}$. Figure 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. Errors between the Jacobian-vector product and products found using first-order forward differences, equation (7), are also included in the figure. The same random vector, $\mathbf{z}$, is used in each of the products, and second-difference dissipation is removed. These results strongly suggest that the Jacobian matrix is accurate to machine error.

Once the Jacobian is available, many linear solvers can be applied to the solution of the flow adjoints. To ensure accurate gradients, the flow adjoint system is solved to a relative tolerance of 10^{-8}. This tolerance requires a considerable number of Krylov iterations, unlike the looser 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.

For this work, we use a flexible variant of the Krylov method GCROT,[33] preconditioned with the (transposed) approximate Schur preconditioner used in the flow solver. 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.

We demonstrate the performance of GCROT 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 3(a). We consider FGMRES(m) and GCROT(m,δ) with $m = 20$ and $m = 50$. The parameter δ provides a threshold below which vectors are discarded by our implementation of GCROT; we use $\delta = 0.01$. Both Krylov solvers are preconditioned with the approximate Schur preconditioner constructed using an ILU(2) factorization of the local block matrices.

Figure 2 plots the L^2-norm of the relative residual versus equivalent residual evaluations. An equivalent residual evaluation is the CPU time needed to compute the residual $\boldsymbol{\mathcal{F}}$. Typically, our flow solver requires approximately 7000 equivalent residual evaluations to converge ten orders of magnitude. Figure 2 indicates that the adjoint system is solved with GCROT in 35 % of the time needed for a 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. Our 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.

We conclude this section by mentioning a Krylov-based adjoint approach suggested by Griewank,[34] in the general case, and by Martins,[35] in the case of aerodynamic optimization. Krylov iterative methods only require matrix-vector products of the form $\mathcal{A}^T\mathbf{z}$. These products can be computed using reverse-mode automatic differentiation, much in the same way the forward-difference equation (7) 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 may diminish.

B. Mesh Adjoint Equations

There are two types of B-spline mesh adjoint equations, given by (14) and (15), which we restate here in a transposed form:

$$\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(m)}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\lambda}^{(m)} = -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{b}^{(m)}}\right)^T - \left(\frac{\partial \boldsymbol{\mathcal{F}}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\psi} \tag{18}$$

$$\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i)}}{\partial \mathbf{b}^{(i)}}\right)^T \boldsymbol{\lambda}^{(i)} = -\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i+1)}}{\partial \mathbf{b}^{(i)}}\right)^T \boldsymbol{\lambda}^{(i+1)}, \qquad i \in \{m-1, m-2, \ldots, 1\} \tag{19}$$

We will begin by discussing the left-hand side of the linear systems (18) and (19). The system matrix appearing in these equations can be found by differentiating the control mesh movement equation (3) 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 us to reuse the solution strategy applied to equation (3), i.e. we use the conjugate gradient method preconditioned with ILU(p).

Unlike the left-hand sides, the right-hand sides of equations (18) and (19) are very different. To evaluate the right-hand side of the adjoint equation (18) we make liberal use of the chain rule:

$$-\left(\frac{\partial \mathcal{J}}{\partial \mathbf{b}^{(m)}}\right)^T - \left(\frac{\partial \boldsymbol{\mathcal{F}}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\psi} = -\left(\frac{\partial \mathbf{x}}{\partial \mathbf{b}^{(m)}}\right)^T \left[\left.\frac{\partial \mathcal{J}}{\partial \mathbf{x}}\right|_{\mathbf{m}} + \left(\left.\frac{\partial \mathcal{J}}{\partial \mathbf{m}}\right|_{\mathbf{x}} + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{\mathcal{F}}}{\partial \mathbf{m}}\right) \frac{\partial \mathbf{m}}{\partial \mathbf{x}}\right]^T \tag{20}$$

where $\mathbf{x}$ are the grid coordinates, defined by equation (1), and $\mathbf{m}$ are the grid metrics. The term $\partial \mathcal{J}/\partial \mathbf{x}|_{\mathbf{m}}$ denotes the partial derivative of the objective with respect to the grid coordinates while freezing the metric terms; similarly for $\partial \mathcal{J}/\partial \mathbf{m}|_{\mathbf{x}}$. Equation (20) provides a right-hand-side reformulation that is significantly easier to implement. Note that none of the matrices appearing in (20) need to be stored, only the resulting vector-matrix and matrix-vector products.

When the number of increments is greater than one, we must solve the additional adjoint equations (19). 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 $\mathbf{b}^{(i)}$; therefore, in the present work, we evaluate the right-hand sides of (19) using the complex-step method. Evaluating the right-hand sides in this way requires approximately the same CPU time as solving the linear system. However, the relatively small control mesh implies only a small penalty in total CPU time. Clearly, we would not recommend using the complex-step method to evaluate the right-hand side of (19) when the equations of linear elasticity are applied to the individual grid points.

C. Gradient Accuracy

Given the complexity of the algorithm and the use of hand differentiation, verifying gradient accuracy is important. Consider a 12 block mesh around a generic wing with no sweep; see Figure 3(a). Each block consists of $23 \times 33 \times 17$ nodes and is fit with B-spline volumes. The wing is parametrized using the B-spline control points corresponding to the surface; these control points are depicted as white markers in Figure 3(a). In total, there are 297 geometric design variables. This total excludes the y-coordinate of control points on the symmetry plane and all coordinates of one control point on the symmetry plane, which is fixed to prevent translation.

Including the angle of attack, there are 298 design variables. Checking each computed gradient component against a finite-difference approximation would be time consuming and unnecessary. We use a directional derivative to check all the gradient components simultaneously. For a given direction $\mathbf{w}$ the analytical directional derivative is given by

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

while the second-order finite-difference approximation is

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

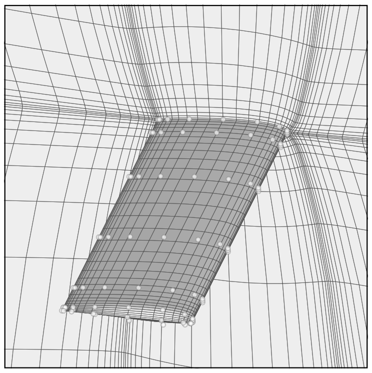
where ϵ is the perturbation parameter.

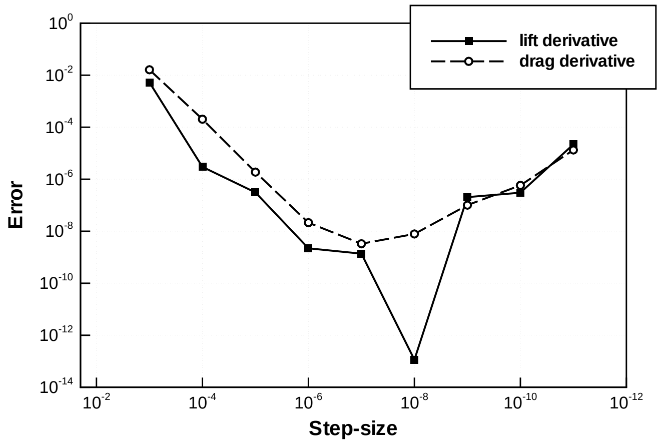
Individual components of the gradient can differ in magnitude by 2–4 orders. It is tempting, therefore, to choose a direction $\mathbf{w}$ such that each element of the gradient makes an equal contribution to $D_{\mathbf{w}}\mathcal{J}$; however, this tends to increase the step-size range over which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

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

which gives a directional derivative equal to the L^1 norm of the gradient. This direction may help reduce subtractive cancellation between gradient components.

Figure 3(b) plots the relative error between the analytical and finite-difference values of $D_{\mathbf{w}}\mathcal{J}$, 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 their finite-difference approximations with optimal step sizes.


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



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

Figure 3. Mesh, design variables, and results for gradient verification.

D. Optimization Algorithm

For the optimization, we use the software package SNOPT[28] developed for 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[a] is approximated using the quasi-Newton method BFGS (see, for example, reference 27). We use the full memory BFGS update rather than the limited memory option since the storage requirements are modest relative to the flow solver.

Convergence histories presented in this paper use an optimality measure defined by the SNOPT algorithm; the measure is closely related to the KKT conditions. For some results we also report the merit function history. 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 paper, the objective function is drag, so the merit function is equal to the drag at convergence.

We 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 V.B. 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 moves according to the leading edge control point. The number of geometric degrees of freedom is reduced to 6. 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 25% increase in CPU time per optimization cycle. In the future, we may consider imposing the lift constraint as an equation in the flow solver as implemented in our 2-dimensional optimizer, Optima2D.[32, 36]

As mentioned above, the optimization algorithm uses drag and lift for the objective and constraint, respectively. Choosing the dimensional variables over their non-dimensional versions, C_L and C_D, was motivated by the relationship between induced drag and lift:

$$D = \frac{L^2}{q\pi b^2 e} \tag{21}$$

where $q = \frac{1}{2}\rho_\infty U_\infty^2$ is the free-stream dynamic pressure, and b is the span. The efficiency factor e accounts for non-optimal loading. By fixing lift and span, equation (21) implies that the only means of reducing drag

[a]Note that the Lagrangian in SNOPT is not the same as the one defined in (10). 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.

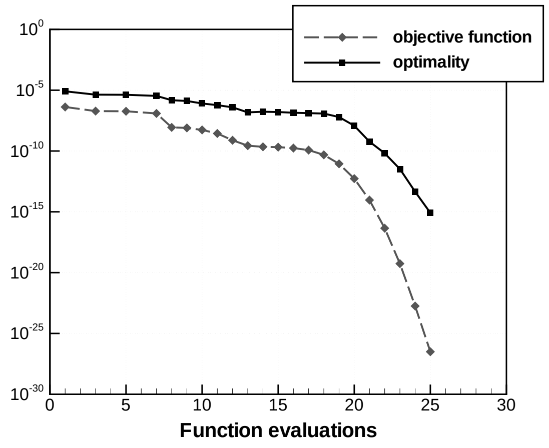


Figure 4. **Convergence history for the inverse design verification.**

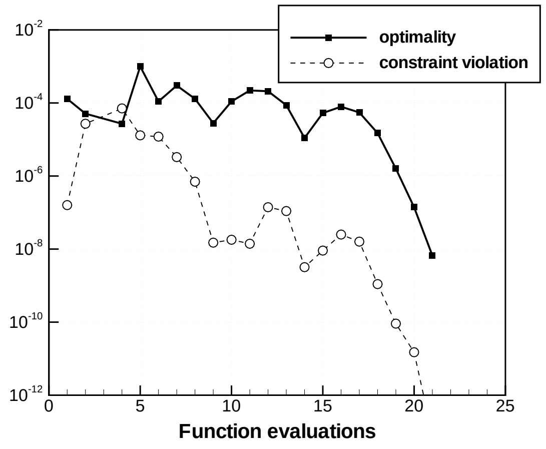


Figure 5. **Convergence history for the twist optimization case.**

is through the efficiency factor.

V. Verification and Validation

In this section, we 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. To our knowledge, twist optimization has not been used as a validation case in the literature.

The lift, drag, and area values reported in this section and the next are based on the half-span geometry. Unless otherwise indicated, the reference area is the projected area of the initial shape. In all examples the reference area is fixed, so we can report coefficients of lift and drag, rather than lift and drag, without ambiguity.

A. Inverse Design

As a simple verification, we 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 3(a). The initial angle of attack is four degrees. The target design is produced by randomly perturbing the 4 design variables. The optimizer is given the unperturbed wing and angle of attack as the initial design; the goal is to recover the perturbed shape and angle of attack based on a target pressure distribution.

To obtain the target pressure distribution, we 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

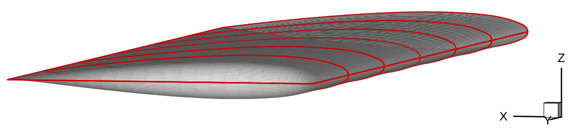
$$\mathcal{J} = \frac{1}{2}\sum_{i=1}^{N_{\text{surf}}} (p_i - p_{i,\text{targ}})^2 \Delta A_i$$

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.

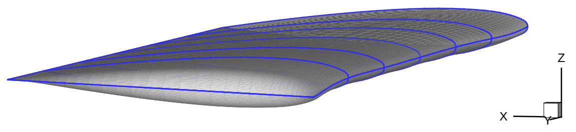
Figure 4 shows the convergence history for the inverse design problem. The gradient converges 10 orders and the objective converges 20 orders in 25 objective function and gradient evaluations.

B. Twist Optimization

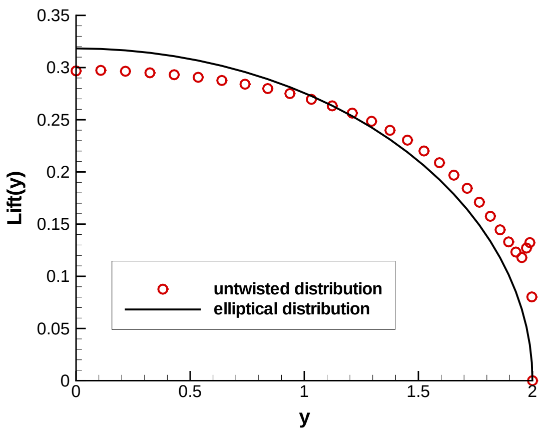
According to linear aerodynamic theory, induced drag for a planar wake is minimized by an elliptical span-wise lift distribution; thus, linear theory provides a useful benchmark for optimization algorithms. This



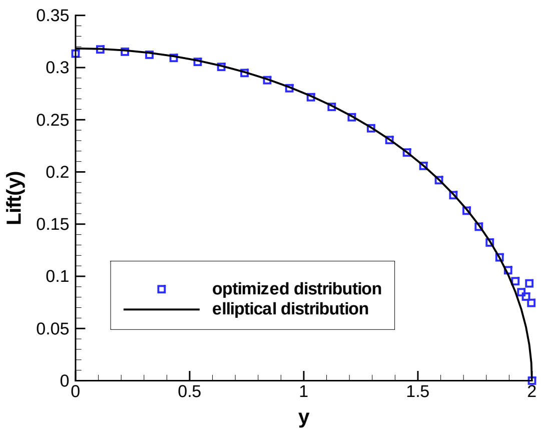
(a) Initial untwisted wing



(b) Optimized twist



(c) Initial distribution



(d) Optimized distribution

Figure 6. Initial and optimized designs and their lift distributions.

is also a challenging benchmark: the analysis in Appendix A 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, elliptic lift distributions are not unique. The same distribution can be obtained using changes in planform, twist, sectional lift, or some combination of these. For this validation, we vary the twist. The design examples, presented below, will illustrate why planform is a poor choice for recovering the optimal distribution predicted by linear theory.

The initial geometry 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 4/3. The grid has a 12 block topology with approximately 1.158×10^6 nodes. The blocks are fit with B-spline volumes such that the wing surface is parametrized with 9×7 control points on the upper and lower patches; the parametrization is similar to the one in Figure 3(a). The 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 design meets the C_L constraint of 0.375.

The trailing edge control points are fixed, and linear constraints are applied such that the twist of each spanwise station is a function of the z-coordinate of the leading edge control point. Fixing the trailing edge helps reduce non-planar effects, although, side-edge separation[37] makes a completely planar wake difficult to achieve in practice. Finally, the twist of the wingtip edge is constrained to be the same as the twist at the neighbouring control point section; this constraint is necessary to prevent mini-winglets.

The convergence history for the twist optimization is shown in Figure 5. 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 reduced approximately 2.1% from 0.00755 to 0.00739. The optimal induced drag predicted by lifting line theory, based on $C_L = 0.375$, is $C_{D,i} = 0.00746$; numerical errors may be responsible for the slightly lower drag produced by the algorithm.

Figure 6 shows the geometry and lift distribution 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 wing tip. The increased sectional lift is caused by the tip vortex. The side-edge separation induces a non-planar wake which implies that the elliptical lift

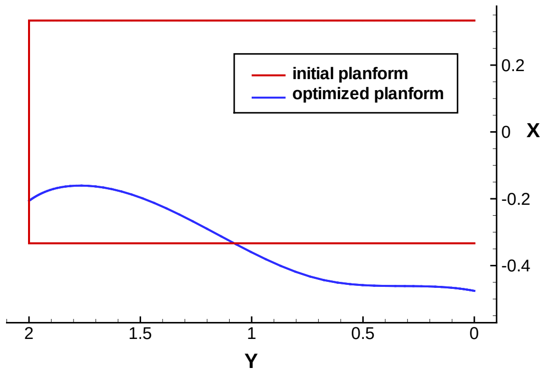


Figure 7. Initial and optimal planform shapes.

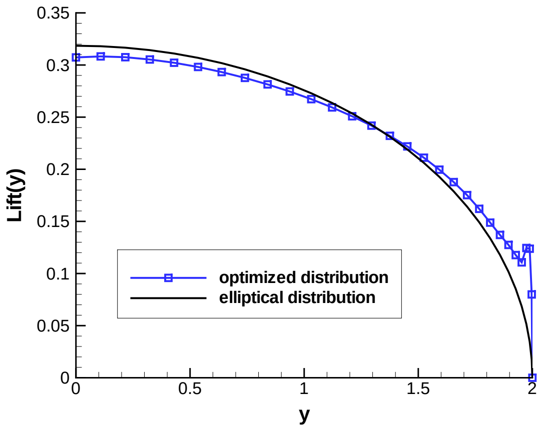


Figure 8. Elliptical lift distribution and distribution for optimal shape.

distribution is no-longer optimal.[37] In Section VI.A the influence of side-edge separation is discussed further in the context of planform optimization.

VI. Design Examples

A. Planform Optimization

An elliptical planform should also produce an elliptical lift distribution; therefore, according to linear theory for planar wakes, optimizing planform should produce an elliptical shape. By including the effects of wake geometry, Smith and Kroo[38] have shown that the optimal planform may not be elliptical depending on the shape of the trailing edge. Their results suggest we should expect some deviation from linear theory in terms of planform shape, but not necessarily lift distribution.

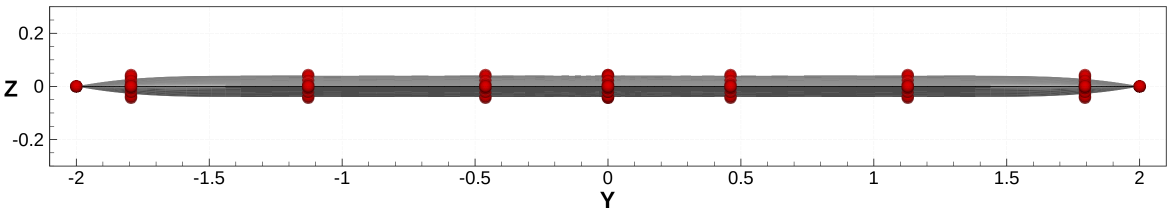
The initial geometry is a rectangular planform with a root chord of 2/3, semi-span of 2, and NACA 0012 sections. In this example, we use the surface area as the reference area, and its value is fixed using a constraint; see below. The upper and lower surfaces of the wing are parametrized using 9 nodes in the streamwise direction and 6 nodes in the spanwise direction. Again, the trailing edge control points are fixed to reduce the effects of a non-planar 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: for each angle of attack, there is an optimal planform. For this example we adopt a surface area constraint of $S = 1.36176$, the area of the initial planform shape.

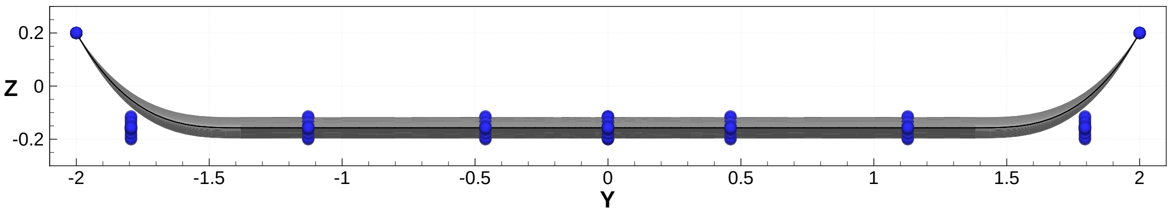
The Mach number is fixed at 0.5. The target lift coefficient is 0.36717, and the initial angle of attack is set such that this lift constraint is satisfied at the first iteration. Plots showing the optimality, constraint violation, and merit function history can be found in Figure 11 in the appendix.

The outline of the optimal planform is plotted in Figure 7. This planform is clearly not elliptical. More importantly, as Figure 8 demonstrates, the lift distribution is not elliptical. There are two possible explanations for this result: either this is the optimal planform, or numerical errors are overwhelming the physics. The grid is the same as the one used for the twist optimization: a 12 block topology with 1.158×10^6 nodes. This resolution usually provides sufficient accuracy for subsonic inviscid flows.

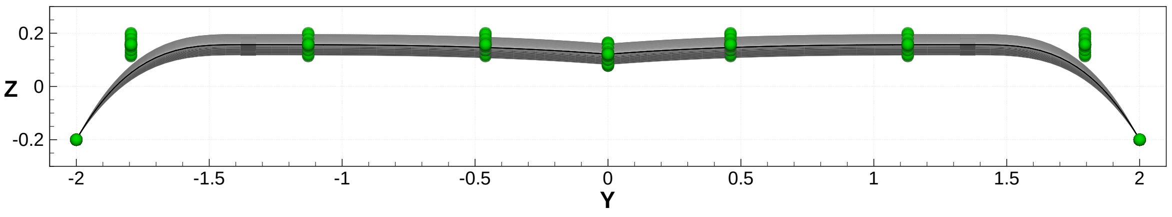
If numerical errors are not responsible for the non-elliptical lift distribution, there must be a physical explanation. We propose two possible mechanisms. First, the tip vortex releases close to the leading edge and creates a non-planar wake. In particular, the high-speed flow in the vortex lowers the pressure on the upper surface near the tip, similar to the vortices on a delta-wing. There may be a compromise between maintaining this vortex-induced low pressure region and establishing an elliptic lift distribution. A second explanation is that the tip spike creates secondary counter-rotating vorticity on its inboard side that interferes



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



(b) Optimized spanwise vertical shape: $\mathbf{C_D = 0.0069}$



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

Figure 9. Initial and optimized designs for spanwise vertical shape.

with the primary tip vortex.

Whatever mechanism is responsible for the optimal planform, the improvement over an elliptically loaded wing is small; see Figure 11(b). This is consistent with the analysis in Appendix A. Consequently, induced drag would likely have a negligible influence in a more realistic planform optimization including viscous and wave drag, and non-aerodynamic disciplines.

B. Spanwise Vertical Shape Optimization: Winglet Generation

For a fixed span, non-planar configurations can produce much lower induced drag than those with planar wakes. In this section and the next we consider examples that exploit non-planar wakes.

Again, the initial geometry is a rectangular wing with NACA 0012 sections, a semi-span of 2, and chord length of 2/3. The grid 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 direction, 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 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 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 the appendix, Figure 12.

The initial and optimized designs are shown in Figures 9(a) and 9(b), respectively. The optimized design has maximized the vertical extent near the tip, which Kroo has remarked is the "critical parameter" for non-planar systems.[3] The initial design has an induced drag of $C_D = 0.00752$, while the final design produces

$C_D = 0.0069$, a reduction of approximately 8.2%. Note that further reductions would be possible if twist or planform changes were permitted to optimally load the configuration.

To identify possible local optima for this problem, we 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 9(c); however, this configuration produces a drag reduction of only 3.9% ($C_D = 0.00723$). Eppler,[39] 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 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 vortex 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.

There is some evidence that the optimal winglet dihedral is strongly influenced by viscous effects. For example, Gerontakos and Lee[40] 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.[41]

C. Box-Wing Optimization

In this final example we optimize the loading for a box-wing configuration. For closed systems, like the box-wing, the optimal loading is not unique.[3] 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. A 6 block grid surrounds the box-wing geometry with approximately 6.02×10^5 nodes. The surface is parametrized using 9 control points in the streamwise direction and 5 control points in the spanwise and vertical directions. The vertical surfaces are linearly constrained by the upper and lower horizontal surfaces. 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 reference 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.

The convergence history for this problem is provided in Figure 13 in the appendix. 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}, despite decreasing the flow adjoint tolerance to 10^{-9}. This convergence difficulty may be related to the non-unique design space producing a singular Hessian.

Figure 10 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, linear theory predicts an optimal drag coefficient of 0.0133; hence, relative to a planar system, the optimized box-wing has reduced the drag by 12.6%.

VII. Summary and Conclusions

We have described a gradient-based algorithm for induced drag minimization. Parametrization and mesh movement are integrated using a B-spline volume mesh approach. Flow solutions are obtained using an efficient parallel Newton-Krylov algorithm and SBP-SAT discretization. A Lagrangian formulation is used to enforce the mesh movement and flow solution at each optimization design cycle. The objective gradient is calculated using a discrete adjoint approach. We have shown that the Krylov solver GCROT is an efficient and robust alternative to restarted GMRES when solving the flow adjoints. In the present implementation,

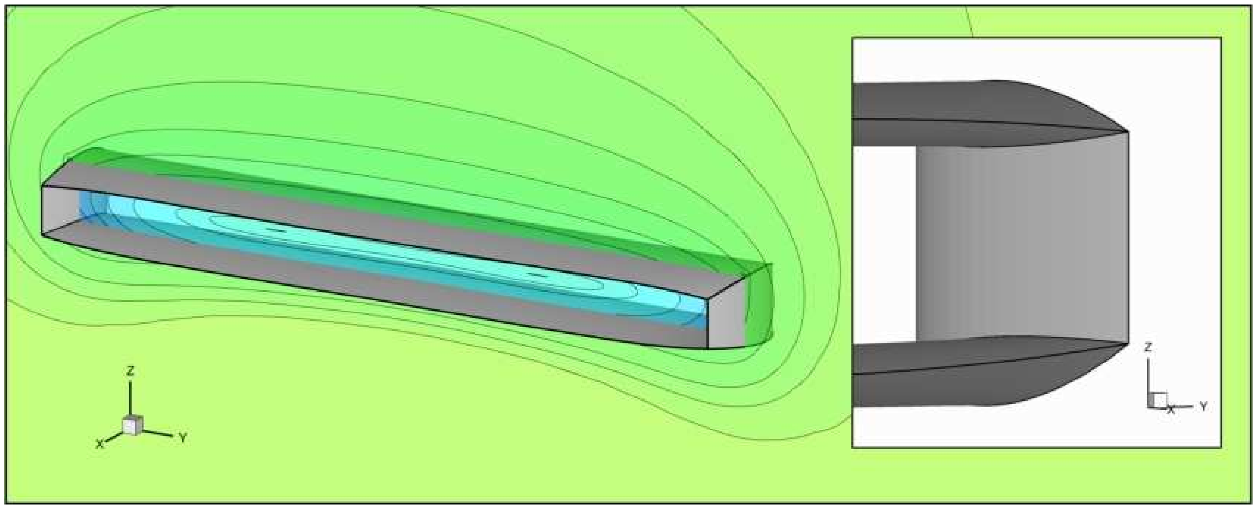


Figure 10. Optimized box-wing configuration with pressure contours on $x = 0$ slice. Inset: tip detail showing trailing edge shape.

the optimization is performed using the SNOPT package.

We have applied the optimization algorithm to a number of cases to verify, validate, and investigate its behaviour. The results of these cases are summarized below.

- Twist optimization provides a simple yet important validation case for aerodynamic optimization. Using twist to recover an elliptical lift distribution should be considered an essential benchmark.
- Unlike twist, planform is less suited to recovering an elliptical lift distribution. Planform design appears to exploit the non-planar characteristics of the tip vortex; this leads to a trade-off between the elliptical lift distribution and the strength of the tip vortex.
- Induced drag is not as sensitive to planform changes as it is to twist changes. This implies that numerical error must be carefully controlled to obtain a grid-converged planform shape. The insensitivity of induced drag to planform shape allows for greater flexibility in the design for viscous effects and non-aero disciplines.
- Optimization of spanwise vertical position lead to a winglet-up configuration. While the winglet-down configuration is a local optimum, it is not as efficient.

The present algorithm represents the first step toward a tool for optimization of unconventional configurations. Future work will include the addition of viscous and turbulence effects, and high-order discretizations.

Acknowledgments

The authors gratefully acknowledge financial assistance from the Natural Sciences and Engineering Research Council (NSERC), the Canada Research Chairs program, Mathematics of Information Technology and Complex systems (MITACS), and the University of Toronto.

References

[1]Committee on Aviation Environmental Protection (CAEP): seventh meeting, *The potential use of alternative fuels for aviation*, Montréal, Canada, Feb. 2007, International Civil Aviation Organization, CAEP/7-IP/28.

[2]Liebeck, R., "Design of the blended wing body subsonic transport," *Journal of Aircraft*, Vol. 41, No. 1, 2004, pp. 10–25.

[3]Kroo, I., "Drag due to lift: concepts for prediction and reduction," *Annual Review of Fluid Mechanics*, Vol. 33, 2001, pp. 587–617.

[4]Hicken, J. E. and Zingg, D. W., "Integrated parametrization and grid movement using B-spline meshes," *The 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, No. AIAA–2008–6079, Victoria, British Columbia, Canada, 2008.

[5]de Boor, C., *A practical guide to splines*, Springer–Verlag, Berlin, Germany, revised ed., 2001.

[6]Hayes, J. G., *Numerical Analysis*, chap. Curved knot lines and surfaces with ruled segments, Springer Berlin/ Heidelberg, 1982, pp. 140–156.

[7]Hoschek, J., "Intrinsic parametrization for approximation," *Computer Aided Geometric Design*, Vol. 5, No. 1, 1988, pp. 27–31.

[8]Burgreen, G. W. and Baysal, O., "Three-dimensional aerodynamic shape optimization using discrete sensitivity analysis," *AIAA Journal*, Vol. 34, No. 9, Sept. 1996, pp. 1761–1770.

[9]Nemec, M., Zingg, D. W., and Pulliam, T. H., "Multipoint and multi-objective aerodynamic shape optimization," *AIAA Journal*, Vol. 42, No. 6, 2004, pp. 1057–1065.

[10]Truong, A. H., Oldfield, C. A., and Zingg, D. W., "Mesh movement for a discrete-adjoint Newton-Krylov algorithm for aerodynamic optimization," *AIAA Journal*, Vol. 46, No. 7, July 2008, pp. 1695–1704.

[11]Meijerink, J. A. and van der Vorst, H. A., "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix," *Mathematics of Computation*, Vol. 31, No. 137, Jan. 1977, pp. 148–162.

[12]Hicken, J. E. and Zingg, D. W., "A parallel Newton-Krylov flow solver for the Euler equations on multi-block grids," *18th AIAA Computational Fluid Dynamics Conference*, No. AIAA–2007–4333, Miami, Florida, United States, June 2007.

[13]Strand, B., "Summation by parts for finite difference approximations for d/dx," *Journal of Computational Physics*, , No. 110, 1994, pp. 47–67.

[14]Carpenter, M. H., Gottlieb, D., and Abarbanel, S., "Time-stable boundary conditions for finite-difference schemes solving hyperbolic systems: methodology and application to high-order compact schemes," *Journal of Computational Physics*, , No. 111, 1994, pp. 220–236.

[15]Jameson, A., Schmidt, W., and Turkel, E., "Numerical solution of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes," *14th Fluid and Plasma Dynamics Conference*, Palo Alto, CA, 1981, AIAA Paper 81–1259.

[16]Pulliam, T. H., "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, Brussels, Belgium, Jan. 1986.

[17]Swanson, R. C. and Turkel, E., "On central-difference and upwind schemes," *Journal of Computational Physics*, , No. 101, 1992, pp. 292–306.

[18]Kelley, C. T., *Solving Nonlinear Equations With Newton's Method*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003.

[19]Zingg, D. W. and Chisholm, T. T., "Jacobian-free Newton-Krylov methods: issues and solutions," *Proceedings of The Fourth International Conference on Computational Fluid Dynamics*, Ghent, Belgium, July 2006.

[20]Nielsen, E. J., Walters, R. W., Anderson, W. K., and Keyes, D. E., "Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code," *12th AIAA Computational Fluid Dynamics Conference*, San Diego, CA, 1995, AIAA Paper 95–1733.

[21]Kim, D. B. and Orkwis, P. D., "Jacobian update strategies for quadratic and near-quadratic convergence of Newton and Newton-like implicit schemes," *31st AIAA Aerospace Sciences Meeting and Exhibit*, No. AIAA–93–0878, Reno, Nevada, 1993.

[22]Saad, Y. and Schultz, M. H., "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, July 1986, pp. 856–869.

[23]Saad, Y., "A flexible inner-outer preconditioned GMRES algorithm," *SIAM Journal on Scientific and Statistical Computing*, Vol. 14, No. 2, 1993, pp. 461–469.

[24]Saad, Y., *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, 2nd ed., 2003.

[25]Gropp, W. D., Kaushik, D. K., Keyes, D. E., and Smith, B. F., "High-performance parallel implicit CFD," *Parallel Computing*, , No. 27, 2001, pp. 337–362.

[26]Saad, Y. and Sosonkina, M., "Distributed Schur complement techniques for general sparse linear systems," *SIAM Journal of Scientific Computing*, Vol. 21, No. 4, 1999, pp. 1337–1357.

[27]Nocedal, J. and Wright, S. J., *Numerical Optimization*, Springer–Verlag, Berlin, Germany, 1999.

[28]Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: an SQP algorithm for large-scale constrained optimization," *SIAM Journal on Optimization*, Vol. 12, No. 4, 2002, pp. 979–1006.

[29]Nielsen, E. J. and Kleb, B., "Efficient construction of discrete adjoint operators on unstructured grids by using complex variables," *The 43rd AIAA Aerospace Sciences Meeting and Exhibit*, No. AIAA–2005–0324, Reno, Nevada, 2005.

[30]Squire, W. and Trapp, G., "Using complex variables to estimate derivatives of real functions," *SIAM Review*, Vol. 40, No. 1, 1998, pp. 110–112.

[31]Mader, C. A., Martins, J. R. R. A., and Marta, A. C., "Towards Aircraft Design Using an Automatic Discrete Adjoint Solver," *18th AIAA Computational Fluid Dynamics Conference*, No. AIAA–2007–3953, Miami, Florida, United States, June 2007.

[32]Nemec, M., *Optimal Shape Design of Aerodynamic Configurations: A Newton-Krylov Approach*, Ph.D. thesis, University of Toronto, 2003.

[33]de Sturler, E., "Truncation strategies for optimal Krylov subspace methods," *SIAM Journal of Numerical Analysis*, Vol. 36, No. 3, 1999, pp. 864–889.

[34]Griewank, A., *Evaluating derivatives*, SIAM, Philadelphia, PA, 2000.

[35]Martins, J. R. R. A., personal communication, Feb. 2007.

[36]Zingg, D. W. and Billing, L., "Toward practical aerodynamic design through numerical optimization," *18th AIAA Computational Fluid Dynamics Conference*, No. AIAA–2007–3950, Miami, Florida, United States, June 2007.

[37]Smith, S. C., "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.

[38]Smith, S. C. and Kroo, I. M., "Computation of induced drag for elliptical and crescent-shaped wings," *Journal of Aircraft*, Vol. 30, No. 4, 1993, pp. 446–452.
[39]Eppler, R., "Induced drag and winglets," *Aerospace Science and Technology*, Vol. 1, No. 1, 1997, pp. 3–15.
[40]Gerontakos, P. and Lee, T., "Effects of winglet dihedral on a tip vortex," *Journal of Aircraft*, Vol. 43, No. 1, 2006, pp. 117–124.
[41]Maskell, E., "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.
[42]Anderson, J. D., *Fundamentals of aerodynamics*, McGraw–Hill, Inc., New York, NY, 3rd ed., 2001.
[43]Vretblad, A., *Fourier analysis and its applications*, Springer–Verlag, New York, NY, 2003.

A. Effect of Perturbations on the Lift Distribution and 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 42)

$$\Gamma_{\text{ellip}}(\theta) = 2bVA_1 \sin(\theta).$$

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. 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:

$$\|\Gamma - \Gamma_{\text{ellip}}\|_2 = \mathrm{O}(\epsilon) \tag{22}$$

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[43]):

$$|A_n| \leq \frac{c}{n^2} \tag{23}$$

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,[42]

$$\begin{aligned} C_{D,i} &= \left(\frac{\pi b^2 A_1^2}{S}\right)\left[1 + \epsilon^2 \sum_{n=2}^{\infty} n \left(\frac{A_n}{A_1}\right)^2\right] \\ &\leq \left(\frac{\pi b^2 A_1^2}{S}\right)\left[1 + \left(\frac{\epsilon c}{A_1}\right)^2 \sum_{n=2}^{\infty} \frac{1}{n^3}\right] && \text{(using inequality (23))} \\ &\leq \left(\frac{\pi b^2 A_1^2}{S}\right)\left[1 + \left(\frac{\epsilon c}{A_1}\right)^2 \left(\frac{\pi^2}{6} - 1\right)\right] && \text{(sum of bounding } p\text{-series, } p = 2) \end{aligned}$$

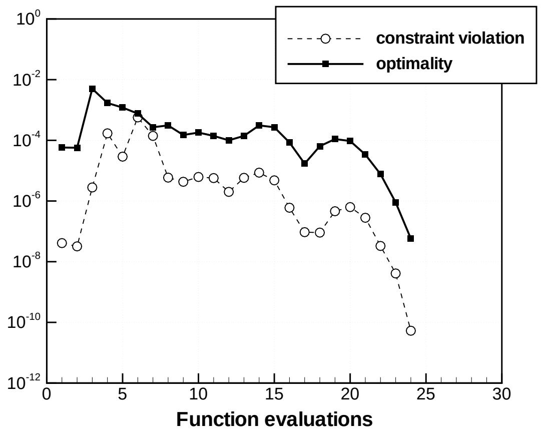
Thus, we have shown that the difference between the induced drags of the perturbed and elliptical distributions has the following asymptotic behaviour:

$$|C_{D,i} - C_{D,i,\text{ellip}}| = \mathrm{O}(\epsilon^2) \tag{24}$$

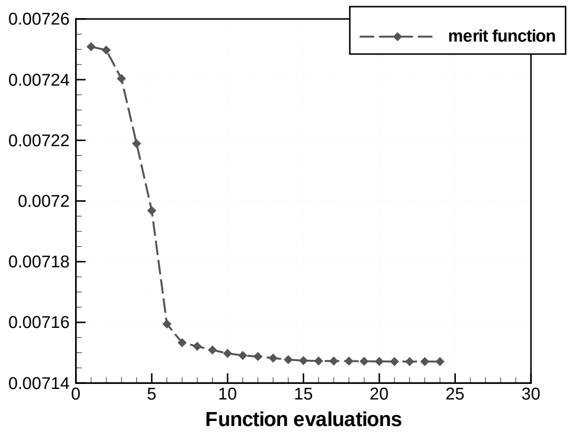
On the one hand, equations (22) and (24) suggest that a fairly large perturbation of the elliptical lift distribution will have a small effect on the induced drag. On the other hand, these equations underscore the difficulty of recovering the elliptical lift distribution through optimization: if we want to obtain a lift distribution that is within ϵ of the elliptical distribution, the induced drag must be accurate to within ϵ^2.

B. Optimization Convergence Histories

A. Planform Optimization



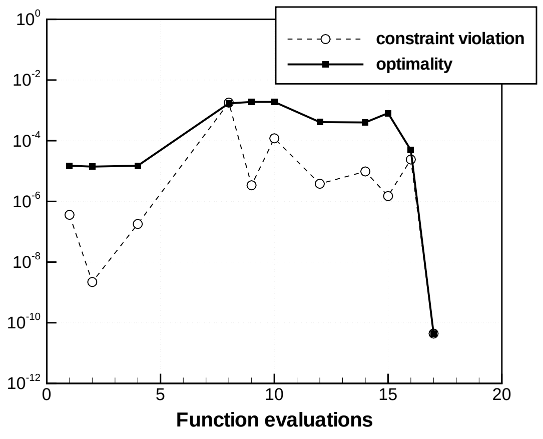
(a) optimality and constraint violation



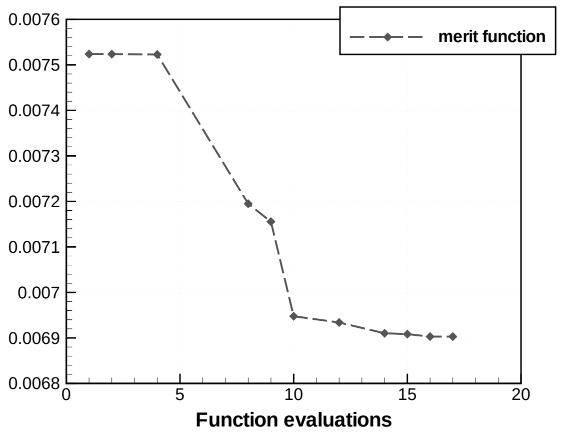
(b) merit function

Figure 11. Convergence history for the planform shape optimization. Note that the merit function becomes the objective (drag) as the constraint violation goes to zero.

B. Spanwise Vertical Shape Optimization



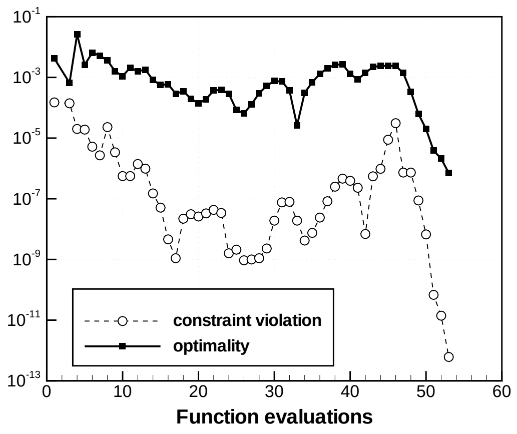
(a) optimality and constraint violation



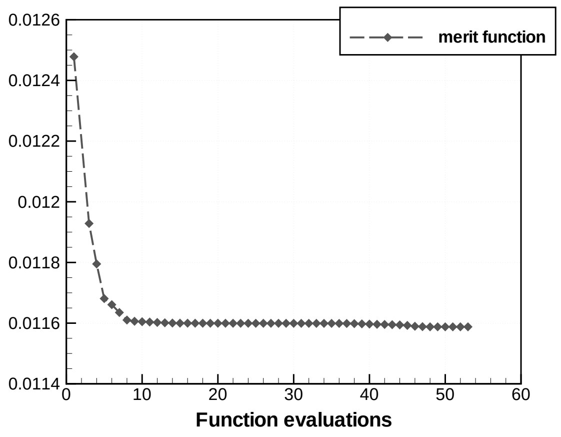
(b) merit function

Figure 12. Convergence history for the spanwise vertical shape optimization. Note that the merit function becomes the objective (drag) as the constraint violation goes to zero.

C. Box-Wing Optimization



(a) optimality and constraint violation



(b) merit function

Figure 13. Convergence history for the box-wing configuration shape optimization. Note that the merit function becomes the objective (drag) as the constraint violation goes to zero.