Please cite this document as:
A. B. Lambe and J. R. R. A. Martins. Matrix-free aerostructural optimization of aircraft wings. *Structural and Multidisciplinary Optimization*, 53(3):589603, March 2016. doi:10.1007/s00158-015-1349-2.
This document can be found at: `http://mdolab.engin.umich.edu`.

Matrix-Free Aerostructural Optimization of Aircraft Wings

Andrew B. Lambe[1] and Joaquim R. R. A. Martins[2]

Abstract In structural optimization subject to failure constraints, computing the gradients of a large number of functions with respect to a large number of design variables may not be computationally practical. Often, the number of constraints in these optimization problems is reduced using constraint aggregation at the expense of a higher mass of the optimal structural design. This work presents results of structural and coupled aerodynamic and structural design optimization of aircraft wings using a novel matrix-free augmented Lagrangian optimizer. By using a matrix-free optimizer, the computation of the full constraint Jacobian at each iteration is replaced by the computation of a small number of Jacobian-vector products. The low cost of the Jacobian-vector products allows optimization problems with thousands of failure constraints to be solved directly, mitigating the effects of constraint aggregation. The results indicate that the matrix-free optimizer reduces the computational work of solving the optimization problem by an order of magnitude compared to a traditional sequential quadratic programming optimizer. Furthermore, the use of a matrix-free optimizer makes the solution of large multidisciplinary design problems, in which gradient information must be obtained through iterative methods, computationally tractable.

Keywords: Structural Optimization, Multidisciplinary Design Optimization, Matrix-Free, Large-Scale Optimization, Constraint Aggregation]

1 Introduction

When solving structural optimization problems or multidisciplinary design optimization (MDO) Martins and Lambe (2013) problems involving a structural analysis, we want to include structural failure criteria directly in our problem formulation as constraints. However, to do so generally requires a large number of constraints to account for both the types of failure and the locations at which failure may occur. When the structural design is parametrized with a large number of design variables, the formulated optimization problems are expensive to solve.

The most common way of addressing the high cost of optimization is to reduce the size of the optimization problem using constraint aggregation (Poon and Martins, 2007). The basic idea of constraint aggregation is to replace a group of constraints with a single constraint that accurately models the feasible design space produced by the original set. For example, replacing a set of constraints $\vec{C}(\vec{x}) \leq 0$, where $\vec{C} : \mathbb{R}^n \to \mathbb{R}^m$, with the maximum-value function $\max_i\{C_i(\vec{x})\} \leq 0$ aggregates the constraint set perfectly. However, the maximum-value function is not smooth, so it is incompatible with gradient-based optimization methods. Because gradient-based optimizers are preferred for solving optimization problems with thousands of variables and constraints, we need to use aggregation techniques that are smooth.

[1]Postdoctoral Fellow, Department of Mechanical Engineering, York University, Toronto, Ontario, Canada
[2]Professor, Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan, USA

The dominant aggregation technique in use today is the Kreisselmeier-Steinhauser (KS) function Kreisselmeier and Steinhauser (1979). The KS function replaces $\vec{C}(\vec{x}) \leq 0$ with

$$KS\left(\vec{C}(\vec{x})\right) = c_{\max} + \frac{1}{\rho_{KS}} \ln \left[\sum_{i=1}^{m} \exp\left(\rho_{KS}(C_i(\vec{x}) - c_{\max})\right) \right] \leq 0, \tag{1}$$

where $c_{\max} = \max_i\{C_i(\vec{x})\}$, and ρ_{KS} is a user-specified parameter. The recommended value of ρ_{KS} from the literature to balance the smoothness of the KS function with the accuracy of the feasible region model is 50 (Raspanti et al., 2000; Akgün et al., 2001; Poon and Martins, 2007). Poon and Martins (2007) also discuss a strategy for adaptively increasing the KS parameter. Akgün et al. (2001) generalize the KS function (1) to a functional over a continuous domain. The KS functional has been further studied recently by Kennedy (2015) and Kennedy and Hicken (2015). For this work, our focus remains on the KS function (1).

One particularly attractive feature of the KS function is that it is a *conservative* aggregation technique. By conservative, we mean that if a given design point $\vec{x}$ is feasible with respect to the KS constraint (1), then it is necessarily feasible with respect to the original set of constraints. Obviously, this is a convenient feature when applied to constraints based on structural failure. However, the drawback is that the feasible design space generated by the KS function is an underestimate of the original feasible design space. The error in the KS function estimate of the maximum value is

$$c_{\max} \leq KS(\vec{C}(\vec{x})) \leq c_{\max} + \frac{\ln m}{\rho_{KS}}.$$

Because the design space is underestimated, the optimal structure generated from a mass minimization problem with KS constraints will necessarily have a higher mass than the optimal structure generated by the same problem without using KS aggregation. The amount of the mass overestimate is not known directly, but is driven by the number of aggregated constraints in the KS function and the choice of ρ_{KS}.

We propose to mitigate the issues of smooth constraint aggregation by formulating and solving optimization problems with thousands of variables and constraints. To do so, we require a gradient-based optimizer that is matrix-free; the optimizer may work with matrices of derivative information via matrix-vector products but cannot access the matrices directly. By not forming the derivative matrices, we greatly reduce the per-iteration cost of the optimization process and efficiently solve large structural and multidisciplinary design problems. We emphasize that we do not remove constraint aggregation entirely from our problem formulations. To do so would be practically impossible given that failure criteria for continuum structures are themselves continuum quantities which would have to be discretized or aggregated to form a finite-dimensional optimization problem. Rather, our goal is to reduce the computational cost of increasing the size of the optimization problem so that more accurate optimal mass values can be computed.

In this work, we aim to confirm the feasibility and performance of the matrix-free approach. We apply the matrix-free optimizer of Arreckx et al. (2015) to a pair of aircraft wing design problems to verify the performance of the matrix-free optimizer compared to a conventional optimizer. One problem considers only the structure itself, while the other considers both structural and aerodynamic properties of the wing. This paper elaborates on the results presented in our recent conference proceeding (Lambe and Martins, 2015).

The remainder of this paper is organized as follows: Section 2 reviews methods for computing gradients for optimization problems and highlights the utility of the matrix-free approach to optimization. Section 3 describes our augmented Lagrangian optimization algorithm and the adaptations necessary to make this algorithm matrix-free. Section 4 briefly describes our analysis and optimization software used to solve the test problems. Sections 5 and 6 present results on the two variations of the wing design problem. Section 7 summarizes our work on matrix-free optimization.

2 Gradient Computation

The exploration of matrix-free optimization methods is motivated by the structure of the gradient computations found in a "nested" problem formulation (Haftka and Kamat, 1989). A nested formulation is one in which the structural or multidisciplinary analysis is completed every time the objective and constraint functions of the optimization problem are evaluated. Formally, the problem is given by

$$\begin{aligned} \text{minimize} \quad & F(\vec{x},\vec{y}) \\ \text{with respect to} \quad & \vec{x} \\ \text{subject to} \quad & \vec{C}(\vec{x},\vec{y}) \leq 0 \\ & \vec{x}_L \leq \vec{x} \leq \vec{x}_U \\ \text{where} \quad & \vec{y} = \vec{Y}(\vec{x}) \text{ solves } \vec{R}(\vec{x},\vec{y}) = 0 \text{ given } \vec{x} \end{aligned} \tag{2}$$

where $\vec{x} \in \mathbb{R}^n$ represents the design variables, $\vec{x}_L \in \mathbb{R}^n$ and $\vec{x}_U \in \mathbb{R}^n$ are the lower and upper bounds of the design variables, F is the design objective, $\vec{y} \in \mathbb{R}^M$ represents the state variables, and $\vec{R} \in \mathbb{R}^M$ represents the (discretized) governing equations of the structural or multidisciplinary analysis. Throughout this work, we assume that $M \gg m,n$. Because the state variables are recomputed at each iteration of an optimization procedure via the structural or multidisciplinary analysis, we can define the state variables as implicit functions of the design variables. Therefore, $\vec{y} = \vec{Y}(\vec{x})$, where $\vec{Y}$ is an implicitly-defined function. To compute the total change in the objective or constraints with respect to the design variables requires accounting for the change in the state variables as well.

Our derivation of the gradient (sensitivity) expression follows that of Martins and Hwang (2013). We focus on the case of a structural or single-discipline optimization problem. Extensions to the multidisciplinary case are straightforward and are available in the literature (Sobieszczanski-Sobieski, 1990; Martins et al., 2005; Martins and Hwang, 2013). We adopt the convention of using capital letters to denote functions and the corresponding lower-case letters to denote variables representing computed values of the functions. For example, $\vec{c} = \vec{C}(\vec{x},\vec{y})$ shows the relationship between the function $\vec{C}$ and the resulting variables $\vec{c}$. Note that $\vec{c}$ still depends on $\vec{x}$ and $\vec{y}$ as a result.

The total change in the constraints with respect to the design variables is given by

$$\begin{aligned} \frac{\mathrm{d}\vec{c}}{\mathrm{d}\vec{x}} &= \frac{\partial \vec{C}}{\partial \vec{x}} + \frac{\partial \vec{C}}{\partial \vec{y}} \frac{\partial \vec{Y}}{\partial \vec{x}} \\ &= \frac{\partial \vec{C}}{\partial \vec{x}} + \frac{\partial \vec{C}}{\partial \vec{y}} \frac{\mathrm{d}\vec{y}}{\mathrm{d}\vec{x}}, \end{aligned} \tag{3}$$

where the shorthand

$$\frac{\partial \vec{C}}{\partial \vec{x}} = \frac{\partial (C_1,...,C_m)}{\partial (x_1,...,x_n)} \in \mathbb{R}^{m \times n}$$

is adopted to describe the Jacobian of a set of functions with respect to a set of variables. Partial derivatives may be computed by any method available to the user, e.g., finite differencing, the complex-step method (Martins et al., 2003), algorithmic differentiation, or analytically. However, note that the term $\partial \vec{Y} / \partial \vec{x}$ in equation (3) is implicitly-defined, so it is particularly troublesome to calculate by the traditional methods. A more efficient approach is to use the governing equations of the discipline and recognize that, because we adopted a nested formulation, a change in the design variables does not change the governing equation values. Therefore,

$$\frac{\mathrm{d}\vec{r}}{\mathrm{d}\vec{x}} = \frac{\partial \vec{R}}{\partial \vec{x}} + \frac{\partial \vec{R}}{\partial \vec{y}} \frac{\mathrm{d}\vec{y}}{\mathrm{d}\vec{x}} = 0. \tag{4}$$

(By convention, $\vec{r} = \vec{R}(\vec{x}, \vec{y})$.) By rearranging (4) and substituting the result back into (3), we obtain the final expression

$$\frac{\mathrm{d}\vec{c}}{\mathrm{d}\vec{x}} = \frac{\partial \vec{C}}{\partial \vec{x}} - \left[\frac{\partial \vec{C}}{\partial \vec{y}}\right]\left[\frac{\partial \vec{R}}{\partial \vec{y}}\right]^{-1}\frac{\partial \vec{R}}{\partial \vec{x}}. \tag{5}$$

Note that the same expression can be used to compute the objective function gradient by swapping F for $\vec{C}$.

Since the inverse term in (5) is only available through the solution of a set of linear equations, the computation of $\mathrm{d}\vec{c}/\,\mathrm{d}\vec{x}$ splits into two methods called the (discrete) *direct* and (discrete) *adjoint* methods. In the direct method, the linear systems are of the form

$$\left[\frac{\partial \vec{R}}{\partial \vec{y}}\right]\frac{\mathrm{d}\vec{y}}{\mathrm{d}x_j} = -\frac{\partial \vec{R}}{\partial x_j} \tag{6}$$

for a particular variable x_j, so the number of linear systems to solve to form the Jacobian (5) is identical to the number of variables, i.e., n linear solves. In the adjoint method, the linear systems are of the form

$$\left[\frac{\partial \vec{R}}{\partial \vec{y}}\right]^T\left[\frac{\mathrm{d}c_i}{\mathrm{d}\vec{r}}\right]^T = -\left[\frac{\partial C_i}{\partial \vec{y}}\right]^T \tag{7}$$

for a particular constraint C_i. The term $\mathrm{d}\vec{c}/\,\mathrm{d}\vec{r}$ enters equation (5) as

$$\frac{\mathrm{d}\vec{c}}{\mathrm{d}\vec{x}} = \frac{\partial \vec{C}}{\partial \vec{x}} + \frac{\mathrm{d}\vec{c}}{\mathrm{d}\vec{r}}\frac{\partial \vec{R}}{\partial \vec{x}}. \tag{8}$$

The number of linear systems to solve to form the Jacobian (5) using the adjoint method is identical to the number of constraints, i.e, m linear solves.

In both the direct and adjoint methods, the linear solves are the most expensive step in forming the Jacobian because the dimensions of $\vec{R}$ and $\vec{y}$ far exceed the number of variables and constraints in the optimization problem. However, the optimization problem size still drives the cost of the gradient computation by determining the number of linear solves. If problem (2) contains a large number of design variables but only a few constraints, the adjoint method is preferred because the same Jacobian can be formed at a lower cost. If problem (2) contains a large number of constraints and a small number of design variables, the direct method is preferred because, again, the same Jacobian can be formed at a lower cost. However, optimization problems like stress-constrained mass minimization can contain both thousands of design variables and thousands of constraints. In these problems, constraint aggregation is used to reduce the number of constraints, making the adjoint method attractive. The cost of the computation can be controlled by selecting the number of aggregated constraints to use in the optimization problem.

As noted in Section 1, we would like to mitigate the issues associated with constraint aggregation and still be able to solve optimization problems with many constraints efficiently. To avoid the high cost of forming the constraint Jacobian directly, we propose to access it only through matrix-vector products. If we multiply the Jacobian (5) by an arbitrary vector $\vec{v} \in \mathbb{R}^n$, we obtain the expression

$$\left[\frac{\mathrm{d}\vec{c}}{\mathrm{d}\vec{x}}\right]\vec{v} = \left[\frac{\partial \vec{C}}{\partial \vec{x}}\right]\vec{v} - \left[\frac{\partial \vec{C}}{\partial \vec{y}}\right]\left[\frac{\partial \vec{R}}{\partial \vec{y}}\right]^{-1}\left[\frac{\partial \vec{R}}{\partial \vec{x}}\right]\vec{v}. \tag{9}$$

Note that, since the multiplication can take place right-to-left, we only need to solve one linear system (with right-hand side $[\partial \vec{R}/\partial \vec{x}]\vec{v}$) to obtain the matrix-vector product. If we multiply the transpose Jacobian by an arbitrary vector $\vec{w} \in \mathbb{R}^m$, we obtain the expression

$$\left[\frac{\mathrm{d}\vec{c}}{\mathrm{d}\vec{x}}\right]^T\vec{w} = \left[\frac{\partial \vec{C}}{\partial \vec{x}}\right]^T\vec{w} - \left[\frac{\partial \vec{R}}{\partial \vec{x}}\right]^T\left[\frac{\partial \vec{R}}{\partial \vec{y}}\right]^{-T}\left[\frac{\partial \vec{C}}{\partial \vec{y}}\right]^T\vec{w}. \tag{10}$$

Again, evaluating the multiplications right-to-left, we only need to solve one linear system to obtain the matrix-vector product. Compared to forming the Jacobian itself, the cost of forming Jacobian-vector products is lower by a factor of n or m. Our goal, then, is to develop an efficient optimizer that can only access products of the constraint Jacobian, rather than the Jacobian itself. The resulting reduction in optimization cost would scale with the problem dimensions.

3 Optimization Algorithm

The idea of a "matrix-free" optimizer has been present in the optimization community for some time. Heinkenschloss and Vicente (1999) describe the basic requirements for a matrix-free interface to solve optimal control problems, though they do not describe the optimizers themselves. Griewank and Walther (2002) and Bosse (2009) develop matrix-free algorithms using quasi-Newton approximations to the relevant Hessian and Jacobian matrices. Curtis et al. (2009) describe an algorithm using inexact step computation that is tailored to sparse equality-constrained problems with rank-deficient Jacobians. Finally, Gondzio (2012) describes a matrix-free interior-point method for solving very large, sparse, quadratic problems. However, none of the optimizers described in these works apply to nonlinear optimization problems with inequality constraints and dense Jacobians. We therefore set out to develop our own optimization software.

In terms of which optimization algorithm to implement, optimizers of the sequential quadratic programming (SQP) or interior-point type are generally acknowledged to be the fastest gradient-based optimizers for general problems. However, more detailed investigation suggests that the augmented Lagrangian method presents fewer barriers to implementation as a matrix-free optimizer (Arreckx et al., 2015). Therefore, we have deliberately chosen to sacrifice some speed for ease of implementation in our solver.

We have developed a matrix-free implementation of the augmented Lagrangian algorithm of Conn et al. (1992). We have tested this optimizer on optimization problems with up to 5000 design variables and 5000 inequality constraints. In this section, we summarize the main features of our algorithm. Full details are presented by Arreckx et al. (2015), including benchmarking results and a scalability study using a simple structural optimization problem.

Given a standard nonlinear optimization problem,

$$\begin{aligned} \text{minimize} \quad & F(\vec{x}) \\ \text{w.r.t.} \quad & \vec{x} \\ \text{s.t.} \quad & \vec{C}(\vec{x}) \leq 0 \\ & \vec{x}_L \leq \vec{x} \leq \vec{x}_U, \end{aligned} \tag{11}$$

where $\vec{x}, \vec{x}_L, \vec{x}_U \in \mathbb{R}^n$ and $\vec{C} : \mathbb{R}^n \to \mathbb{R}^m$,the augmented Lagrangian algorithm introduces slack variables to transform inequality constraints into equality constraints and relaxes the equality constraints using a penalty function. The resulting problem,

$$\begin{aligned} \text{minimize} \quad & \Phi(\vec{x},\vec{t};\boldsymbol{\lambda},\rho) = F(\vec{x}) + \boldsymbol{\lambda}^T(\vec{C}(\vec{x}) - \vec{t}) + \\ & \frac{\rho}{2}(\vec{C}(\vec{x}) - \vec{t})^T(\vec{C}(\vec{x}) - \vec{t}) \\ \text{with respect to} \quad & \vec{x} \in \mathbb{R}^n, \vec{t} \in \mathbb{R}^m \\ \text{subject to} \quad & \vec{x}_L \leq \vec{x} \leq \vec{x}_U \\ & \vec{t} \geq 0, \end{aligned} \tag{12}$$

contains only a nonlinear objective and bound constraints. The Lagrange multipliers $\boldsymbol{\lambda} \in \mathbb{R}^m$ and penalty parameter ρ are updated after every minimization based on $||\vec{C}(\vec{x}) - \vec{t}||_\infty$, i.e., the maximum infeasibility of the constraints. This process is repeated until both the infeasibility metric $||\vec{C}(\vec{x}) - \vec{t}||_\infty$ and the norm of the

gradient of the Lagrangian $||\nabla F - [\nabla \vec{C}]\boldsymbol{\lambda}||_\infty$ reach a desired tolerance level, e.g., 10^{-5}, indicating that the algorithm has found a feasible local minimum. Rules given by Conn et al. (1992) allow problem (12) to be solved inexactly prior to each multiplier update, yet still converge. The augmented Lagrangian is preferred over other penalty methods because ρ does not need to become arbitrarily large for the algorithm to converge to a local minimum (Bertsekas, 1996).

Problem (12) is solved by an L_∞ trust region approach. (Nocedal and Wright, 2006, Chapter 4) Using the L_∞ norm rather than the more traditional L_2 norm allows us to easily handle the bound constraints within the trust-region subproblem. The trust-region subproblem is given by

$$\begin{aligned} \text{minimize} \quad & Q(\vec{p}) = \frac{1}{2}\vec{p}^T\vec{B}\vec{p} + \vec{g}^T\vec{p} \\ \text{with respect to} \quad & \vec{p} \in \mathbb{R}^{n+m} \\ \text{such that} \quad & \vec{p}_L \leq \vec{p} \leq \vec{p}_U, \end{aligned} \tag{13}$$

where $\vec{B} \in \mathbb{R}^{(n+m)\times(n+m)}$ is an estimate of the Hessian $\nabla^2\Phi$, $\vec{g} \in \mathbb{R}^{n+m}$ is the gradient $\nabla\Phi$, $\vec{p}$ is the search direction in both $\vec{x}$ and $\vec{t}$, and $\vec{p}_L$ and $\vec{p}_U$ represent the lower and upper bounds on the step $\vec{p}$. The quality of the step $\vec{p}$ is assessed by the usual trust-region metric: the ratio of the actual reduction in $\phi = \Phi(\vec{x},\vec{t};\boldsymbol{\lambda},\rho)$ to the predicted reduction in $q = Q(\vec{p})$. Subproblem (13) is solved by the algorithm of Moré and Toraldo (1991), modified to account for the case where the approximate Hessian $\vec{B}$ is indefinite. (The modification consists in stopping the conjugate-gradient part of the algorithm if negative curvature is detected, and returning the negative curvature direction. This is similar to the modified conjugate-gradient method proposed by Steihaug (1983).) Moré and Toraldo's algorithm needs only matrix-vector products with $\vec{B}$ to solve (13), so all that is needed to make the algorithm matrix-free is a suitable approximation to $\nabla^2\Phi$.

Analytically, the true Hessian of the augmented Lagrangian is given by

$$\nabla^2\Phi = \begin{bmatrix} \nabla^2 F + \sum_{i=1}^m \lambda_i \nabla^2 C_i + \sum_{i=1}^m \left(\rho\left(C_i(\vec{x}) - t_i\right)\right)\nabla^2 C_i + \rho\vec{J}^T\vec{J} & \rho\vec{J}^T \\ \rho\vec{J} & \rho\vec{I} \end{bmatrix}, \tag{14}$$

where $\vec{J} = \left[\nabla\vec{C}\right]^T \in \mathbb{R}^{m\times n}$, i.e., the constraint Jacobian with respect to $\vec{x}$, and $\vec{I} \in \mathbb{R}^{m\times m}$ is the identity matrix. Even when using quasi-Newton approximations, exploiting the structure of (14) can greatly improve the performance of the algorithm. Tests on analytic problems conducted by Arreckx et al. (2015) used a Hessian model $\vec{B}$ consisting of a limited-memory symmetric rank-one (LSR1) quasi-Newton approximation to $\nabla^2 F - \sum_{i=1}^m \lambda_i \nabla^2 C_i$ (the Hessian of the Lagrangian) and the true Jacobian-vector products to model $\vec{J}$. The other Hessian term, $\sum_{i=1}^m \left(\rho\left(C_i(\vec{x}) - t_i\right)\right)\nabla^2 C_i$, could be incorporated into the quasi-Newton approximation via a structured quasi-Newton method (Martínez, 1988) or truncated entirely because $C_i(\vec{x}) - t_i$ approaches zero as the algorithm proceeds. Unfortunately, while this algorithm is fast in terms of the number of trust-region iterations, it is costly in terms of the number of Jacobian-vector products because Moré and Toraldo's algorithm can require many matrix-vector products with $\vec{B}$ to solve a bound-constrained problem. Because of the bound constraints, the number of conjugate-gradient iterations needed to solve (13) — and, therefore, the number of matrix-vector products — is not necessarily bounded by the size of the $\vec{B}$ matrix. Therefore, to decouple the number of expensive matrix-vector products from the number of conjugate gradient iterations, we focus on approximations of (14) that incorporate approximations of $\vec{J}$.

We have devised two approximation strategies which we refer to as the split-quasi-Newton approach and the approximate-Jacobian approach. In the split-quasi-Newton approach, the augmented Lagrangian is broken up into Lagrangian and infeasibility functions, $\mathscr{L}$ and $\mathscr{I}$, and a separate quasi-Newton method is

used to approximate the Hessian of each function. In particular,

$$\begin{aligned} \Phi(\vec{x},\vec{t};\boldsymbol{\lambda},\rho) &= \mathscr{L}(\vec{x},\vec{t};\boldsymbol{\lambda}) + \rho\,\mathscr{I}(\vec{x},\vec{t}) \\ \mathscr{L}(\vec{x},\vec{t};\boldsymbol{\lambda}) &= F(\vec{x}) + \boldsymbol{\lambda}^T(\vec{C}(\vec{x}) - \vec{t}) \\ \mathscr{I}(\vec{x},\vec{t}) &= \frac{1}{2}(\vec{C}(\vec{x}) - \vec{t})^T(\vec{C}(\vec{x}) - \vec{t}). \end{aligned} \tag{15}$$

We use an LSR1 quasi-Newton approximation to $\nabla^2\mathscr{L}$ and a limited-memory Broyden–Fletcher–Goldfarb–Shanno
(LBFGS) quasi-Newton approximation to $\nabla^2\mathscr{I}$. These choices were made based on the fact that $\nabla^2\mathscr{I}$ is positive-semidefinite near a local minimum, while $\nabla^2\mathscr{L}$ could be indefinite. In terms of computational cost, only a single extra Jacobian-vector product is required per trust-region iteration to compute $\nabla\mathscr{I}$ for the second quasi-Newton approximation. On average, the per-iteration cost of this algorithm is one evaluation of the objective and all constraints, one evaluation of the objective function gradient, and four matrix-vector products with the constraint Jacobian regardless of the problem dimensions.

In the approximate-Jacobian approach, we directly approximate the Jacobian itself within (14) using a quasi-Newton method and truncate the term $\sum_{i=1}^{m}(\rho(C_i(\vec{x}) - t_i))\nabla^2 C_i$. Again, we use an LSR1 approximation to $\nabla^2\mathscr{L}$ for the Lagrangian Hessian. The Jacobian is approximated by a full-memory adjoint Broyden method (Schlenkrich et al., 2010). The adjoint Broyden update is given by

$$\vec{A}^{k+1} = \vec{A}^k + \frac{\boldsymbol{\sigma}^k\boldsymbol{\sigma}^{k,T}}{\boldsymbol{\sigma}^{k,T}\boldsymbol{\sigma}^k}\left(\vec{J}^{k+1} - \vec{A}^k\right), \tag{16}$$

where $\vec{A}^k \in \mathbb{R}^{m\times n}$ is the approximate Jacobian at iteration k and $\vec{J}^{k+1}$ is the true Jacobian at iteration $k+1$. The direction vector $\boldsymbol{\sigma}^k \in \mathbb{R}^m$ is chosen to be

$$\boldsymbol{\sigma}^k = (\vec{J}^{k+1} - \vec{A}^k)(\vec{x}^{k+1} - \vec{x}^k). \tag{17}$$

In contrast to quasi-Newton methods for square matrices, there are no limited-memory variants of the adjoint Broyden method with a convergence guarantee, so we use the full-memory version. To improve the computational performance of this method on large problems in a parallel computing environment, message passing interface (MPI) standard instructions are used to distribute the matrix approximation over multiple processors and form matrix-vector products with $\vec{A}$. The choice of $\boldsymbol{\sigma}$ given in (17) causes the adjoint Broyden update to require two Jacobian-vector products instead of one. Therefore, the per-iteration cost is higher than that of the split-quasi-Newton strategy by one Jacobian-vector product. However, compared to the split-quasi-Newton strategy, the approximate-Jacobian strategy can provide a better model of the structure of (14) because the identity term created by the slack variables is captured perfectly.

We close this section with a few implementation notes. Our trust-region algorithm uses three key enhancements to accelerate convergence. First, we use "magical steps" (Conn et al., 1999) to automatically update the slack variables to their best values given a particular set of design variables. Second, we use the backtracking procedure of Nocedal and Yuan (1998) in an effort to keep the trust region large. Finally, we use a nonmonotone strategy (Toint, 1997) to allow the optimizer to quickly navigate curved regions of the feasible domain. Our algorithm is written in Python and distributed as part of the NLPy package (Orban, 2014). The code is publicly available on Github (http://github.org/dpo/nlpy).

4 Computational Tools

Our matrix-free optimizer, AUGLAG, is benchmarked against the SQP optimizer SNOPT (Gill et al., 2002). Like AUGLAG, SNOPT uses a limited-memory quasi-Newton approximation to $\nabla^2\mathscr{L}$ to solve a given optimization problem. However, SNOPT requires the full constraint Jacobian to be computed at each iteration,

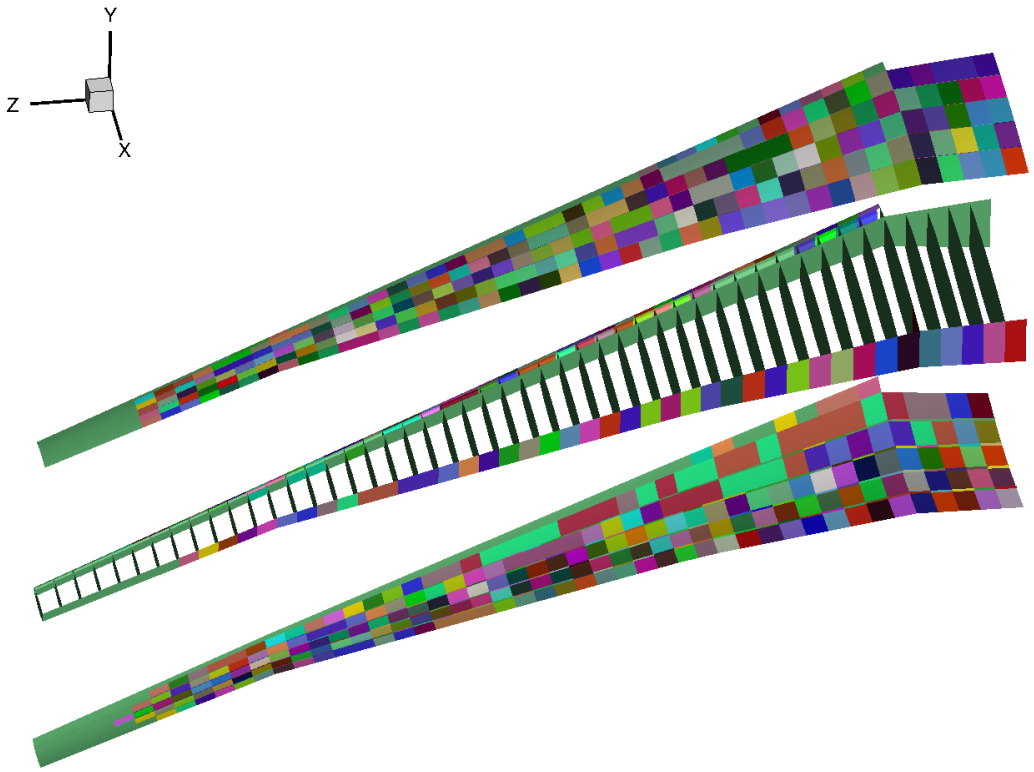

Figure 1: Exploded view of structure layout used in the test problems. The coloured patches represent groups of finite elements to which individual thickness variables and failure constraints are assigned.

while AUGLAG needs only Jacobian-vector products. While we expect to see a difference in the results due to the difference in optimization algorithms, our aim is to show that AUGLAG is still competitive with SNOPT due to the low cost of the trust-region iterations in AUGLAG.

The wing structure used in our test problems is analyzed using the Toolkit for the Analysis of Composite Structures (TACS) (Kennedy and Martins, 2014a), a finite-element analysis code. The wing aerodynamics were analyzed using the three-dimensional panel code TriPan (Kennedy and Martins, 2014b). All analysis and optimization codes were accessed through the MACH framework (MDO of aircraft configurations at high fidelity) (Kenway et al., 2014b). The MACH framework includes modules for aerostructural analysis and geometry warping.

Prior to this project, both the TACS and TriPan codes and the MACH framework possessed modules for efficiently computing derivatives using the adjoint method. However, in order to enable the matrix-free approach to optimization, modules were added to compute forward and transpose matrix-vector products with the partial derivative matrices shown in (5). We expect other researchers interested in using a matrix-free optimizer would need to undertake similar modifications to their solvers. However, if the direct and adjoint methods are already available, the implementation is relatively straightforward. In terms of performance, the cost of a single Jacobian-vector product should roughly equal the cost of one linear solve using the direct method. Similarly, the cost of a single transpose Jacobian-vector product should rougly equal the cost of one linear solve using the adjoint method.

5 Structural Optimization

The first problem we study is the mass minimization of a wing box subject to failure constraints. The outer

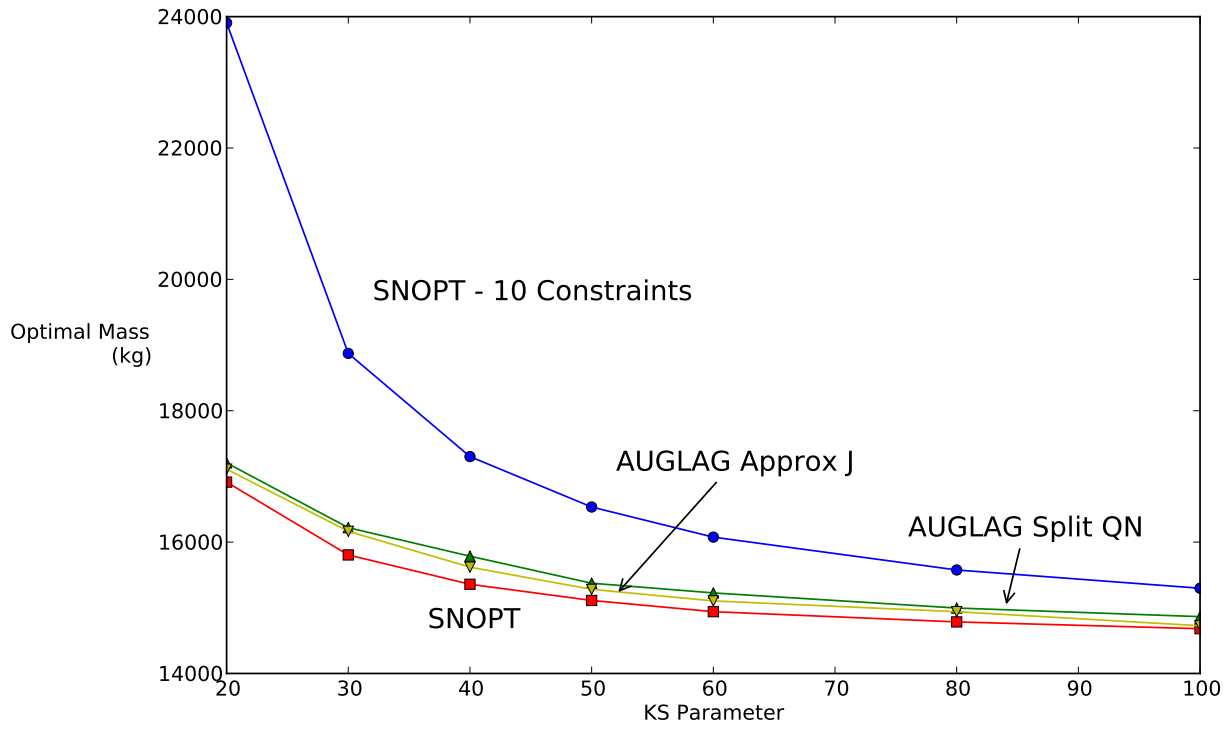

Figure 2: Optimal masses of the wing design computed by each optimizer for different values of ρ_{KS}. Both optimizers found better optimal masses using 2832 constraints than using 10 KS constraints. Despite the different convergence tolerances used, the spread between optimizers is less than 3% of the best mass computed for each KS parameter value. At the largest parameter values, the difference is 1%.

wing geometry is based on the Boeing 777-200ER civil transport. Figure 1 shows the structural layout used. The wing is 30.5 m from root to tip and uses the RAE 2822 transonic airfoil as the basic shape. Because this problem ignores multidisciplinary effects, the mass of the aircraft is assumed to be a constant 298 000 kg even though the mass of the wing structure is allowed to vary. The constant aircraft mass is used to determine aerodynamic loads for two load cases: a 2.5g pull-up maneuver and a 1g push-over maneuver. Both cases are assumed to be symmetric, so symmetry boundary conditions are enforced in both the aerodynamic and structural analyses. The loads are computed assuming that the aircraft is flying at 10 000 ft altitude and 0.84 Mach number. Atmospheric properties for the load calculation are determined using the US Standard Atmosphere.

The wing model itself is composed of third-order MITC shell elements on all surfaces. Skin stiffeners on both the top and bottom skins are modelled explicitly and have a fixed height. In all, the wing contains nearly 46 000 finite elements and 250 000 degrees of freedom. Aluminium alloy 2024 is selected as the material for the whole wing structure.

Design variables and failure constraints are assigned to the wing by subdividing the different structural components into patches. Figure 1 illustrates these patches by the colour scheme. Each patch contains a group of finite elements that share a thickness design variable and two failure constraints, one per load case. Failure is determined based on the local Von Mises stress exceeding a yield stress value. Note that we still use KS aggregation to compute the failure criterion, but the aggregation only takes place over the element patch. In all, the mass minimization problem contains 1416 variables and 2832 constraints.

The optimal designs are computed using both SNOPT and AUGLAG for a variety of ρ_{KS} values. Fig-

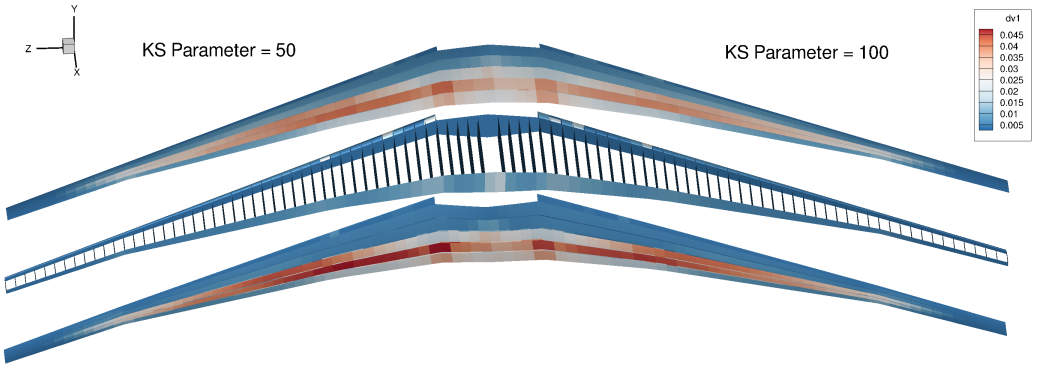

Figure 3: Optimal wing structures generated by SNOPT for $\rho_{KS} = 50$ and $\rho_{KS} = 100$. The structure generated by setting $\rho_{KS} = 100$ is noticeably thinner in the middle part of the wing skin on both top and bottom surfaces.

ure 2 plots the optimal mass computed by each optimizer for each parameter value to verify that both optimizers converge on similar solutions. For SNOPT the feasibility and optimality tolerances were set to 10^{-5}, while for AUGLAG the tolerances were set to 10^{-4} for the approximate-Jacobian version and 3×10^{-4} for the split-quasi-Newton version. We found that AUGLAG had difficulty converging to the same solution tolerance as SNOPT. We attribute this issue to the difference in optimization algorithms as the augmented Lagrangian method cannot update the Lagrange multipliers as frequently as an SQP algorithm and is known to converge more slowly as a result. Conn et al. (1991) report only an R-linear rate of local convergence, while many SQP algorithms achieve a superlinear rate of convergence (Nocedal and Wright, 2006, Section 18.7). Nevertheless, if the tolerance is sufficiently tight, Figure 2 demonstrates that both algorithms can converge on similar optimal designs.

Because we have sought to reduce the amount of constraint aggregation used in our test problem, the reader may wonder how much of a difference reintroducing aggregation would make to the optimal solution. This issue merits a more thorough discussion than we can provide here. Our goal in this work is to test the viability of the matrix-free optimizer, not determine an optimal aggregation strategy for the problem. However, Figure 2 provides a partial answer. We solved the same mass minimization problem in SNOPT using only 10 KS constraints, instead of the 2832 KS constraints used in all other cases. Figure 2 shows that using fewer constraints leads to much greater overestimation of the optimal wing mass. For $\rho_{KS} = 100$, the overestimation is 5% of the best mass found. For $\rho_{KS} = 20$, the optimum mass is overestimated by more than 50%.

Figures 3 and 4 show the effect of changing ρ_{KS} from 50 to 100 on the optimal structure itself. Figure 3 shows the thickness of each structural element while Figure 4 shows the corresponding stress distribution for the 2.5g load case. In this problem, the -1g load case did not actively constrain the optimal design. Both figures were generated from the SNOPT optimization results. The larger ρ_{KS} value produces a thinner structure in the middle part of the wing skin and this thinner structure leads to a more fully-stressed wing design. The higher stresses shown in Figure 4 are indicative of a more efficient structural design for the prescribed load.

The difference between the optima found by SNOPT and AUGLAG for a given choice of ρ_{KS} lies in the distribution of the material in the wing skins. Figure 5 shows the difference between the thickness in the wings produced by SNOPT and AUGLAG Split QN for $\rho_{KS} = 50$. From Figure 2, the difference in mass between the wings is 2%. However, the difference in the convergence tolerance leads to a different mass distribution. The wing produced by AUGLAG Split QN has thicker skins near both the leading and trailing edges of the wing and thinner skins mid-chord than the wing produced by SNOPT.

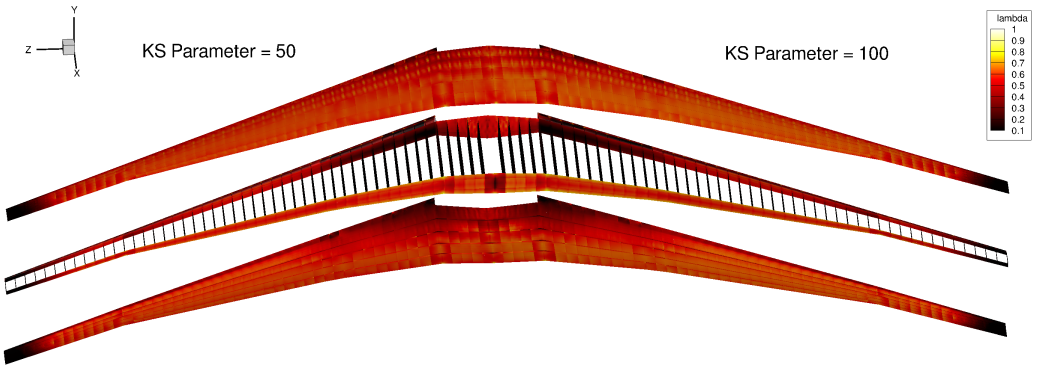

Figure 4: Stress distributions induced by the 2.5g maneuver condition on the optimal wing structures shown in Figure 3. The structure generated by setting $\rho_{KS} = 100$ is more fully-stressed, indicating a more efficient structure.

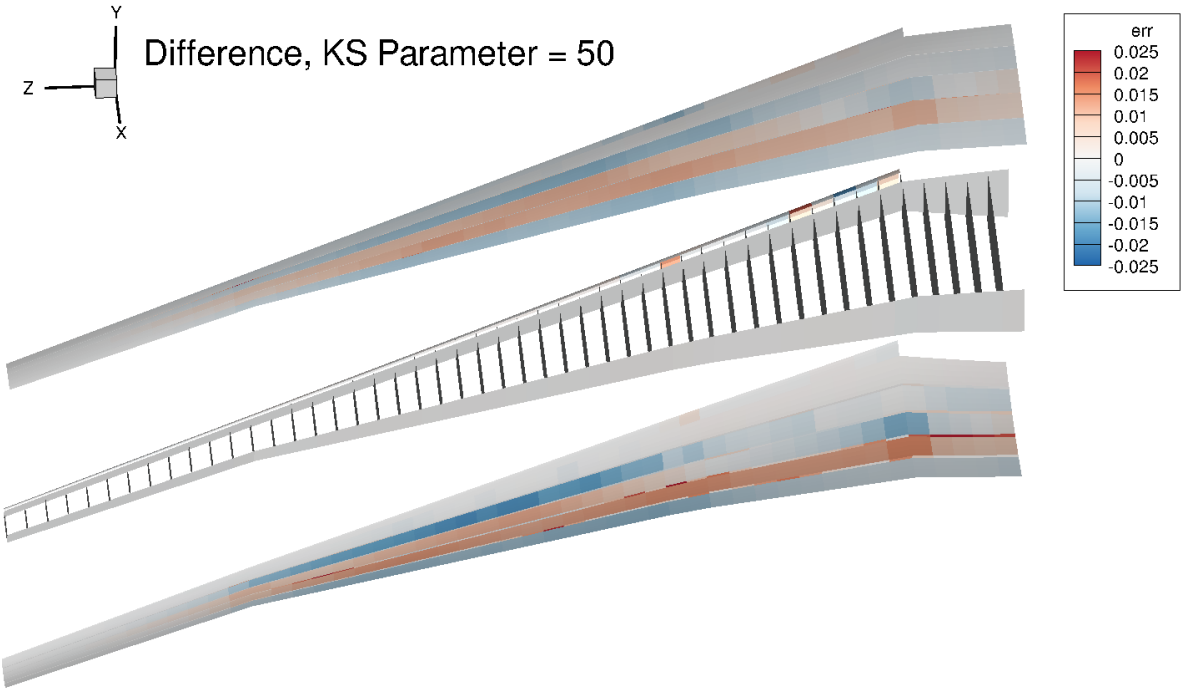

Figure 5: Difference in structural thickness between SNOPT optimum and AUGLAG-SplitQN optimum for $\rho_{KS} = 50$. Positive values indicate where the wing produced by SNOPT is thicker.

Table 1: Average run times for specific computations in the wing structure optimization problem

Computation	**Wall Time, 32 proc.**
Objective and 2832 constraints	2.31 s
Jacobian of 2832 constraints	188.50 s
Objective gradient only	0.02 s
Jacobian product with 2832 cons.	0.40 s
Transpose Jacobian product with 2832 cons.	0.33 s

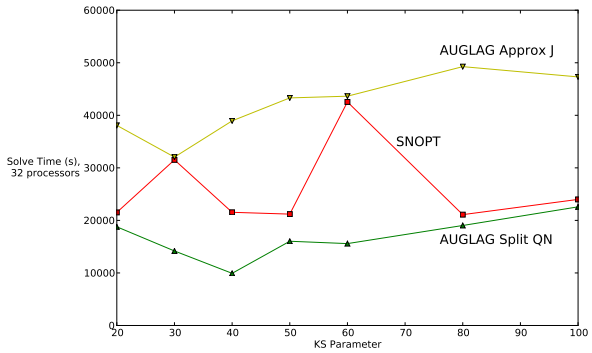

Figure 6: Run time to solve the wing structure optimization problem for a range of ρ_{KS} values

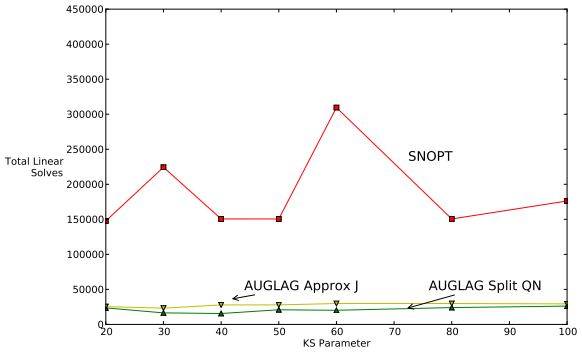

Figure 7: Number of linear solve operations to solve the wing structure optimization problem for a range of ρ_{KS} values

The main focus of this study is the computational cost of obtaining optimal designs. Figures 6 and 7 show the cost of the optimization in terms of both run time (using 32 processors in parallel) and the number of linear solve operations. The latter metric treats one matrix-vector product as equivalent in cost to forming a single row or column of the constraint Jacobian. A single linear solve operation is also needed in the evaluation of the constraints for this problem. Because the linear solve is the most expensive operation in computing function and gradient information for the problem, minimizing the number of these operations reduces the computational cost. Figure 7 shows that both versions of AUGLAG are far more efficient than SNOPT at optimizing the wing design, in terms of the number of linear solves, for a range of parameter values. Reductions in the number of linear solves can be up to an order of magnitude.

Despite the strong reduction in the number of linear solves, Figure 6 shows that, in terms of run time, AUGLAG shows little improvement over SNOPT, despite the relaxed convergence tolerance used. Only the split-quasi-Newton version of AUGLAG is competitive across a range of parameter values when run time is considered. The explanation for this discrepancy lies in Table 1, which displays the computational time required by key operations in the optimization process. Despite the fact that the problem contains more than a thousand variables and constraints, the time required to evaluate the entire Jacobian is only 90 times greater than the time required to evaluate the constraints. This result stems from the efficiency of the adjoint method implementation in the TACS structural analysis. (TACS uses a specialized sparse direct factorization method to solve the linear system and can reuse the matrix factors for multiple right-hand sides at the same design point.) The assumption that all linear solve operations require roughly equal time is violated in this problem. We remark, though, that these results are not typical if the gradient computation is less efficient.

The variation in the results with the increasing ρ_{KS} value shown in Figures 6 and 7 raises the question of whether or not this variation is caused by the change in the design space with varying ρ_{KS} for the same starting point. The starting point used in the first set of tests is a constant thickness in all elements. Figures 8 and 9 show results for SNOPT and the split-quasi-Newton version of AUGLAG using sets of randomly-

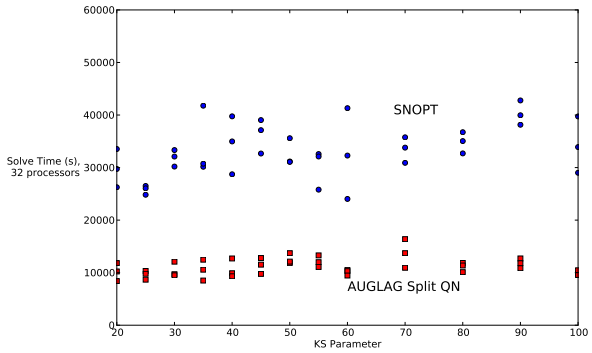

Figure 8: Run time to solve the wing structure optimization problem from a random starting point

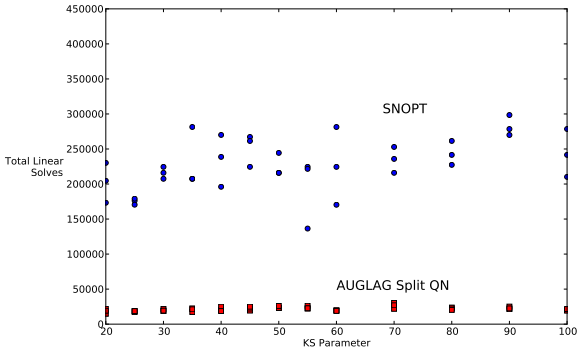

Figure 9: Number of linear solve operations to solve the wing structure optimization problem from a random starting point

generated thickness values as the starting points. The initial thickness values were generated by selecting normally-distributed random values with a mean and standard deviation of the distribution equal to the midpoint of the thickness variable bounds. Random values computed outside the bound range were redefined to the end points of the range. The data in Figures 8 and 9 fall into specific ranges depending on the optimizer. In all cases, the optimal solution for the wing mass was similar to the corresponding values provided by Figure 2. The similarity in the figures suggests that the solve time is strongly correlated with the number of linear solves, which is driven by the number of optimizer iterations. Therefore, the variation is most likely driven by small changes in the optimizer search path, both in the search direction used for the line search and the length of the step itself. The variation is not a specific property of the starting point or optimization problem.

6 Aerostructural Optimization

By coupling the aerodynamic and structural states, we create an MDO problem. The wing geometry and model are the same as those used in Section 5. However, the multidisciplinary wing design problem is of particular interest because the computation of both the constraints and their gradients is more complicated than the single-discipline case. The linear systems generated and solved in this problem include interdisciplinary coupling terms that are never explicitly formed as matrices. We are forced to use iterative, i.e., matrix-free, methods to solve these linear systems and compute the constraints and Jacobian matrix. Since we are unable to form and factorize the $\partial R/\partial y$ matrix in (5), the computational cost of forming the Jacobian is greatly increased. A matrix-free optimizer becomes the natural choice to reduce the computational cost of finding an optimal design.

The objective of this MDO problem is to minimize the take-off gross weight (TOGW) of the aircraft for a 7725-nautical-mile design mission. TOGW includes the weight of the wing structure, the weight of the fuel burned, and fixed weights representing the payload and the rest of the aircraft structure. The total fuel burned is computed by the Breguet range equation, assuming a constant lift-to-drag ratio of the wing for the duration of the mission, i.e.,

$$W_{FB} = W_L \left(\exp\left(\frac{R c_T g}{V} \frac{D}{L} \right) - 1 \right). \tag{18}$$

In equation (18), R is the design range of the aircraft, W_{FB} is the weight of fuel burned, W_L is the landing weight, V is the cruise speed, c_T is the thrust-specific fuel consumption, g is the gravitational constant, and L/D is the lift-to-drag ratio. W_L is the sum of payload, non-wing aircraft weight, reserve fuel, non-structural wing weight, and structural wing weight. In contrast with the structural design problem of Section 5, the

Table 2: Aircraft specifications from (Boe, 2011; Kenway et al., 2014a)

Aircraft Data	**Value**
Gross take-off mass (max)	298 000 kg
Non-wing empty aircraft mass	107 700 kg
Payload mass	34 000 kg
Reserve fuel mass	15 000 kg
Cruise altitude	35 000 ft
Maneuver condition altitude	10 000 ft
Fuselage length	63.7 m
Fuselage diameter	6.19 m
Design Range	7725 nmi
Maneuver condition load factors	-1.0g, 2.5g
Fuel fraction at cruise condition	0.20
Fuel fraction at maneuver conditions	1.00
Non-structural wing mass	25% of structural wing mass

varying weight of the wing structure is incorporated into the TOGW calculation as part of the aircraft landing weight. Table 2 provides the additional quantities needed to define this problem.

In addition to designing the structure, the optimizer is also given the ability to change the twist of the wing to alleviate high structural loads. The jig twist angle is defined at five stations on the wing (10.2%, 30%, 34%, 75%, and 100% of the half-span) and interpolated using a geometry warping scheme based on free-form deformation volumes (Kenway et al., 2014b). The changing aerodynamic loads are mapped to the structure using a system of rigid links between the aerodynamic and structural meshes (Kenway et al., 2014b). The optimizer is also allowed to select the angle-of-attack of the wing at each flight condition via three additional angle-of-attack design variables. The addition of aerodynamic and geometry variables gives the optimizer the some ability to control lift-to-drag ratio at each flight condition.

Our optimization problem now contains three flight conditions: the cruise condition and the two maneuver conditions previously outlined for the structural optimization problem. Structural failure constraints are enforced at all three flight conditions. We also introduce a constraint for each flight condition that the lift generated by the wing must equal the total aircraft weight. In order to generate a realistic lift-to-drag ratio at cruise, we introduce a fuselage drag component to add to the wing drag computed by TriPan. The fuselage drag is computed using the fuselage dimensions provided in Table 2 and the formulae provided by Roskam (1998, Volume 6). Atmospheric properties are determined using the US Standard Atmosphere at the altitudes provided in Table 2.

The resulting optimization problem now contains 1424 design variables and 4251 constraints. The size of the problem, coupled with its multidiscplinary nature, makes computation of the Jacobian prohibitively expensive. Table 3 displays the run time required to compute the objective and constraints, the Jacobian, and Jacobian-vector products. Using 32 processors, computing the full Jacobian would take about 7.5 hours. Under this time constraint, an optimizer like SNOPT would only be able to complete five iterations in a typical two-day high-performance computing job. Using our structural optimization problem as a benchmark, we estimate the aerostructural problem would take around 50 iterations to converge. The run time to solve the problem would total to nearly 16 days.

In contrast to SNOPT and other traditional gradient-based optimizers, a matrix-free optimizer should be very effective at solving this problem. Unlike the structural design problem, Table 3 shows that the difference in run time between evaluating the objective and constraints and computing the Jacobian is a factor of 1600. Given that our problem has thousands of variables and constraints, this value more closely

Table 3: Average run times for specific computations in the aerostructural optimization problem

Computation	**Wall Time, 32 proc.**
Objective and 4251 constraints	16.82 s
Jacobian of 4251 constraints	26 926.00 s
Objective gradient only	3.86 s
Jacobian product with 4251 cons.	14.36 s
Transpose Jacobian product with 4251 cons.	8.15 s

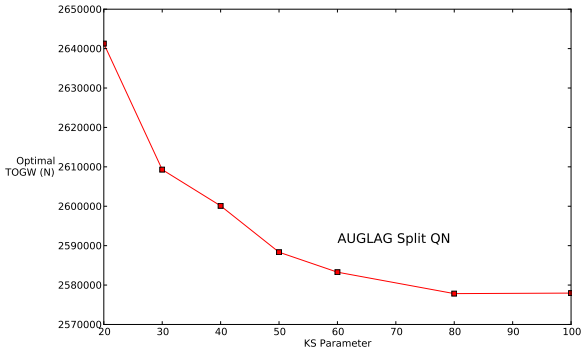

Figure 10: Optimal aircraft TOGW values for the aerostructural optimization problem for a range of ρ_{KS} values. The variation in TOGW for $\rho_{KS} > 50$ is less than 1%.

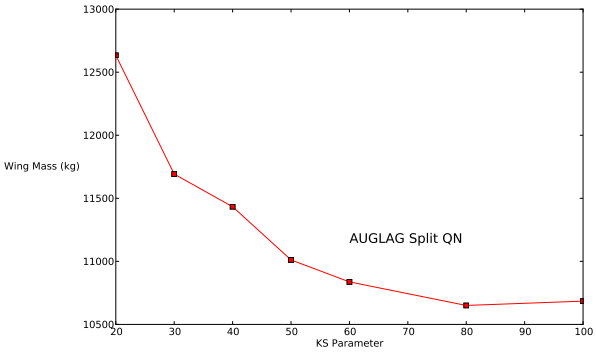

Figure 11: Optimal wing structure masses computed in the aerostructural optimization problem for a range of ρ_{KS} values. The reduction in wing mass as ρ_{KS} increases is similar to that observed in Figure 2 in the structural problem results.

aligns with our assumption that the number of large linear solve operations provides a reliable estimate of computational cost. Furthermore, given the success of AUGLAG in reducing the number of linear solves relative to SNOPT, as shown in Figure 7, we expect AUGLAG to perform particularly well on this problem.

Before discussing the computational effort to solve the problem, we briefly comment on the optimal designs themselves. Figure 10 shows the optimal TOGW computed by AUGLAG for each value of ρ_{KS}. While there is some variation in the optimal TOGW, the difference decays rapidly as ρ_{KS} increases. For ρ_{KS} values greater than 50, the difference in TOGW is less than 1%. Much greater variation exists if we only examine the wing structure itself. Figure 11 plots the mass of the wing structure at the optimal solution. Note the difference in the wing mass between this problem and the structural problem in Figure 2. However, in terms of variation in the optimum relative to the best mass, both Figure 2 and Figure 11 show a 4% drop in mass between the $\rho_{KS} = 50$ case and the $\rho_{KS} = 100$ case.

Figures 12 and 13 show the structural thickness and the stress distribution for the 2.5g load case. As with the structural design problem, using a larger ρ_{KS} parameter value resulted in a lighter structural design and higher stress under load. Compared to Figure 3, Figure 12 shows a much thinner wing skin at the optimal solution. This lighter structural design has two major causes. First, in the aerostructural problem, the optimizer is allowed to twist the wing to alleviate some of the structural load at the constraining load case. Second, as noted in Figure 10, the weight of the whole aircraft in the aerostructural problem is lower than that used in the structural problem. Because the optimizer can explicitly estimate the fuel required to complete the design mission in this version, the aircraft mass is allowed to decrease in this problem.

One aspect of the optimal design that is not greatly affected by the choice of ρ_{KS} is the shape of the deflected wing. Figure 14 compares the wing shape under the 2.5g load condition for $\rho_{KS} = 50$ and $\rho_{KS} = 100$. The difference in the wing tip deflection between the designs is approximately 10 cm, less than the

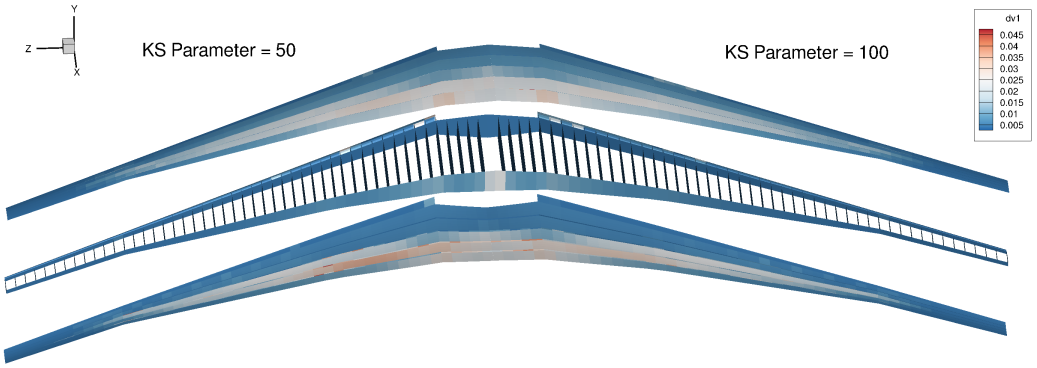

Figure 12: Optimal wing structures generated by AUGLAG for ρ_{KS} values of 50 and 100 for the aerostructural design problem. The wing structures are substantially thinner than the results given by Figure 3 because the estimated mass of the full aircraft is much lower, reducing the peak structural load.

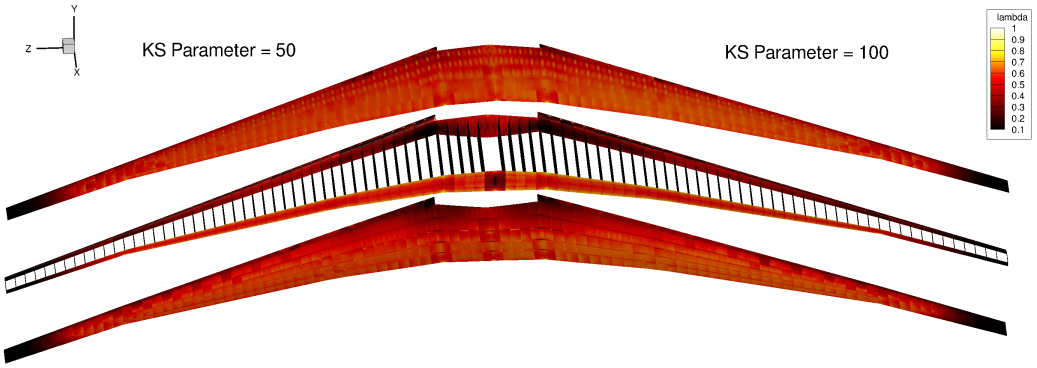

Figure 13: Stress distributions induced by the 2.5g maneuver condition on the optimal wing structures shown in Figure 12. As with the structural design problem, the structure generated by setting $\rho_{KS} = 100$ is more fully-stressed.

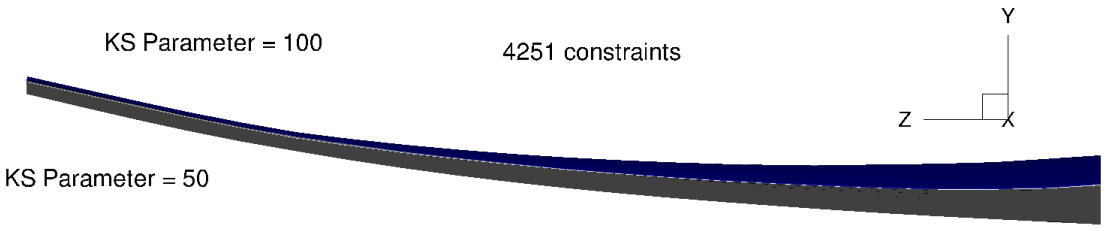

Figure 14: Deflected wing shapes under the 2.5g maneuver condition for the optimal designs using $\rho_{KS} = 50$ (bottom) and $\rho_{KS} = 100$ (top). The difference in the deflection at the tip is about 10cm, less than the thickness of the wing itself.

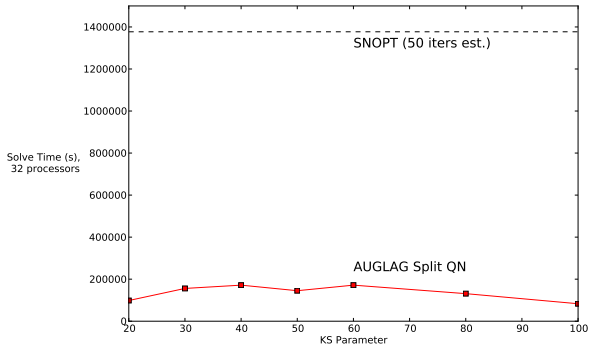

Figure 15: Run time to solve the aerostructural optimization problem for a range of ρ_{KS} values

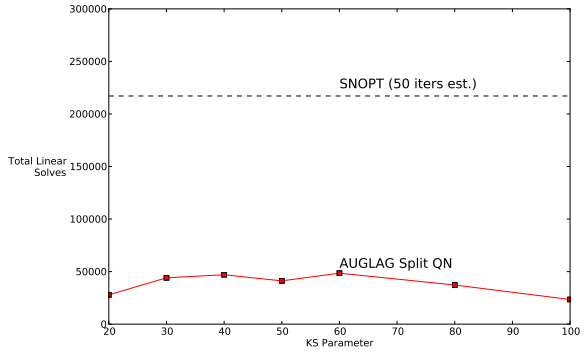

Figure 16: Number of linear solve operations to solve the aerostructural optimization problem for a range of ρ_{KS} values

Table 4: Computational resources to solve the aerostructural optimization problem. Data for AUGLAG is averaged over seven values of ρ_{KS}.

Optimizer	**Number of Linear Solves**	**Wall Time, 32 proc.**
SNOPT (50 iteration estimate)	217107	382.5 hr
AUGLAG Split QN (average)	38481	38.0 hr

thickness of the wing at the tip. This small change is due to the fact that, as we noted above, the weight of the optimized aircraft and, consequently, the load on the wing differ by less than 1% between the two design cases.

Using the split-quasi-Newton version of AUGLAG and a convergence tolerance of 3×10^{-4}, we are able to compute optimal wing designs within the two-day limit of a typical high-performance computing job. Table 4 shows the run time and number of linear solves required by AUGLAG, averaged over seven ρ_{KS} values, compared with our estimate of the resources required for SNOPT to solve the same problem. The data is shown in more detail in Figures 15 and 16. As with the structural example, the results seem insensitive to changes in the ρ_{KS} value over the tested range and a small amount of oscillation is observed. In terms of both the number of linear solves and run time, AUGLAG is up to an order of magnitude faster than the estimated results of SNOPT.

7 Conclusion

We have presented results for the optimization of an aircraft wing using a new matrix-free optimizer. The wing designs consider only a structural analysis or a coupled aerodynamic and structural analysis. We deliberately avoided aggressive constraint aggregation to formulate optimization problems with both thousands of variables and thousands of constraints. Our matrix-free optimizer is capable of solving these large optimization problems much more quickly than a traditional optimizer. If the constraint Jacobian is particularly expensive to form, the total computational effort can be reduced by up to an order of magnitude.

The positive results described in this paper point to the strong potential of matrix-free methods in structural and multidisciplinary optimization. Because we used an augmented Lagrangian algorithm rather than SQP, we would expect a suitable matrix-free SQP algorithm to show even better performance solving the problems we described. Regarding our application, the size of the structural model and our decision to use a panel code instead of full computational fluid dynamics kept the computational cost of each multidisciplinary analysis down. Higher-fidelity models would increase computational cost and would require even

more intensive use of high-performance computing. Under these conditions, gradient and Jacobian computation becomes even more expensive and matrix-free optimization algorithms become even more important for problems with many variables and constraints.

Acknowledgments

Computations were performed on the GPC supercomputer at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; and the University of Toronto.

References

(2011) 777-200/200ER/300 airplane characteristics for airport planning. Tech. rep., Boeing Commercial Airplanes, Seattle, WA

Akgün MA, Haftka RT, Wu KC, Walsh JL, Garcelon JH (2001) Efficient Structural Optimization for Multiple Load Cases Using Adjoint Sensitivities. AIAA Journal 39(3):511–516, doi:10.2514/2.1336

Arreckx S, Lambe A, Martins JRRA, Orban D (2015) A matrix-free augmented Lagrangian algorithm with application to large-scale structural design optimization. Optimization and Engineering In press.

Bertsekas DP (1996) Constrained Optimization and Lagrange Multiplier Methods. Athena Scientific

Bosse T (2009) A Derivative-matrix-free NLP Solver Without Explicit Nullspace Representation. Phd, Humboldt-Universitat zu Berlin

Conn AR, Gould NIM, Toint PL (1991) A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds. SIAM Journal on Numerical Analysis 28(2):545–572

Conn AR, Gould NIM, Toint PL (1992) LANCELOT: a Fortran package for large-scale nonlinear optimization (release A). Springer-Verlag, Berlin

Conn AR, Vicente LN, Visweswariah C (1999) Two-step Algorithms for Nonlinear Optimization with Structured Applications. SIAM Journal on Optimization 9(4):924–947

Curtis FE, Nocedal J, Wächter A (2009) A Matrix-Free Algorithm for Equality Constrained Optimization Problems with Rank-Deficient Jacobians. SIAM Journal on Optimization 20(3):1224–1249, doi:10.1137/08072471X

Gill PE, Murray W, Saunders MA (2002) SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization. SIAM Journal on Optimization 12(4):979–1006

Gondzio J (2012) Matrix-free interior point method. Computational Optimization and Applications 51:457–480, doi:10.1007/s10589-010-9361-3, URL `http://link.springer.com/10.1007/s10589-010-9361-3`

Griewank A, Walther A (2002) On Constrained Optimization by Adjoint based Quasi-Newton Methods. Optimization Methods and Software 17:869–889, doi:10.1080/1055678021000060829

Haftka RT, Kamat MP (1989) Simultaneous Nonlinear Structural Analysis and Design. Computational Mechanics 4:409–416, doi:10.1007/BF00293046

Heinkenschloss M, Vicente LN (1999) An Interface between Optimization and Application for the Numerical Solution of Optimal Control Problems. ACM Transactions on Mathematical Software 25(2):157–190

Kennedy GJ (2015) Strategies for adaptive optimization with aggregation constraints using interior-point methods. Computers & Structures 153:217–229, doi:10.1016/j.compstruc.2015.02.024, URL http://linkinghub.elsevier.com/retrieve/pii/S0045794915000620

Kennedy GJ, Hicken JE (2015) Improved constraint-aggregation methods. Computer Methods in Applied Mechanics and Engineering 289:332–354, doi:10.1016/j.cma.2015.02.017, URL http://linkinghub.elsevier.com/retrieve/pii/S0045782515000663

Kennedy GJ, Martins JRRA (2014a) A parallel finite-element framework for large-scale gradient-based design optimization of high-performance structures. Finite Elements in Analysis and Design 87:56–73, doi:10.1016/j.finel.2014.04.011, URL http://linkinghub.elsevier.com/retrieve/pii/S0168874X14000730

Kennedy GJ, Martins JRRA (2014b) A parallel aerostructural optimization framework for aircraft design studies. Structural and Multidisciplinary Optimization 50(6):1079–1101, doi:10.1007/s00158-014-1108-9

Kenway GKW, Kennedy GJ, Martins JRRA (2014a) Aerostructural optimization of the Common Research Model configuration. 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference pp 1–19, doi:10.2514/6.2014-3274, URL http://arc.aiaa.org/doi/abs/10.2514/6.2014-3274

Kenway GKW, Kennedy GJ, Martins JRRA (2014b) Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and derivative computations. AIAA Journal 52(5):935–951, doi:10.2514/1.J052255

Kreisselmeier G, Steinhauser R (1979) Systematic Control Design by Optimizing a Vector Performance Indicator. In: Symposium on Computer-Aided Design of Control Systems, IFAC, Zurich, Switzerland, pp 113–117

Lambe AB, Martins JRRA (2015) Structural and aerostructural design of aircraft wings with a matrix-free optimizer. In: 11th World Congress on Structural and Multidisciplinary Optimization, Sydney, Australia

Martínez HJ (1988) Local and Superlinear Convergence of Structured Secant Methods from the Convex Class. Tech. rep., Rice University, Houston, TX

Martins JRRA, Hwang JT (2013) Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models. AIAA Journal 51(11):2582–2599, doi:10.2514/1.J052184, URL http://arc.aiaa.org/doi/abs/10.2514/1.J052184

Martins JRRA, Lambe AB (2013) Multidisciplinary design optimization: A survey of architectures. AIAA Journal 51(9):2049–2075, doi:10.2514/1.J051895

Martins JRRA, Sturdza P, Alonso JJ (2003) The Complex-Step Derivative Approximation. ACM Transactions on Mathematical Software 29(3):245–262, doi:10.1145/838250.838251, URL http://doi.acm.org/10.1145/838250.838251

Martins JRRA, Alonso JJ, Reuther JJ (2005) A Coupled-Adjoint Sensitivity Analysis Method for High-Fidelity Aero-Structural Design. Optimization and Engineering 6(1):33–62, doi:10.1023/B:OPTE.0000048536.47956.62

Moré JJ, Toraldo G (1991) On the Solution of Large Quadratic Programming Problems with Bound Constraints. SIAM Journal on Optimization 1(1):93–113

Nocedal J, Wright SJ (2006) Numerical Optimization, 2nd edn. Springer-Verlag

Nocedal J, Yuan Y (1998) Combining trust region and line search techniques. In: Advances in Nonlinear Programming, chap 7, pp 153–175

Orban D (2014) NLPy - a large-scale optimization toolkit in Python. Tech. rep., GERAD

Poon NMK, Martins JRRA (2007) An adaptive approach to constraint aggregation using adjoint sensitivity analysis. Structural and Multidisciplinary Optimization 34:61–73, doi:10.1007/s00158-006-0061-7

Raspanti CG, Bandoni JA, Biegler LT (2000) New strategies for flexibility analysis and design under uncertainty. Computers & Chemical Engineering 24:2193–2209, doi:10.1016/S0098-1354(00)00591-3, URL `http://linkinghub.elsevier.com/retrieve/pii/S0098135400005913`

Roskam J (1998) Airplane Design, vol 1-8, 2nd edn. DARCorporation, Ottawa, KS

Schlenkrich S, Griewank A, Walther A (2010) On the local convergence of adjoint Broyden methods. Mathematical Programming 121:221–247, doi:10.1007/s10107-008-0232-y

Sobieszczanski-Sobieski J (1990) Sensitivity of Complex, Internally Coupled Systems. AIAA Journal 28(1):153–160, doi:10.2514/3.10366

Steihaug T (1983) Local and Superlinear Convergence for Truncated Iterated Projections Methods. Mathematical Programming 27:176–190

Toint PL (1997) Non-monotone trust-region algorithms for nonlinear optimization subject to convex constraints. Mathematical Programming 77(3):69–94, doi:10.1007/BF02614518, URL `http://link.springer.com/10.1007/BF02614518`