A MODULAR MATRIX-FREE APPROACH TO MULTIDISCIPLINARY DESIGN OPTIMIZATION

By

Alp Dener

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: AERONAUTICAL ENGINEERING

Approved by the
Examining Committee:

Jason E. Hicken, Thesis Adviser

Assad Oberai, Member

Onkar Sahni, Member

John E. Mitchell, Member

Rensselaer Polytechnic Institute
Troy, New York

November 2017
(For Graduation December 2017)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

First and foremost, a special thanks go out to my thesis advisor, Jason Hicken. It was perhaps the greatest stroke of luck in my professional life to have started my doctoral studies at RPI the same year he started his own career as an Assistant Professor. I could not have asked for a better mentor, teacher and research partner than he has been for me throughout the last five years, and I am forever grateful for his willingness to take me on as his first doctoral student.

I would like to thank Joaquim Martins for granting access to his research group's high-fidelity flow and structural solvers, and Gaetan Kenway for providing extensive technical support. I also greatly appreciate Justin Gray's generosity in sharing his software design expertise throughout our collaboration.

The hard work and friendship of my colleagues at the Optimal Design Lab, Pengfei Meng, Anthony Ashley, Kinshuk Panda, Jared Crean, and Jianfeng Yan, were always a valuable source of motivation. I am thankful for the great work environment they have helped create.

I would also like to thank Onkar Sahni, Assad Oberai and John Mitchell for kindly agreeing to be on my doctoral committee and reviewing my work.

Attending RPI would not have been possible without the support of the National Science Foundation under grant number 1332819. Additionally, numerical experiments in this thesis were made possible by the exceptional computing resources offered by the Scientific Computing Research Center and the Center for Computational Innovations at RPI.

Finally, I would like to extend my eternal love and gratitude to my family; in particular my mom and dad, who have instilled in me a great appreciation for science and education, and have relentlessly pushed me to realize my full potential. I would not be who I am and where I am today without the sacrifices they have made to provide me with the greatest opportunities in life.

ABSTRACT

The individual discipline feasible (IDF) formulation is a multidisciplinary design optimization (MDO) architecture that provides modularity for the underlying discipline solvers. Similar to reduced-space methods, the IDF formulation does not require the optimization algorithm to converge the state variables for each discipline in addition to the optimization variables; the state equations are still solved fully at each optimization iteration. However, IDF decouples the discipline equations from each other through the introduction of coupling variables and constraints to the optimization problem. Consequently, the discipline solutions at each optimization iteration can be performed independently and in parallel. This promotes the use of existing discipline-specific PDE solvers, and lowers the software development challenge of creating efficient coupled discipline analyses.

Despite its advantages in modularity, the IDF architecture has remained largely unused by researchers and practitioners alike since its introduction in the early 1990s. The addition of large numbers of variables and constraints into the optimization problem proved to be challenging for conventional gradient-based optimization approaches. In particular, the explicit constraint Jacobian required by these algorithms is prohibitively expensive to compute for IDF problems.

In this thesis, we propose a reduced-space inexact-Newton-Krylov (RSNK) algorithm that can address the challenges posed by the IDF formulation. RSNK achieves this with three key components: a matrix-free formulation that avoids explicit Jacobians and Hessians, a new Krylov solver tailored for nonconvex saddle-point problems, and a novel matrix-free preconditioner for the IDF architecture. We implement RSNK in a parallel-agnostic optimization library, and verify its efficacy on a range of low- and high-fidelity test problems drawn from aerospace applications. Our findings demonstrate that RSNK scales favorably with the size of the design space, exhibits superlinear asymptotic convergence, and can efficiently solve large-scale PDE-governed MDO problems.

CHAPTER 1
INTRODUCTION AND CONTRIBUTIONS

The advent of high-performance computing has enabled researchers in both industry and academia to create predictive computational models for a variety of physical phenomena. The availability of these tools has made simulation-based design an indispensable tool for the development of new technologies and products by helping engineers efficiently evaluate more prototypes than would be possible with physical experimentation alone.

In the present work, we focus on engineering systems that involve complex interactions between two or more physical disciplines or subsystems. Examples include compliant lift surfaces that experience significant deformation under aerodynamic loads, or boundary-layer ingestion where changes to the airframe have a significant impact on air flow at the engine intake. In these situations, single-discipline analysis often fails to capture the phenomena of interest, and high-fidelity, coupled partial-differential-equations (PDEs) are necessary to accurately capture the governing physics. We believe that multidisciplinary design optimization (MDO), which develops and applies numerical optimization techniques to such problems, can help engineers identify and exploit complex trade-offs that may otherwise remain hidden.

One of the critical steps in performing multidisciplinary design optimization (MDO) is to determine the structure and organization of the problem. Decisions on how to couple the different disciplines, and how to solve the coupled problem, ultimately impact the size and characteristics of the optimization problem, which in turn influence the choice of optimization algorithm. These problem formulations are referred to as MDO "architectures" or "formulations" in literature, and fall under one of two categories: monolithic or distributed. Monolithic MDO architectures are single stand-alone optimization problems that encompass all variables and constraints. Distributed architectures, on the other hand, partition the optimization

Portions of this chapter previously appeared as: A. Dener and J. E. Hicken, "Matrix-free algorithm for the optimization of multidisciplinary systems," *Structural and Multidisciplinary Optimization*, vol. 56, no. 6, pp. 1429–1446, Jun. 2017

problem into sequential or nested subproblems, each solving only a subset of the design variables and constraints.

Distributed architectures are primarily motivated by the way engineering design is performed in industry, where large systems are broken up into largely independent subsystems and assigned to separate teams of engineers. For instance, the propulsion and airframe subsystems for an aircraft are typically designed by different departments or companies that each possess specialized expertise over the governing physics of their discipline. Distributed architectures allow these teams to work independently and communicate infrequently.

However, distributed architectures are also known to exhibit poor convergence rates at the multidisciplinary system-level optimization, and they are not competitive with monolithic architectures on large-scale problems with strong interdisciplinary coupling [2]. These characteristics motivate us to adopt a monolithic MDO architecture instead; our goal in this thesis is to develop an algorithm that can solve monolithic MDO problems efficiently, while maintaining modularity of the underlying disciplines.

To facilitate this effort, we begin with a review of monolithic MDO architectures and related research. For a more comprehensive review of all MDO architectures, we refer the reader to Martins and Lambe [3].

1.1 A Generic Multidisciplinary Analysis Statement

In order to effectively discuss MDO architectures, we first introduce a model multidisciplinary analysis (MDA) problem. For simplicity, we will restrict our discussion to two physical disciplines represented by the algebraic state equations

$$\begin{aligned} \mathcal{R}_{\boldsymbol{u}}(\boldsymbol{u}, \mathcal{E}_{\boldsymbol{u}}(\boldsymbol{v}), \boldsymbol{x}) &= \boldsymbol{0}, \\ \mathcal{R}_{\boldsymbol{v}}(\boldsymbol{v}, \mathcal{E}_{\boldsymbol{v}}(\boldsymbol{u}), \boldsymbol{x}) &= \boldsymbol{0}, \end{aligned} \tag{1.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ are the design variables, and $\boldsymbol{u} \in \mathbb{R}^p$ and $\boldsymbol{v} \in \mathbb{R}^q$ are state variables associated with each discipline. These state equations depend on each other through $\mathcal{E}_{\boldsymbol{u}}(\boldsymbol{v}) \in \mathbb{R}^s$ and $\mathcal{E}_{\boldsymbol{v}}(\boldsymbol{u}) \in \mathbb{R}^r$, which are functions that extract or compute the information that couples the two disciplines together.

In the applications of interest, the MDA defined by (1.1) represents a set of discretized PDEs. For instance, in an aero-structural problem, $\mathcal{R}_{\boldsymbol{u}}$ might be the Euler equations discretized using a finite-volume scheme, and $\mathcal{R}_{\boldsymbol{v}}$ might be the equations of linear elasticity discretized using the finite-element method. In this example, the state $\boldsymbol{u}$ would denote the conservative flow variables averaged over each volume in the mesh, and $\boldsymbol{v}$ would be the nodal displacements of the finite-element model. Furthermore, $\mathcal{E}_{\boldsymbol{v}}$ uses the state $\boldsymbol{u}$ to compute the pressure on the wing surface, while $\mathcal{E}_{\boldsymbol{u}}$ uses $\boldsymbol{v}$ to determine the structural deformations of the outer mold line.

1.2 Simultaneous Analysis and Design

In order to covert the MDA in (1.1) into a generic MDO formulation, we introduce the objective function, $\mathcal{J} : \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$, and construct the optimization problem,

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}}{\text{minimize}} \quad & \mathcal{J}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}), \\ \text{subject to} \quad & \mathcal{R}_{\boldsymbol{u}}(\boldsymbol{u},\mathcal{E}_{\boldsymbol{u}}(\boldsymbol{v}),\boldsymbol{x}) = \boldsymbol{0}, \\ & \mathcal{R}_{\boldsymbol{v}}(\boldsymbol{v},\mathcal{E}_{\boldsymbol{v}}(\boldsymbol{u}),\boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{1.2}$$

where the discipline residuals are expressed as constraints to be satisfied by the optimization algorithm.

In the context of MDO, (1.2) is referred to as simultaneous analysis and design (SAND) [4]. Its defining characteristic is that the optimization variables include both the design variables, $\boldsymbol{x}$, and the state variables, $\boldsymbol{u}$ and $\boldsymbol{v}$. This approach, where the optimization algorithm is responsible for simultaneously converging both the design and the states, is more generally referred to as a "full-space" formulation in the PDE-constrained optimization literature.

Efficient full-space PDE-constrained optimization algorithms have been developed and applied to large-scale single-discipline problems [5]–[9]; however, the full-space formulation can be difficult to implement and poses challenges to modularity. In particular, PDE models often take advantage of specialized solution

techniques for efficiency and robustness. This is especially true for nonlinear PDEs, which may require tailored globalization strategies to ensure robust convergence. A general-purpose full-space algorithm cannot easily anticipate all such globalization strategies. The complexity of the task also increases with each added discipline and, to the best of our knowledge, no one has solved high-fidelity MDO problems using full-space methods.

1.3 Multidisciplinary Feasible

The difficulties in modularity for the SAND formulation motivate the use of the "reduced-space" formulation, given by

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \mathcal{J}(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x}), \boldsymbol{v}(\boldsymbol{x})) \\ \text{governed by} \quad & \mathcal{R}_{\boldsymbol{u}}(\boldsymbol{u}, \mathcal{E}_{\boldsymbol{u}}(\boldsymbol{v}), \boldsymbol{x}) = \boldsymbol{0}, \\ & \mathcal{R}_{\boldsymbol{v}}(\boldsymbol{v}, \mathcal{E}_{\boldsymbol{v}}(\boldsymbol{u}), \boldsymbol{x}) = \boldsymbol{0}. \end{aligned} \tag{1.3}$$

The notation change from "subject to" to "governed by" indicates that the state equations are no longer imposed as constraints in the optimization problem, but instead considered as governing equations that have to be solved at every new design point. Consequently, the reduced-space formulation defines the state variables as implicit functions of the design $\boldsymbol{x}$ via the state equations. Thus, the optimization variables consist only of the design variables. In the context of MDO, the reduced-space formulation (1.3) is referred to as multidisciplinary feasible (MDF).

Under the MDF formulation, every evaluation of the objective function $\mathcal{J}$ requires a complete solution of the coupled state equations (1.1). Block-Jacobi and block-Gauss-Seidel are popular choices for solving the coupled state equations in MDF, but these iterative methods have been shown to be considerably less efficient than monolithic Newton-Krylov MDAs [10]. Conversely, implementing such a monolithic MDA with existing legacy solvers requires significant intrusion into the source code, and may be impractical with commercial software. Furthermore, the use of gradient-based algorithms necessitates a coupled-adjoint implementation or expensive (and potentially inaccurate) finite-difference approximations. Despite

these challenges, there has been significant success applying the MDF formulation to, for example, aero-elastic optimization problems [11]–[14].

1.4 Individual Discipline Feasible

There are clear trade-offs between the full- and reduced-space formulations of solving multidisciplinary optimization problems. The SAND formulation may achieve better optimization efficiency, but it comes at the cost of greater time investment into software development. The MDF formulation can facilitate the reuse of existing PDE solvers, but at the expense of fully coupled MDAs at each optimization iteration.

This trade-off motivates a compromise, which we believe is provided by the individual discipline feasible (IDF) formulation [15], [16]:

$$\begin{aligned} \underset{\boldsymbol{x},\bar{\boldsymbol{u}},\bar{\boldsymbol{v}}}{\text{minimize}} \quad & \mathcal{J}(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x}, \bar{\boldsymbol{v}}), \boldsymbol{v}(\boldsymbol{x}, \bar{\boldsymbol{u}})), \\ \text{subject to} \quad & \mathcal{E}_{\boldsymbol{v}}(\boldsymbol{u}) - \bar{\boldsymbol{u}} = \boldsymbol{0}, \\ & \mathcal{E}_{\boldsymbol{u}}(\boldsymbol{v}) - \bar{\boldsymbol{v}} = \boldsymbol{0}, \\ \text{governed by} \quad & \mathcal{R}_{\boldsymbol{u}}(\boldsymbol{u}, \bar{\boldsymbol{v}}, \boldsymbol{x}) = \boldsymbol{0}, \\ & \mathcal{R}_{\boldsymbol{v}}(\boldsymbol{v}, \bar{\boldsymbol{u}}, \boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{1.4}$$

which modifies (1.3) by introducing coupling variables, $\bar{\boldsymbol{u}} \in \mathbb{R}^r$ and $\bar{\boldsymbol{v}} \in \mathbb{R}^s$, and their corresponding coupling constraints. Under the IDF formulation, the states are still expressed as implicit functions of design variables, $\boldsymbol{x}$, but they are no longer directly dependent on each other. This is because the state equations for each discipline can be solved independently using the coupling variables prescribed by the optimization algorithm. Consequently, IDF avoids a full MDA at every evaluation of the objective function. This advantage extends to sensitivity analysis as well, where the adjoint solutions are also decoupled.

Unfortunately, the decoupled state and adjoint equations come at a cost, as IDF presents two significant disadvantages. State-of-the-art gradient-based optimization algorithms frequently use limited-memory quasi-Newton approximations of the Hessian for such large-scale problems [17], [18], and it is known that these

approaches scale unfavorably with the size of the design space [19].

More significantly, the cost of evaluating the Jacobian of the IDF coupling constraints is prohibitively expensive for most multidisciplinary problems modeled with PDEs. To illustrate, the IDF formulation applied to a three-dimensional aero-structural problem could yield on the order of 10^3-10^4 new degrees of freedom in the optimization problem, and it requires the same number of adjoint solutions at every optimization iteration to form the constraint Jacobian.

However, these are limitations of the optimization algorithms used to solve IDF problems, and not of the IDF architecture itself. This suggests that alternative optimization algorithms may avoid these limitations.

1.5 Contributions

In this thesis, we develop a reduced-space inexact-Newton-Krylov (RSNK) algorithm targeted at addressing the challenges of the IDF formulation outlined above, and demonstrate its efficacy on a range of low- and high-fidelity problems drawn from the field of aerospace engineering. Our goals are to avoid explicit formulation of the Hessian or the constraint Jacobian, achieve good cost scaling with increasing number of design and coupling variables, and leverage the superlinear asymptotic convergence expected from inexact-Newton-type methods. By fulfilling these three goals, we hope to make the modular IDF formulation practical on large problems, and encourage greater use of MDO in general.

Below, we describe the core contributions of this thesis towards the stated goals, and provide an outline of the upcoming chapters.

1.5.1 Second-Order Adjoints for KKT Matrix-Vector Products

A key aspect of the RSNK algorithm is the use of a Krylov iterative method for the solution of the linear system that arises from applying Newton's method to the first-order optimality conditions. In this context, a matrix-free implementation requires the computation of an efficient matrix-vector product. However, the linear system solved in this application involves second-order derivative information that is typically not available in PDE-governed optimization problems. In this thesis, we

introduce a second-order adjoint-based method for computing matrix-vector products for the solution of the first-order optimality conditions. While second-order adjoints themselves are not novel, to the best of our knowledge, this thesis and its related publications represent the first application of second-order adjoints in the context of equality-constrained optimization.

1.5.2 Matrix-Free Preconditioner for the IDF Architecture

The primal-dual linear equation that is solved at each Newton iteration by a Krylov solver is an ill-conditioned saddle-point system. An effective preconditioner is required to achieve good rates of convergence and produce high-quality step directions in the optimization. However, the matrix-free nature of the RSNK algorithm makes it impractical to use common preconditioning techniques based on incomplete factorizations. In this thesis, we will present a novel matrix-free preconditioner for the individual discipline feasible architecture, and demonstrate its efficacy on three different multidisciplinary design optimization problems.

1.5.3 Parallel-Agnostic Optimization Library

The core motivation behind the use of the individual discipline feasible architecture in this work is to take advantage of its modularity in discipline/PDE solvers. However, the RSNK algorithm we present in this thesis relies on products and solutions with the PDE Jacobian, and different PDE solvers adopt different data structures and parallelization schemes for these tasks. This presents the need for the implementation of the RSNK algorithm to remain agnostic to the implementation of the underlying PDE solver(s). In this thesis, we describe an optimization library written in Python that utilizes reverse communication techniques and a linear algebra abstraction layer to separate the optimization algorithms from the underlying solvers.

1.6 Thesis Outline

The remaining chapters of this thesis are organized as follows:

- Chapter 2 reviews the sequential quadratic optimization approach, and introduces the RSNK algorithm. The second-order adjoint-based KKT-matrix-vector product and the nonconvex Krylov solver are described in detail. The chapter concludes with a study of the RSNK algorithm's computational cost scaling with respect to the size of the design space on a high-fidelity aerodynamic shape optimization problem.

- Chapter 3 introduces a matrix-free preconditioning strategy for the IDF formulation, and demonstrates its efficacy on two low-fidelity problems.

- Chapter 4 discusses a parallel-agnostic optimization library that implements the RSNK algorithm, and benchmarks its components on a set of simple test problems.

- Chapter 5 presents an application of the RSNK algorithm on a high-fidelity multidisciplinary aero-structural optimization problem.

- Chapter 6 summarizes the work and lists potential future research avenues based on the findings.

CHAPTER 2
REDUCED-SPACE INEXACT-NEWTON-KRYLOV ALGORITHM

2.1 Introduction

Having motivated the choice of the IDF architecture, we now turn our attention to developing an optimization algorithm that can efficiently solve this type of problem formulation. To accomplish this goal, we will adopt an inexact-Newton-Krylov approach, which is known to exhibit superlinear asymptotic convergence and favorable algorithmic scaling as the number of variables increases on a wide range of nonlinear problems. We aim to take advantage of these features in reduced-space optimization, and render the computational cost of the IDF formulation practically viable for high-fidelity engineering problems.

To facilitate the discussion, we re-state the IDF formulation first described in Chapter 1:

$$\begin{aligned} \underset{\boldsymbol{x},\bar{\boldsymbol{u}},\bar{\boldsymbol{v}}}{\text{minimize}} \quad & \mathcal{J}(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x},\bar{\boldsymbol{v}}), \boldsymbol{v}(\boldsymbol{x},\bar{\boldsymbol{u}})), \\ \text{subject to} \quad & \mathcal{E}_{\boldsymbol{v}}(\boldsymbol{u}) - \bar{\boldsymbol{u}} = \boldsymbol{0}, \\ & \mathcal{E}_{\boldsymbol{u}}(\boldsymbol{v}) - \bar{\boldsymbol{v}} = \boldsymbol{0}, \\ \text{governed by} \quad & \mathcal{R}_{\boldsymbol{u}}(\boldsymbol{u},\bar{\boldsymbol{v}},\boldsymbol{x}) = \boldsymbol{0}, \\ & \mathcal{R}_{\boldsymbol{v}}(\boldsymbol{v},\bar{\boldsymbol{u}},\boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{2.1}$$

where $\boldsymbol{x}$, $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ are the optimization variables (also called "design" variables), $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ are coupling variables, and $\boldsymbol{u}$ and $\boldsymbol{v}$ are state variables. Recall that $\mathcal{E}_{\boldsymbol{u}}$ and $\mathcal{E}_{\boldsymbol{v}}$ are functions that extract or compute the information that couples the two

Portions of this chapter previously appeared as: A. Dener and J. E. Hicken, "Matrix-free algorithm for the optimization of multidisciplinary systems," *Structural and Multidisciplinary Optimization*, vol. 56, no. 6, pp. 1429–1446, Jun. 2017

Portions of this chapter previously appeared as: A. Dener, G. K. W. Kenway, J. E. Hicken, and J. R. R. A. Martins, "Comparison of inexact- and quasi-Newton algorithms for aerodynamic shape optimization," presented at the 53rd AIAA Aerospace Sciences Meeting, Kissimmee, FL, USA, 2015

disciplines together.

We would like to develop an algorithm that is generally applicable to all reduced-space PDE-governed optimization problems. To that end, we combine the coupling constraints into a single constraint vector and the state equations into a single residual vector:

$$\mathcal{C}(\boldsymbol{y}, \boldsymbol{w}) = \begin{bmatrix} \mathcal{E}_{\boldsymbol{v}}(\boldsymbol{u}) - \bar{\boldsymbol{u}} \\ \mathcal{E}_{\boldsymbol{u}}(\boldsymbol{v}) - \bar{\boldsymbol{v}} \end{bmatrix}, \quad \text{and} \quad \mathcal{R}(\boldsymbol{y}, \boldsymbol{w}) = \begin{bmatrix} \mathcal{R}_{\boldsymbol{u}}(\boldsymbol{x}, \boldsymbol{u}, \bar{\boldsymbol{v}}) \\ \mathcal{R}_{\boldsymbol{v}}(\boldsymbol{x}, \boldsymbol{v}, \bar{\boldsymbol{u}}) \end{bmatrix}, \tag{2.2}$$

where $\boldsymbol{w}^T = [\boldsymbol{u}^T \ \boldsymbol{v}^T]$ denotes the multidisciplinary state and $\boldsymbol{y}^T = [\boldsymbol{x}^T \ \bar{\boldsymbol{u}}^T \ \bar{\boldsymbol{v}}^T]$ denotes the optimization variables. With this notation, the IDF optimization statement simplifies to

$$\begin{aligned} \underset{\boldsymbol{y}}{\text{minimize}} \quad & \mathcal{J}(\boldsymbol{y}, \boldsymbol{w}(\boldsymbol{y})), \\ \text{subject to} \quad & \mathcal{C}(\boldsymbol{y}, \boldsymbol{w}(\boldsymbol{y})) = \boldsymbol{0}, \\ \text{governed by} \quad & \mathcal{R}(\boldsymbol{y}, \boldsymbol{w}) = \boldsymbol{0}, \end{aligned} \tag{2.3}$$

where the state equations $\boldsymbol{w}(\boldsymbol{y})$ are implicit functions of the design variables, as is expected in a reduced-space formulation. This simplification will allow us to define a general-purpose optimization algorithm applicable to both single and multidisciplinary problems with an arbitrary number of disciplines and nonlinear equality constraints.

In this chapter, we begin with a review of sequential quadratic programming, paying close attention to considerations relevant to PDE-governed reduced-space problems. We then introduce the building blocks of the inexact-Newton-Krylov approach and assemble them into a complete algorithm. The chapter will conclude with an application of the algorithm to a high-fidelity single-discipline aerodynamic shape optimization problem.

2.2 Sequential Quadratic Optimization Review

Our target applications in this thesis are large-scale PDE-governed optimization problems where the objective function and constraints depend both on design

and state variables. The relationship between these functionals and the design variables involves the solution of nonlinear state equations, which can be computationally expensive in high-fidelity problems. This motivates the use of gradient-based nonlinear optimization techniques that can avoid large numbers of expensive function evaluations.

Sequential quadratic optimization (SQO), also commonly known as sequential quadratic programming (SQP), is one such gradient-based method that can effectively solve nonlinear optimization problems where the first and second derivatives of the objective function and the constraints are continuous. SQO computes steps by solving a series of quadratic subproblems, and it can be used in both a line search and trust region framework. In this section, we present a review of SQO in order to provide a theoretical foundation for our inexact-Newton algorithm.

We begin by introducing the reduced-space Lagrangian for the optimization problem in (2.3):

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{\lambda}) = \mathcal{J}(\boldsymbol{y}, \boldsymbol{w}(\boldsymbol{y})) + \boldsymbol{\lambda}^T \mathcal{C}(\boldsymbol{y}, \boldsymbol{w}(\boldsymbol{y})), \tag{2.4}$$

where $\boldsymbol{\lambda}$ are the Lagrange multipliers associated with the equality constraints. Differentiating (2.4) with respect to $\boldsymbol{y}$ and $\boldsymbol{\lambda}$ produces the Karush-Kuhn-Tucker (KKT) first-order necessary conditions that must be satisfied by the local optimum:

$$\frac{d\mathcal{L}}{d\boldsymbol{y}} = \frac{d\mathcal{J}}{d\boldsymbol{y}} + \boldsymbol{\lambda}^T \frac{d\mathcal{C}}{d\boldsymbol{y}} = \mathbf{0}, \tag{2.5a}$$

$$\frac{d\mathcal{L}}{d\boldsymbol{\lambda}} = \mathcal{C} = \mathbf{0}, \tag{2.5b}$$

where $d/d\boldsymbol{y}$ denotes a total derivative operator with respect to $\boldsymbol{y}$, and $d/d\boldsymbol{\lambda}$ denotes a total derivative with respect to $\boldsymbol{\lambda}$.

Let $(\boldsymbol{y}_k, \boldsymbol{\lambda}_k)$ be the k^{th} iterate in an SQO algorithm. The SQO approach constructs a quadratic subproblem,

$$\begin{aligned} \underset{\boldsymbol{p}}{\text{minimize}} \quad & \mathcal{L}_k + \boldsymbol{p}^T \frac{d\mathcal{L}_k}{d\boldsymbol{y}} + \frac{1}{2}\boldsymbol{p}^T \mathsf{W}_k \boldsymbol{p}, \\ \text{subject to} \quad & \mathsf{A}_k \boldsymbol{p} + \mathcal{C}_k = \mathbf{0}, \end{aligned} \tag{2.6}$$

where $\mathsf{W}_k = d^2\mathcal{L}_k/d\boldsymbol{y}^2$ is the Hessian of the Lagrangian, and $\mathsf{A}_k = d\mathcal{C}_k/d\boldsymbol{y}$ is the total constraint Jacobian evaluated at $(\boldsymbol{y}_k, \boldsymbol{\lambda}_k)$. The constant term $\mathcal{L}_k$ is typically discarded from the problem as it has no dependence on $\boldsymbol{p}$ and does not affect the optimization.

Under the assumptions that the constraint Jacobian A_k has full row rank and the Hessian W_k is positive-definite in the null-space of the constraints, the quadratic subproblem has a unique solution, $(\boldsymbol{p}, \boldsymbol{d})$, given by

$$\begin{aligned} \mathsf{W}_k\boldsymbol{p} + \boldsymbol{A}_k^T(\boldsymbol{d} + \boldsymbol{\lambda}_k) + \frac{d\mathcal{J}}{d\boldsymbol{y}} &= \mathbf{0}, \\ \mathsf{A}_k\boldsymbol{p} + \mathcal{C}_k &= \mathbf{0}. \end{aligned} \tag{2.7}$$

This system of equations can also be expressed as

$$\begin{bmatrix} \mathsf{W}_k & \mathsf{A}_k^T \\ \mathsf{A}_k & 0 \end{bmatrix} \begin{pmatrix} \boldsymbol{p} \\ \boldsymbol{d} \end{pmatrix} = \begin{pmatrix} -\frac{d\mathcal{J}_k}{d\boldsymbol{y}} - \mathsf{A}_k^T\boldsymbol{\lambda}_k \\ -\mathcal{C}_k \end{pmatrix}, \tag{2.8}$$

which is equivalent to applying the Newton's method to the KKT conditions in (2.5). In the present work, we take advantage of this relationship to construct our inexact-Newton algorithm.

SQO algorithms employ a number of different methods to solve (2.7). The KKT matrix may be directly inverted; however, the indefinite nature of this matrix requires indefinite factorization techniques such as those proposed by Bunch and Kaufman [21] or Duff and Reid [22]. Other popular approaches include range-space [23], [24] and null-space [24], [25] methods.

The common thread across these solution methods is that they all require either the Hessian of the Lagrangian, or its inverse. However, the Hessian contains second-order derivative information that is usually not readily available, especially in PDE-governed optimization. Most SQO algorithms address this issue through the use of quasi-Newton approximations of the Hessian, which was first proposed by Davidon [26] and later popularized by Fletcher [27]. Today, the most commonly used quasi-Newton approximations are the Symmetric-Rank-1 [28] and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [29]–[32] methods: BFGS also has a limited-

memory variant [33] that is often used on large-scale problems.

Quasi-Newton approximations of the Hessian utilize updates based on the first derivative of the Lagrangian. The same first derivative is also used for convergence checks at each optimization iteration. Furthermore, most SQO solution methods also require the constraint Jacobian A_k to be provided explicitly for factorization. In reduced-space optimization, these first derivatives are total derivatives involving sensitivities with respect to both the design and state variables, and are typically not available analytically. Therefore, to present a complete review of SQO in the context of PDE-governed optimization, we must also address how to provide the required derivatives.

2.2.1 Adjoint Method for Calculating Derivatives

Finite difference methods can be used to compute first-order total derivatives when the objective function and constraints are computationally cheap to evaluate. However, these function evaluations in PDE-governed optimization involve the potentially nonlinear solution of state equations for each design perturbation. Depending on the computational cost of the PDE solution itself, the finite difference approach may become intractable even for small design spaces with only tens of variables and just a few constraints. Instead, the first-order total derivatives can be efficiently computed via the adjoint method [34], [35], which we will review in this section.

We begin by introducing the full-space Lagrangian,

$$\hat{\mathcal{L}}(\boldsymbol{y}, \boldsymbol{\lambda}, \boldsymbol{w}, \boldsymbol{\Psi}) = \mathcal{J}(\boldsymbol{y}, \boldsymbol{w}) + \boldsymbol{\lambda}^T \mathcal{C}(\boldsymbol{y}, \boldsymbol{w}) + \boldsymbol{\Psi}^T \mathcal{R}(\boldsymbol{y}, \boldsymbol{w}), \tag{2.9}$$

where the state equations are now imposed as constraints and associated with the Lagrange multipliers in $\boldsymbol{\Psi}$. As before, differentiating (2.9) with respect to its inputs produces the KKT conditions for the full-space problem:

$$\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{y}} + \boldsymbol{\lambda}^T \frac{\partial \mathcal{C}}{\partial \boldsymbol{y}} + \boldsymbol{\Psi}^T \frac{\partial \mathcal{R}}{\partial \boldsymbol{y}} = \mathbf{0}, \tag{2.10a}$$

$$\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{\lambda}} = \mathcal{C} = \mathbf{0}, \tag{2.10b}$$

$$\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{w}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{w}} + \boldsymbol{\lambda}^T \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}} + \boldsymbol{\Psi}^T \frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} = \mathbf{0}, \tag{2.11}$$

$$\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{\Psi}} = \mathcal{R} = \mathbf{0}, \tag{2.12}$$

where all the derivatives above denote partial derivatives rather than total derivatives.

Recall that in the reduced-space formulation, the state variables are implicit functions of the design via the state equations. This means that (2.12) is satisfied at every design point, and can be eliminated from the problem.

Similarly, (2.11), which is rewritten below in residual form, can also be solved at each optimization iteration to determine the multipliers in $\boldsymbol{\Psi}$:

$$\mathcal{S}(\boldsymbol{y}, \boldsymbol{\lambda}, \boldsymbol{w}, \boldsymbol{\Psi}) \coloneqq \frac{\partial \mathcal{J}}{\partial \boldsymbol{w}} + \boldsymbol{\lambda}^T \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}} + \boldsymbol{\Psi}^T \frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} . \tag{2.13}$$

The matrix multiplying $\boldsymbol{\Psi}$ is the transposed Jacobian of the state equations, which must be invertible if the state equations have a unique solution. Consequently, the multipliers in $\boldsymbol{\Psi}$ can be defined as implicit functions of $\boldsymbol{y}$ and the multipliers in $\boldsymbol{\lambda}$.

Solving (2.11) and (2.12) at each optimization iteration, and, thus, eliminating them from the full-space KKT conditions, establishes an equivalence between (2.5) and (2.10) via $\mathcal{R}(\boldsymbol{y}, \boldsymbol{w}) = \mathbf{0}$ and $\mathcal{S}(\boldsymbol{y}, \boldsymbol{\lambda}, \boldsymbol{w}, \boldsymbol{\Psi}) = \mathbf{0}$:

$$\begin{aligned} \frac{d\mathcal{L}}{d\boldsymbol{y}} &= \frac{d\mathcal{J}}{d\boldsymbol{y}} + \boldsymbol{\lambda}^T \frac{d\mathcal{C}}{d\boldsymbol{y}} \\ &= \frac{\partial \mathcal{J}}{\partial \boldsymbol{y}} + \boldsymbol{\lambda}^T \frac{\partial \mathcal{C}}{\partial \boldsymbol{y}} + \boldsymbol{\Psi}^T \frac{\partial \mathcal{R}}{\partial \boldsymbol{y}} = \mathbf{0}, \\ \frac{d\mathcal{L}}{d\boldsymbol{\lambda}} &= \mathcal{C} = \mathbf{0}, \end{aligned} \tag{2.14}$$

which reveals the role of $\boldsymbol{\Psi}$ in assembling the total derivatives in the reduced-space. We will henceforth refer to $\boldsymbol{\Psi}$ as the first-order adjoint, and (2.13) as the first-order adjoint equation. This concludes our review of the SQO method.

2.3 Inexact-Newton for Optimization

Recall from the earlier review that sequential quadratic optimization is equivalent to applying the Newton's method to the nonlinear system of equations in (2.5),

at least in the case where W is positive-definite. This produces the KKT system (also known as the primal-dual system),

$$\begin{bmatrix} \mathsf{W} & \mathsf{A}^T \\ \mathsf{A} & 0 \end{bmatrix} \begin{pmatrix} \boldsymbol{p} \\ \boldsymbol{d} \end{pmatrix} = - \begin{pmatrix} \frac{d\mathcal{L}}{d\boldsymbol{y}} \\ \frac{d\mathcal{L}}{d\boldsymbol{\lambda}} \end{pmatrix}, \tag{2.15}$$

where we have dropped the subscript k to simplify the notation.

Note that each row in the constraint Jacobian A in the reduced-space is a total derivative. As discussed in the SQO review, the conventional approach taken in gradient-based optimization algorithms requires this matrix to be provided explicitly, because most conventional optimization algorithms need to factor A to find its null-space. However, computing A explicitly for the IDF formulation requires a separate adjoint solution for each nonlinear coupling constraint, which is impractical in general. The same limitation is also present in general PDE-governed optimization problems that feature large numbers of state-based constraints.

Few reduced-space algorithms address the challenge presented by the IDF Jacobian, and state-based constraint Jacobians more generally. One notable exception is the matrix-free augmented Lagrangian algorithm developed by Arreckx *et al.* [36]. However, this approach relies on quasi-Newton approximations that we would like to avoid; large-scale problems targeted in the present work often necessitate the use of limited-memory quasi-Newton approximations, which are known to scale unfavorably with the size of the design space [19].

In contrast, there have been significant efforts in the full-space optimization literature to develop matrix-free[1] Newton-Krylov algorithms that avoid explicit constraint Jacobians [37]–[40]. Inspired by full-space algorithms, we solve the reduced-space KKT system in (2.15), inexactly, using a Krylov solver. We call this approach the "Reduced-Space inexact-Newton-Krylov algorithm", or RSNK for short.

2.3.1 Matrix-Free KKT Matrix-Vector Products

Most Krylov iterative methods can be implemented in a matrix-free fashion because they rely on matrix-vector products rather than the explicit matrix itself.

[1] Notwithstanding the possible use of matrix-based preconditioners for the state Jacobian.

Taking advantage of this feature in sequential quadratic optimization requires a matrix-vector product with the KKT matrix, or, alternatively, products with its sub-blocks. In full-space PDE-constrained optimization problems, the constraint Jacobian is the Jacobian of the PDE residual, which involves partial derivatives only. Products with this Jacobian are readily available, for example, in most finite-element codes, or through algorithmic differentiation. The reduced-space constraint Jacobian, on the other hand, is a total derivative and requires additional considerations. In this section, we will introduce a second-order adjoint-based formulation for computing products between the reduced-space KKT matrix and arbitrary vectors.

We begin by recognizing that the KKT matrix in (2.15) is the Jacobian of the KKT conditions. Consequently, we can approximate products with this matrix and an arbitrary vector $\boldsymbol{z}^T = [\boldsymbol{z}_{\boldsymbol{y}}^T, \boldsymbol{z}_{\boldsymbol{\lambda}}^T]$ via forward-differencing on (2.5); that is

$$\mathsf{K}\boldsymbol{z} = \begin{bmatrix} \mathsf{W} & \mathsf{A}^T \\ \mathsf{A} & 0 \end{bmatrix} \begin{pmatrix} \boldsymbol{z}_{\boldsymbol{y}} \\ \boldsymbol{z}_{\boldsymbol{\lambda}} \end{pmatrix} \approx \frac{1}{\epsilon} \begin{pmatrix} \mathcal{G}(\boldsymbol{y} + \epsilon \boldsymbol{z}_{\boldsymbol{y}}, \boldsymbol{\lambda} + \epsilon \boldsymbol{z}_{\boldsymbol{\lambda}}) - \mathcal{G}(\boldsymbol{y}, \boldsymbol{\lambda}) \\ \mathcal{C}(\boldsymbol{y} + \epsilon \boldsymbol{z}_{\boldsymbol{y}}) - \mathcal{C}(\boldsymbol{y}) \end{pmatrix}, \tag{2.16}$$

where $\mathcal{G} = d\mathcal{L}/d\boldsymbol{y}$ is the total derivative of the Lagrangian with respect to the design variables, and ϵ is the finite-difference step size.

When Jacobian-free Newton-Krylov methods are used to solve nonlinear PDEs [41], the nonlinear equation (i.e. the PDE residual) explicitly depends on the state variables that are being perturbed in the forward-difference approximation. This allows PDE-Jacobian-vector products to be approximated relatively cheaply. In contrast, in reduced-space PDE-governed optimization, the state variables are implicit functions of the design; therefore, in a naive implementation of the forward-difference approximation (2.16), we must solve the nonlinear state equations and the first-order adjoint at the perturbed point every time we evaluate a KKT-matrix-vector product. As we will demonstrate below, the linear adjoint solution is unavoidable, but the nonlinear state solution can be replaced with a linear problem.

Let $\boldsymbol{w} + \epsilon\boldsymbol{\sigma}$ be the state corresponding to the perturbed design point $\boldsymbol{y} + \epsilon\boldsymbol{z}_{\boldsymbol{y}}$ such that,

$$\mathcal{R}(\boldsymbol{y} + \epsilon \boldsymbol{z}_{\boldsymbol{y}}, \boldsymbol{w} + \epsilon\boldsymbol{\sigma}) = \boldsymbol{0}. \tag{2.17}$$

For an infinitesimal perturbation ϵ, we can use a first-order Taylor expansion to expand the left hand side of the above equation to arrive at

$$\mathcal{R}(\boldsymbol{y} + \epsilon \boldsymbol{z_y}, \boldsymbol{w} + \epsilon \boldsymbol{\sigma}) = \mathcal{R}(\boldsymbol{y}, \boldsymbol{w}) + \epsilon \frac{\partial \mathcal{R}}{\partial \boldsymbol{y}} \boldsymbol{z_y} + \epsilon \frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} \boldsymbol{\sigma} + \mathcal{O}(\epsilon^2) = \boldsymbol{0}. \tag{2.18}$$

Recognizing that $\mathcal{R}(\boldsymbol{y}, \boldsymbol{w}) = \boldsymbol{0}$, we rearrange (2.18) and take the limit as $\epsilon \to 0$ to get the linear system,

$$\frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} \boldsymbol{\sigma} = - \frac{\partial \mathcal{R}}{\partial \boldsymbol{y}} \boldsymbol{z_y}, \tag{2.19}$$

where $\boldsymbol{\sigma}$ is a second-order adjoint [42], so called because it helps us assemble second-derivative information.

We repeat this process with the first-order adjoint residual in (2.13). Let $\boldsymbol{\Psi} + \epsilon \boldsymbol{\Phi}$ be the first-order adjoint corresponding to the perturbed primal-dual point $(\boldsymbol{y} + \epsilon \boldsymbol{z_y}, \boldsymbol{\lambda} + \epsilon \boldsymbol{z_\lambda})$ such that,

$$\begin{aligned} \mathcal{S}(\boldsymbol{y} + \epsilon \boldsymbol{z_y}, \boldsymbol{\lambda} + \epsilon \boldsymbol{z_\lambda}, \boldsymbol{w} + \epsilon \boldsymbol{\sigma}, \boldsymbol{\Psi} + \epsilon \boldsymbol{\Phi}) \\ = \mathcal{S}(\boldsymbol{y}, \boldsymbol{\lambda}, \boldsymbol{w}, \boldsymbol{\Psi}) + \epsilon \frac{\partial \mathcal{S}}{\partial \boldsymbol{y}} \boldsymbol{z_y} + \epsilon \frac{\partial \mathcal{S}}{\partial \boldsymbol{\lambda}} \boldsymbol{z_\lambda} \\ + \epsilon \frac{\partial \mathcal{S}}{\partial \boldsymbol{w}} \boldsymbol{\sigma} + \epsilon \frac{\partial \mathcal{S}}{\partial \boldsymbol{\Psi}} \boldsymbol{\Phi} + \mathcal{O}(\epsilon^2) = \boldsymbol{0}. \end{aligned} \tag{2.20}$$

Similar to the primal residual, we recognize that the adjoint residual, $\mathcal{S}(\boldsymbol{y}, \boldsymbol{\lambda}, \boldsymbol{w}, \boldsymbol{\Phi})$, is zero and eliminate it from (2.20). Additionally, we note that the partial derivatives of $\mathcal{S}$ with respect to $\boldsymbol{\lambda}$ and $\boldsymbol{\Psi}$ are given by

$$\frac{\partial \mathcal{S}}{\partial \boldsymbol{\lambda}} = \left(\frac{\partial \mathcal{C}}{\partial \boldsymbol{w}} \right)^T \quad \text{and} \quad \frac{\partial \mathcal{S}}{\partial \boldsymbol{\Psi}} = \left(\frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} \right)^T. \tag{2.21}$$

These modifications simplify (2.20) into the linear system,

$$\left(\frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} \right)^T \boldsymbol{\Phi} = - \left(\frac{\partial \mathcal{C}}{\partial \boldsymbol{w}} \right)^T \boldsymbol{z_\lambda} - \begin{bmatrix} \frac{\partial \mathcal{S}}{\partial \boldsymbol{y}} & \frac{\partial \mathcal{S}}{\partial \boldsymbol{w}} \end{bmatrix} \begin{pmatrix} \boldsymbol{z_y} \\ \boldsymbol{\sigma} \end{pmatrix}, \tag{2.22}$$

where $\boldsymbol{\Phi}$ is another second-order adjoint used in assembling the KKT-matrix-vector

product.

Unfortunately, the partial derivatives of $\mathcal{S}$ with respect to $\boldsymbol{y}$ and $\boldsymbol{w}$ in (2.22) involve the second-derivatives of the objective function, the constraints, and the state equations. For a general-purpose algorithm, we cannot assume that the underlying PDE solvers possess the infrastructure needed to compute these second-derivatives directly. Instead, we define the second term on the right-side of (2.22) in terms of known first-derivatives. To do so, we make use of the fact that this term is in the form of a Jacobian-vector product and evaluate it using forward-differencing on (2.13):

$$\left[\begin{array}{cc} \frac{\partial \mathcal{S}}{\partial \boldsymbol{y}} & \frac{\partial \mathcal{S}}{\partial \boldsymbol{w}} \end{array} \right] \begin{pmatrix} \boldsymbol{z_y} \\ \boldsymbol{\sigma} \end{pmatrix} \approx \frac{1}{\epsilon} \left[\, \mathcal{S}(\boldsymbol{y} + \epsilon \boldsymbol{z_y}, \boldsymbol{\lambda}, \boldsymbol{w} + \epsilon \boldsymbol{\sigma}, \boldsymbol{\Psi}) - \mathcal{S}(\boldsymbol{y}, \boldsymbol{\lambda}, \boldsymbol{w}, \boldsymbol{\Psi}) \, \right]. \tag{2.23}$$

The perturbation factor is chosen as $\epsilon = \sqrt{\epsilon_{mach}}/\|\boldsymbol{z_y}\|_2$ where ϵ_{mach} is the "machine zero" constant specific to the hardware. This approach was used by Nielsen *et al.* [43] for a CFD application and demonstrated to be more accurate than fixed perturbation factors. Our implementation in this algorithm includes two conditional modification. First, we pick $\epsilon = 1.0$ when $\|\boldsymbol{z_y}\|_2 < \epsilon_{mach}$ to safeguard against the multiplying vector being very small. Second, we set $\epsilon = \|\boldsymbol{y}\|_2 \sqrt{\epsilon_{mach}}/\|\boldsymbol{z_y}\|_2$ when $\|\boldsymbol{y}\|_2 \geq \epsilon_{mach}\|\boldsymbol{z_y}\|_2$ to account for cases where the current design point $\boldsymbol{y}$ dominates the multiplying vector $\boldsymbol{z_y}$ in magnitude.

We have now described all the pieces necessary to compute (2.16) while avoiding a solution of the nonlinear state equations. To summarize, we begin the assembly with the primal component of the KKT conditions, namely

$$\begin{aligned} \mathsf{W}\boldsymbol{z_y} + \mathsf{A}^T \boldsymbol{z_\lambda} &\approx \frac{1}{\epsilon} \left[\, \mathcal{G}(\boldsymbol{y} + \epsilon \boldsymbol{z_y}, \boldsymbol{\lambda} + \epsilon \boldsymbol{z_\lambda}) - \mathcal{G}(\boldsymbol{y}, \boldsymbol{\lambda}) \, \right] \\ &\approx \frac{1}{\epsilon} \Big[\left(\frac{\partial \mathcal{J}}{\partial \boldsymbol{y}} \right)_\epsilon + (\boldsymbol{\lambda} + \epsilon \boldsymbol{z_\lambda})^T \left(\frac{\partial \mathcal{C}}{\partial \boldsymbol{y}} \right)_\epsilon \\ &\quad + (\boldsymbol{\Psi} + \epsilon \boldsymbol{\Phi})^T \left(\frac{\partial \mathcal{R}}{\partial \boldsymbol{y}} \right)_\epsilon - \frac{\partial \mathcal{J}}{\partial \boldsymbol{y}} - \boldsymbol{\lambda}^T \frac{\partial \mathcal{C}}{\partial \boldsymbol{y}} - \boldsymbol{\Psi}^T \frac{\partial \mathcal{R}}{\partial \boldsymbol{y}} \Big] \end{aligned} \tag{2.24}$$

where the ϵ subscript denotes first-derivatives that are evaluated at the perturbed design $\boldsymbol{y} + \epsilon \boldsymbol{z_y}$ and the corresponding state variables $\boldsymbol{w} + \epsilon \boldsymbol{\sigma}$. Finally, we perform

a Taylor expansion on the dual component,

$$\begin{aligned} \mathsf{A}\boldsymbol{z}_{\boldsymbol{y}} &\approx \frac{1}{\epsilon}\left[\, \mathcal{C}(\boldsymbol{y}+\epsilon\boldsymbol{z}_{\boldsymbol{y}}) - \mathcal{C}(\boldsymbol{y}) \,\right] \\ &\approx \frac{1}{\epsilon}\left[\, \mathcal{C}(\boldsymbol{y}) + \epsilon\,\frac{\partial \mathcal{C}}{\partial \boldsymbol{y}}\,\boldsymbol{z}_{\boldsymbol{y}} + \epsilon\,\frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}\,\boldsymbol{\sigma} - \mathcal{C}(\boldsymbol{y}) \,\right] \\ &\approx \frac{\partial \mathcal{C}}{\partial \boldsymbol{y}}\,\boldsymbol{z}_{\boldsymbol{y}} + \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}\,\boldsymbol{\sigma}. \end{aligned} \tag{2.25}$$

With this second-order adjoint approach, we can compute KKT-matrix-vector products at a fixed cost of two linear system solutions based on the state Jacobian and transposed Jacobian. Significantly, the cost is virtually independent of the number of design variables and the number of constraints. This formulation extends to MDO problems without any changes, where the two required linear system solutions are based on the Jacobian of the combined multidisciplinary state equations. Another important observation is that the linear system solutions are performed at the $(\boldsymbol{y}, \boldsymbol{w})$ design-state point instead of the perturbed counterparts, so the PDE Jacobian(s) does not need to be reassembled nor preconditioner(s) refactored.

2.3.2 Extending FGMRES for Nonconvexity

The selection of the Krylov iterative method that solves (2.15) requires careful consideration of two particular issues: preconditioning and nonconvexity. For the first, we restrict our choice to flexible Krylov methods that enable the use of non-stationary preconditioners, and re-visit the subject of preconditioning in greater detail in Chapter 3. For the present discussion, we instead focus on the second challenge of nonconvexity.

In convex problems, where the Hessian of the Lagrangian, W, is positive-definitive in the null-space of the constraint Jacobian, A, it is possible to solve (2.15) using any flexible Krylov solver, such as the Flexible Generalized Minimal Residual (FGMRES) [44] method. However, in nonconvex problems, the Krylov solver may converge to stationary points that are not necessarily the minimum. To address this issue, we developed the FLexible Equality-Constrained Subproblem (FLECS) solver [45], which is a Krylov iterative method that extends FGMRES to produce descent directions in the presence of nonconvexity. We provide a review of

this method below.

Like FGMRES, FLECS constructs the primal-dual solution from the subspaces generated by the generalized Arnoldi procedure. If $\mathsf{Z}_j = [\boldsymbol{z}_1 \boldsymbol{z}_2 \dots \boldsymbol{z}_j]$ and $\mathsf{V}_{j+1} = [\boldsymbol{v}_1 \boldsymbol{v}_2 \dots \boldsymbol{v}_j]$ where V_{j+1} has orthonormal columns, then the generalized Arnoldi procedure produces

$$\mathsf{K}\mathsf{Z}_j = \mathsf{V}_{j+1}\bar{\mathsf{H}}_j. \tag{2.26}$$

In the present application, K is the KKT or the primal-dual matrix in (2.15), and $\bar{\mathsf{H}}_j$ is the associated upper Hessenberg matrix. This Arnoldi relation can be expanded into its primal and dual parts to reveal the relations for the KKT matrix sub-blocks;

$$\begin{aligned} \mathsf{W}\mathsf{Z}_j^p + \mathsf{A}^T\mathsf{Z}_j^d &= \mathsf{V}_{j+1}^p\bar{\mathsf{H}}_j, \\ \mathsf{A}\mathsf{Z}_j^p &= \mathsf{V}_{j+1}^d\bar{\mathsf{H}}_j, \end{aligned} \tag{2.27}$$

where the superscripts p and d denote the primal and dual components of the subspace vectors, respectively, and the subscript j denotes the size of the Krylov subspace.

To address nonconvexity, FLECS aims to minimize a quadratic penalty function with a trust radius constraint,

$$\begin{aligned} \underset{\boldsymbol{p}\in\text{span}\{\mathsf{Z}_j^p\}}{\text{minimize}} \quad & \mathcal{Q}(\boldsymbol{p},\mu) \equiv \mathcal{G}^T\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\mathsf{W}\boldsymbol{p} + \frac{\mu}{2}(\mathsf{A}\boldsymbol{p}+\mathcal{C})^T(\mathsf{A}\boldsymbol{p}+\mathcal{C}), \\ \text{subject to} \quad & ||\boldsymbol{p}|| \le \Delta, \end{aligned} \tag{2.28}$$

where $\mathcal{G} \equiv \partial\mathcal{L}/\partial\boldsymbol{y}$. The primal solution, $\boldsymbol{p} = \mathsf{Z}_j^p\boldsymbol{\rho}$, is restricted to the subspace $\text{span}\{\mathsf{Z}_j^p\}$ and bounded by the trust radius Δ. Using the Arnoldi relations, (2.28) can be reduced to the subproblem,

$$\begin{aligned} \underset{\boldsymbol{\rho}\in\mathbb{R}^j}{\text{minimize}} \quad & \left[\mathcal{G}_\mathsf{Z} + \mu(\mathsf{A}^T\mathcal{C})_\mathsf{Z}\right]^T\boldsymbol{\rho} + \frac{1}{2}\boldsymbol{\rho}^T\left[\mathsf{W}_\mathsf{Z} + \mu(\mathsf{A}^T\mathsf{A})_\mathsf{Z}\right]\boldsymbol{\rho}, \\ \text{subject to} \quad & ||\mathsf{Z}_j^p\boldsymbol{\rho}|| \le \Delta, \end{aligned} \tag{2.29}$$

where the subscript Z denotes a Galerkin projection of the terms onto the subspace

span$\{\mathsf{Z}_j\}$ (e.g. $\mathsf{W}_\mathsf{Z} = (\mathsf{Z}_j^p)^T \mathsf{W} \mathsf{Z}_j^p$). In the large-scale applications targeted in the present work, we restrict the Krylov subspace to at most 20 vectors. Consequently, the size of the subproblem in (2.29) remains small.

This small subproblem is solved using the Moré and Sorensen algorithm [46]. The resulting primal solution is combined with the traditional FGMRES solution for the dual variables, $\boldsymbol{d}_j = \mathsf{Z}_j^d \hat{\boldsymbol{\rho}}_j$, where

$$\hat{\boldsymbol{\rho}}_j = \underset{\hat{\boldsymbol{\rho}} \in \mathbb{R}^j}{\operatorname{argmin}} ||\beta \boldsymbol{e}_1 - \bar{\mathsf{H}}_j \hat{\boldsymbol{\rho}}||. \tag{2.30}$$

The use of a quadratic penalty term in FLECS raises the question of how to update the penalty parameter, μ, from one nonlinear iteration to the next. In our previous work, we proposed a heuristic of the form

$$\mu_{k+1} = \max\left(\mu_k, \mu_0 \frac{||\mathcal{C}_0||}{||\mathcal{C}_k||}\right), \tag{2.31}$$

where the subscript $k = 0, 1, 2, \ldots$ denotes the nonlinear optimization iteration. For the results presented in this thesis, we have slightly modified this original update such that,

$$\mu_{k+1} = \max\left(\mu_k, \mu_0 \frac{||\mathcal{C}_0||}{\min\left(||\mathcal{G}_k||,\ ||\mathcal{C}_k||\right)}\right). \tag{2.32}$$

The inclusion of $\mathcal{G}$ in the denominator causes the penalty parameter, μ, to grow more rapidly when the nonlinear convergence of the optimality norm outpaces feasibility. In our experiments, this modification has helped RSNK avoid extra nonlinear iterations near the primal optimum solely to restore feasibility.

2.3.3 Globalizing the Newton Step

The final remaining consideration for the RSNK algorithm is promoting convergence from remote starting points relative to the optimum. SQO algorithms commonly employ line-search or trust-region techniques where a merit function is used to evaluate step acceptance, and in the case of line-search methods, step length along a descent direction. In the present work, a trust-region framework [47] emerges

as a natural fit for globalizing the Newton step in RSNK, as it pairs effectively with the trust radius constraint in the FLECS quadratic subproblem.

For step acceptance, the RSNK algorithm relies on a simple filter [48], which tests $(\mathcal{J}_k, \|\mathcal{C}_k\|)$ pairs at each optimization iteration against previously accepted points. Steps that produce an improvement in either the objective or the constraint norm are accepted and added to the filter. Old points that are "dominated" by the newly added point (i.e. points where both the objective and constraint norm are worse) are removed from the filter. The filter-based approach eliminates the need to develop a predicted decrease approximation for any choice of a merit function, and thus, avoids the ambiguity associated with merit function parameters.

In the event that the filter rejects a Newton step, the FLECS solver reviewed in Section 2.3.2 provides both a second-order correction to account for the Maratos effect and an efficient re-solve that recycles the Krylov subspace built up during the original Newton-step solution. If the second-order correction does not lead to step acceptance, the re-solve is triggered with a smaller trust radius.

2.4 Algorithm Overview

We present an overview of our reduced-space inexact-Newton-Krylov implementation in Alg. 1. In line 9, the matrix-free KKT-matrix-vector product described in Section 2.3.1 is used in conjunction with the Krylov solver described in Section 2.3.2. The Krylov tolerance η in line 8 is dynamically adjusted to obtain superlinear convergence and avoid over solving [49]. The Newton step $\boldsymbol{s}_k$ is globalized in the trust-region framework outlined in Section 2.3.3.

Note that Alg. 1 is formulated for equality-constrained optimization. Its analogue for unconstrained problems is the trust-region Newton-CG algorithm [24], which is an inexact-Newton-Krylov method where the Krylov solver of choice is the Steihaug-Toint Conjugate Gradient method [50]. We apply this latter algorithm to the unconstrained MDF problems in Chapter 5, in conjunction with matrix-free Hessian-vector products formulated using a second-order adjoint approach analogous to the one described in Section 2.3.1 for the IDF products.

Algorithm 1: Reduced-space inexact-Newton-Krylov with filter globalization.

Data: $\boldsymbol{y}_0$, $\boldsymbol{\lambda}_0$, μ_0, η_0, Δ_0, $\Delta_{\min}$, $\Delta_{\max}$, τ_p, τ_d
Result: estimate for the optimal solution $(\boldsymbol{y}^*, \boldsymbol{\lambda}^*)$

set $\mu = \mu_0$, $\eta = \eta_0$, $\Delta = \Delta_0$
for $k = 0, 1, 2, \ldots,$ *max_iter* **do** `// start Newton loop`
  compute KKT conditions $\nabla \mathcal{L}_k = (\mathcal{G}_k^T, \mathcal{C}_k^T)^T$
  if $\|\mathcal{G}_k\| \leq \tau_p \|\mathcal{G}_0\|$ *and* $\|\mathcal{C}_k\| \leq \tau_d \|\mathcal{C}_0\|$ **then** `// check convergence`
    set $(\boldsymbol{y}^*, \boldsymbol{\lambda}^*) = (\boldsymbol{y}_k, \boldsymbol{\lambda}_k)$ and return
  end
  set $\mu \leftarrow \max\left[\mu,\ \mu_0 \|\mathcal{C}_0\| / \min\left(\|\mathcal{G}_k\|,\ \|\mathcal{C}_k\|\right)\right]$
  set $\eta \leftarrow \max\left[\eta,\ \min\left(1.0,\ \sqrt{\nabla \mathcal{L}_k / \nabla \mathcal{L}_0}\right),\ \min\left(\tau_p / \|\mathcal{G}_k\|,\ \tau_d / \|\mathcal{C}_k\|\right)\right]$
  solve (2.15) for $\boldsymbol{s}_k = (\boldsymbol{p}^T, \boldsymbol{d}^T)^T$ using FLECS, with tolerance η, penalty μ, and trust-radius Δ
  for $i = 0, 1, 2, \ldots,$ *max_filter_iter* **do** `// start filter loop`
    if $[\mathcal{J}(\boldsymbol{y}_k + \boldsymbol{p}),\ \|\mathcal{C}(\boldsymbol{y}_k + \boldsymbol{p})\|]$ *is not dominated by filter* **then**
     `// step accepted by filter`
      if $i = 0$ *and* $\|\boldsymbol{p}\| = \Delta$ **then** set $\Delta \leftarrow \min(2\Delta, \Delta_{\max})$
      set filter_success = true, and exit filter loop
    else `// step rejected by filter`
      if $i = 0$ **then**
        compute second-order correction $\boldsymbol{p}_c$
        if $[\mathcal{J}(\boldsymbol{y}_k + \boldsymbol{p} + \boldsymbol{p}_c),\ \|\mathcal{C}(\boldsymbol{y}_k + \boldsymbol{p} + \boldsymbol{p}_c)\|]$ *is not dominated by filter* **then**
          set filter_success = true, set $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{p}_c$, and exit filter loop
        end
      end
      if $\Delta = \Delta_{\min}$ **then** optimization failed, raise error
      set $\Delta \leftarrow \max(\Delta_{\min}, \frac{1}{4}\Delta)$, and re-solve (2.15) for $\boldsymbol{s}_k$ using FLECS, with penalty μ and trust-radius Δ
    end
  end
  if *filter_success* **then** set $\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \boldsymbol{p}$ and $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \boldsymbol{d}$
  else set $\boldsymbol{y}_{k+1} = \boldsymbol{y}_k$ and $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k$
end

2.5 Application: High-Fidelity Aerodynamic Shape Optimization

Having introduced the RSNK algorithm, we now apply it to the lift and pitching moment constrained drag minimization of the NASA Common Research Model (CRM) [51] wing. This single-discipline aerodynamic shape optimization (ASO) problem has been investigated previously by Lyu *et al.* [52] using SNOPT [18], an SQP library that utilizes a limited-memory quasi-Newton approach to approximate the Hessian of the Lagrangian. Consequently, we can use the ASO problem to compare the performance of our RSNK algorithm with that of the quasi-Newton method on a challenging high-fidelity PDE-constrained optimization problem. Our goal is to study the RSNK algorithm's computational cost scaling with increasing number of design variables.

2.5.1 Baseline Geometry

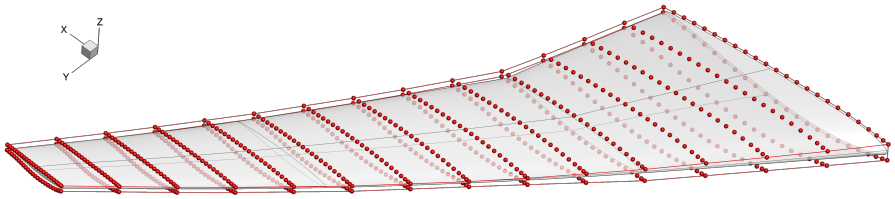


Fig. 2.1. CRM wing and the FFD parameterization [52].

The baseline geometry of interest is obtained by removing the fuselage and the tail from the complete CRM model. The wing is then parameterized using a free-form deformation (FFD) volume approach [53], where the number of FFD control points vary from 72 to 768 in six distinct FFD boxes. The configurations of these FFD boxes are described in Table 2.1, and Fig. 2.1 visualizes the FFD box with 768 control points as an example. The z-coordinates of these control points are used as design variables in the optimization formulation.

Table 2.1. Number of control points in each direction shown for the range of FFD box sizes used in the ASO test case.

	FFD Box Sizes					
Directions	**72**	**192**	**320**	**480**	**616**	**768**
Chordwise	6	12	16	20	22	24
Spanwise	6	8	10	12	14	16
Thickness	2	2	2	2	2	2

2.5.2 Objective Function and Problem Setup

The optimization problem considered in this test case is

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & C_D(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) + \gamma \boldsymbol{x}^T \boldsymbol{x}, \\ \text{subject to} \quad & C_L(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) = C_{L,0}, \\ & C_{My}(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) = C_{My,0}, \\ & V(\boldsymbol{x}) = V_0, \\ \text{governed by} \quad & \mathcal{R}_{Euler}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{0}. \end{aligned} \tag{2.33}$$

The coefficients of lift and pitching moment, C_L and C_{My}, are defined as equality constraints and fixed to their initial values, $C_{L,0} = 0.5$ and $C_{My,0} = -0.17$, respectively. Note that the negative direction for the pitching moment indicates a "nose up" rotation. Additionally, the Mach number and angle of attack are fixed at $M = 0.65$ and $\alpha = 2.2$ respectively.

The original problem formulation explored by Lyu *et al.* [52] also included inequality constraints on the wing thickness in order to prevent the optimization from generating unrealistically thin trailing edges and other wing shapes that may produce poor mesh elements. However, the RSNK algorithm in this work has not yet been extended to support inequality constraints. To address this limitation, we have opted to use a modified objective function that is the sum of the drag coefficient, C_D and a regularization term, which is the squared-$L2$ norm of the design vector multiplied by γ. This regularization has the effect of encouraging optimum designs that are close to the initial design, $\boldsymbol{x}_0$.

The parameter γ controls the magnitude of the regularization term, and it

must be tuned to ensure that the regularization does not overwhelm the coefficient of drag. This parameter was determined through trial and error such that the value of $\gamma \boldsymbol{x}^T \boldsymbol{x}$ is approximately three orders of magnitude smaller than the coefficient of drag at convergence. We will demonstrate with numerical results below that this regularized drag minimization problem behaves similarly to the original thickness constrained problem, and allows us to re-formulate the problem using only equality constraints.

2.5.3 CFD Solver

The coefficients of lift, drag, and pitching moment are evaluated using SUmb [54], a finite-volume flow solver for compressible Euler, laminar Navier-Stokes and RANS equations with support for a variety of turbulence models. SUmb also provides PDE Jacobian products, partial gradient evaluations, and adjoint solutions via matrix-free automatic differentiation. Mesh movement is performed by an efficient analytic inverse distance method [55].

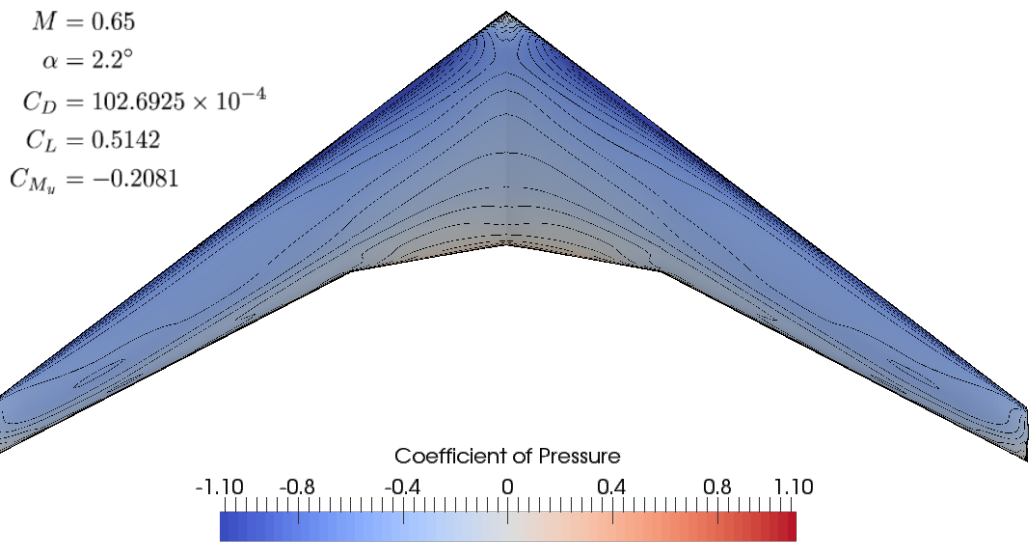


Fig. 2.2. Euler-based coefficient of pressure distribution on the initial CRM wing.

For our investigation, the flow is modeled using the compressible Euler equa-

tions, which are solved with a combined multigrid Runge-Kutta (RK) and Newton-Krylov (NK) approach. This solution was performed with a 840, 192-cell structured grid, using 64 MPI processes, run on Rensselaer Polytechnic Institute's DRP Cluster, equipped with two 8-core 2.6GHz Intel Xeon E5-2650 processors and 125 GB system memory per compute node. The coefficient of pressure distribution on the initial geometry is shown in Fig. 2.2.

2.5.4 Impact of Regularization

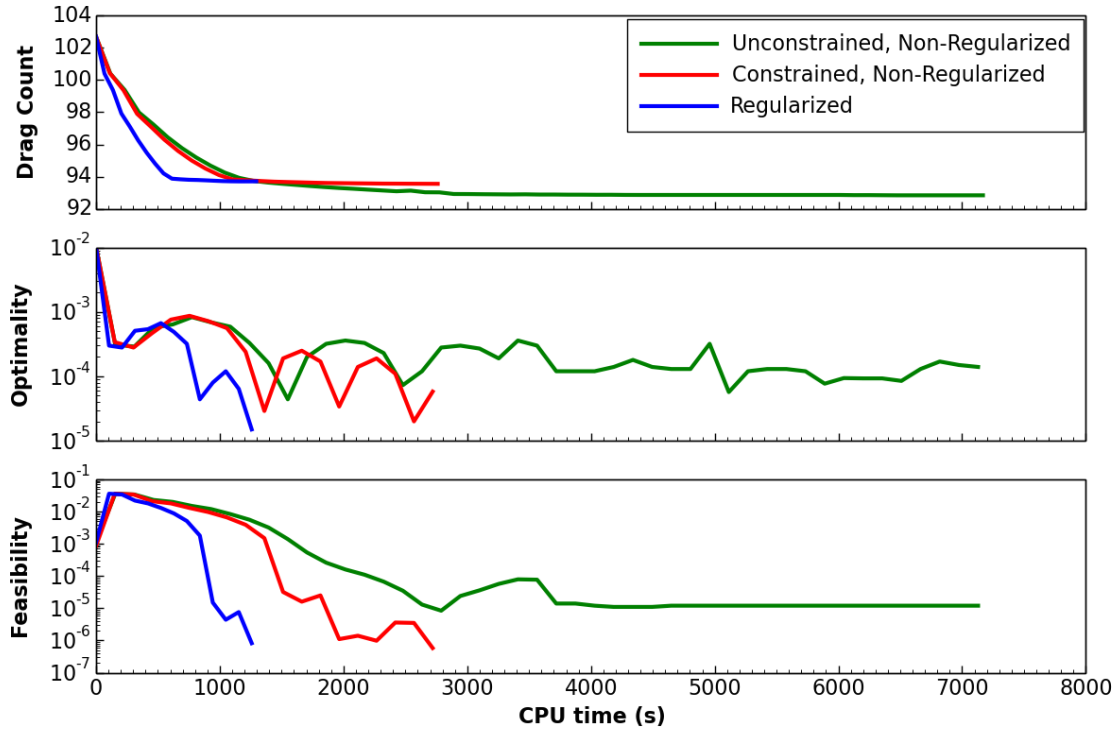


Fig. 2.3. Comparison of SNOPT convergences on three variations of the problem: non-regularized without thickness constraints, non-regularized with thickness constraints, and regularized proxy.

Before we compare the two optimization algorithms, it is important to illustrate the effect of the regularization on the optimization problem. To this end, Fig. 2.3 shows the SNOPT convergence histories for the regularized problem, the corresponding non-regularized problem without any thickness constraints, and the original formulation using the thickness constraints. Note that the objective value

is tracked in terms of "drag counts", which is equal to $10^4 C_D$.

The non-regularized problem, devoid of thickness constraints, is free to explore impractical geometric features, such as extremely thin trailing edges and sharp leading edges, in order to achieve a lower coefficient of drag. The regularized problem, on the other hand, appears to converge to a similar optimum design as the thickness constrained case, which suggests that it is a good proxy for the thickness constrained case. Furthermore, the increased convexity provided by the regularization makes the problem easier for SNOPT as well as RSNK, and, therefore, it does not unfairly favor one algorithm over the other.

2.5.5 Optimization Results

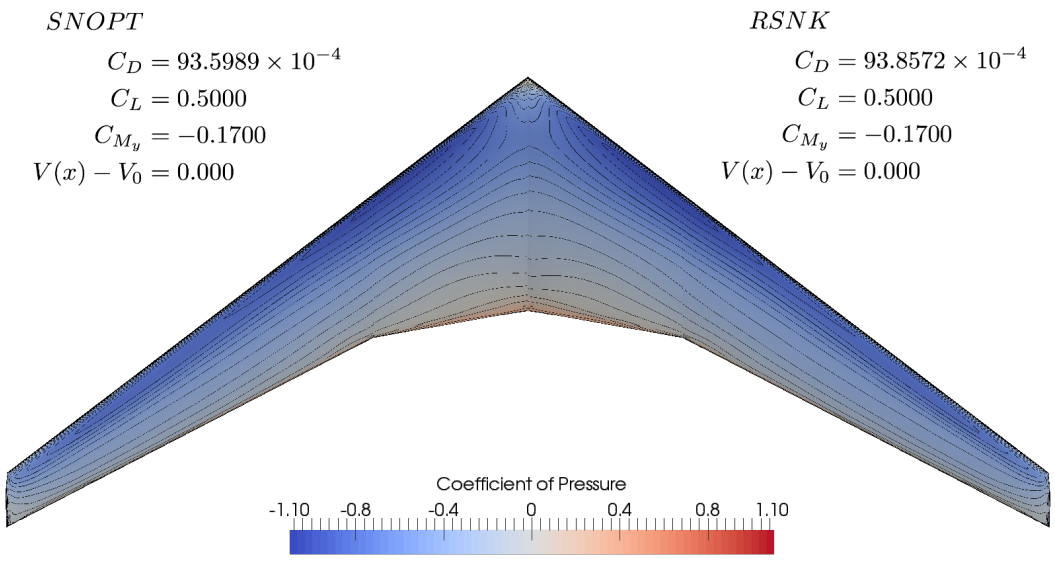


Fig. 2.4. Aerodynamic solution of the optimal design for SNOPT (left half) and RSNK (right half). C_L, C_{My}, and the volume constraint are approximately the same for both optimums to the 8^{th} decimal place.

The coefficient of pressure distributions over the optimized geometry for both RSNK and SNOPT are shown in Fig. 2.4 for the 192 design-variable case. The figures also include the C_D and constraint values for the geometries. The solution

is representative of the entire range of FFD sizes used in this study. All solutions of this regularized problem with either algorithm converge to optima within 0.5 drag counts of each other. A full set of convergence histories for both RSNK and SNOPT for all FFD boxes are available in Appendix A.

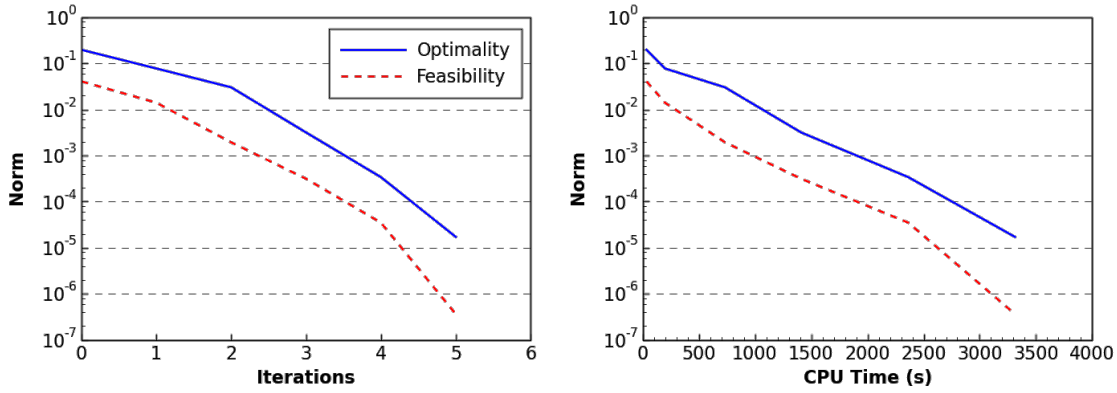


Fig. 2.5. Convergence history of RSNK with 192 FFD coefficients, plotted against CPU time (left) and iterations (right).

Fig. 2.5 shows the convergence history of RSNK on the regularized problem with 192 design variables. The history has been plotted against two different cost metrics: CPU time in seconds and iterations.

Plotted against iterations, RSNK exhibits the expected superlinear asymptotic convergence on this problem. However, superlinearity disappears when plotted against CPU time, because the convergence tolerance for the KKT system gets progressively smaller as the optimization approaches the solution; therefore, the FLECS solver requires more iterations to converge on this non-preconditioned problem. As a result, the elapsed time per outer iteration increases throughout the optimization process, causing the superlinear convergence to be lost when plotted against CPU time.

In practice, the RSNK algorithm aims to solve the KKT system inexactly, requiring convergence to a designated tolerance only within a small number of Krylov iterations. This inherently imposes a limit to how long each outer iteration can take. However, RSNKs superlinearity is nonetheless dependent on sufficiently reducing the KKT system residual. The superlinearity is lost when the solution to the KKT

system becomes too inexact.

The ideal number of Krylov iterations for the FLECS solver is largely dependent on the cost of solving the second-order adjoint systems and the matrix-vector products used to assemble the KKT-matrix-vector product described in Section 2.3.1. For this particular implementation and problem, our experiments indicate that establishing superlinear convergence requires 30 to 40 Krylov iterations. However, allowing this many iterations leads to a loss of competitiveness against SNOPT in terms of CPU time. In the following section, we will illustrate promising and competitive results achieved using 15 to 20 iterations, depending on the size of the design space, but it is worth noting that this restriction on the Krylov iterations leads to a loss of superlinear convergence.

This provides further evidence that an effective preconditioner for the KKT system is needed in order to produce high-quality step directions at lower computational cost. In the present work, we have developed one such preconditioner for the IDF formulation, which we will introduce in Chapter 3.

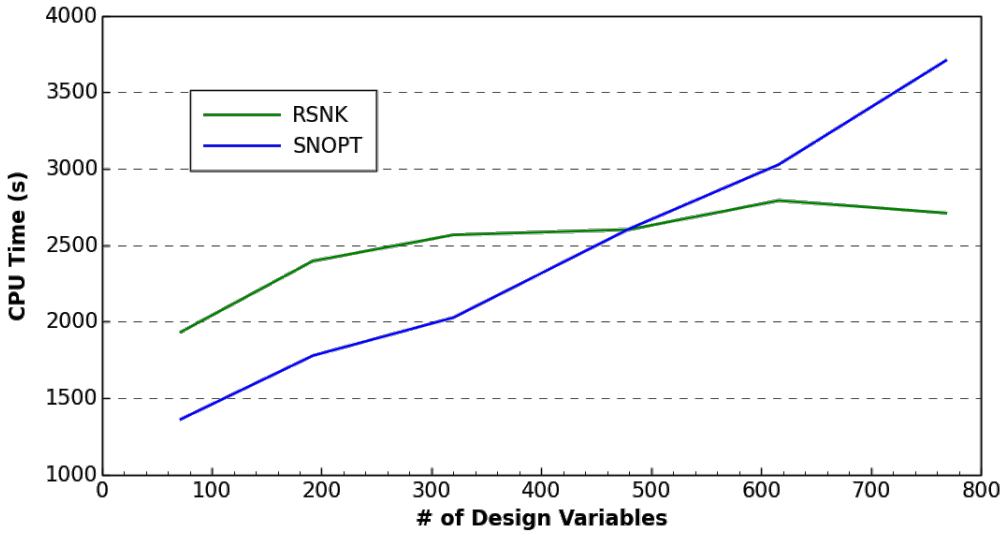


Fig. 2.6. Computational cost scaling of RSNK and SNOPT with respect to the size of the design space.

Convergence results from Fig. A.1 are compiled into a direct comparison of cost scaling in Fig. 2.6. The results demonstrate that RSNK has favorable scaling

with respect to the dimensionality of the design space. Despite the loss of superlinearity discussed earlier, RSNK converges faster than SNOPT for 616 and 768 design variables, and remains competitive at 480. This demonstrates RSNK's potential to be an efficient algorithm for problems with thousands of design variables.

2.6 Summary

In this chapter, we have introduced a reduced-space inexact-Newton-Krylov (RSNK) algorithm intended for the efficient solution of PDE-governed optimization problems with large numbers of design variables and state-based constraints.

A unique challenge of this class of problem is a constraint Jacobian that is computationally too expensive to compute explicitly. Each row of the explicit Jacobian represents a total derivative of a single constraint with respect to the design variables, and would require a separate adjoint solution. To address this challenge, we have developed a matrix-free second-order adjoint-based product with the KKT matrix in Section 2.3.1. Using this method, each product with the KKT matrix can be computed using only two linear system solutions based on the PDE Jacobian regardless of how many state-based constraints there are in the problem.

This matrix-free product is used in conjunction with a Krylov iterative solver to solve the KKT system and compute the Newton step. Since the KKT system is an indefinite saddle-point system, we perform this solution using a specialized Krylov solver called FLECS [45], described in Section 2.3.2. The Newton step is then globalized in a trust-region framework, using a simple filter [48] to assess step-acceptance.

We have demonstrated the efficacy of this algorithm on a high-fidelity aerodynamic shape optimization problem. Our results demonstrate that the computational cost of the RSNK algorithm is relatively insensitive to the size of the design space, and compares favorably against SNOPT (a limited-memory quasi-Newton method) when the optimization problem has several hundred design variables.

CHAPTER 3
MATRIX-FREE PRECONDITIONER FOR THE IDF ARCHITECTURE

3.1 Introduction

A preconditioner is an operator that approximates the inverse of a target matrix and is relatively inexpensive to apply. The application of a preconditioner is meant to improve the conditioning of linear systems of equations, and thereby improve the rate of convergence of iterative methods used to solve these systems.

The reduced-space inexact-Newton-Krylov algorithm we have developed and introduced in Chapter 2 relies on a Krylov iterative solver to compute the Newton step at each optimization iteration. However, it is well known that saddle-point systems like (2.15) are ill-conditioned; see, for example, the review by Benzi *et al.* [56]. Therefore, any Krylov iterative method used to solve (2.15) will require an effective preconditioner in order to produce high-quality steps.

Unfortunately, common and generally applicable preconditioning methods, such as incomplete Cholesky or LU factorizations, are not suitable for the matrix-free solution the KKT system. These methods require the matrix to be available explicitly, and avoiding the explicit computation of the KKT matrix or its sub-blocks is a critical goal in the present work. To the best of our knowledge, no such matrix-free preconditioner exists in the literature for the primal-dual system in the IDF formulation.

Our approach to developing an effective IDF preconditioner was inspired by a preconditioner originally proposed by Biros and Ghattas for full-space PDE-constrained optimization [7]. The preconditioner in [7] [2] approximately inverts the linearized PDE Jacobian, thereby preconditioning the full-space problem by mimicking the reduced-space formulation. In this chapter, we will demonstrate that a

Portions of this chapter previously appeared as: A. Dener and J. E. Hicken, "Matrix-free algorithm for the optimization of multidisciplinary systems," *Structural and Multidisciplinary Optimization*, vol. 56, no. 6, pp. 1429–1446, Jun. 2017

[2] Specifically, the $\tilde{\mathsf{P}}_2$ preconditioner.

matrix-free preconditioner for the IDF problem can be constructed by approximately inverting the IDF coupling Jacobian, thereby preconditioning the IDF problem by mimicking the MDF formulation.

3.2 Constructing the Preconditioner

In the IDF formulation, for an arbitrary vector $\boldsymbol{b}$, we seek an operator M^{-1} such that $\mathsf{M}^{-1}\boldsymbol{b} \approx \mathsf{K}^{-1}\boldsymbol{b}$, where K is the KKT matrix in (2.15).

To identify such an operator, it is instructive to begin with the ideal case, where $\mathsf{M} = \mathsf{K}$. In this case, given $\boldsymbol{b}^T = [\boldsymbol{b}_{\boldsymbol{x}}^T, \boldsymbol{b}_{\bar{\boldsymbol{w}}}^T, \boldsymbol{b}_{\boldsymbol{\lambda}}^T]$, we must solve the following linear system:

$$\begin{bmatrix} \mathsf{H}_{\boldsymbol{xx}} & \mathsf{H}_{\bar{\boldsymbol{w}}\boldsymbol{x}} & \mathsf{A}_{\boldsymbol{x}}^T \\ \mathsf{H}_{\boldsymbol{x}\bar{\boldsymbol{w}}} & \mathsf{H}_{\bar{\boldsymbol{w}}\bar{\boldsymbol{w}}} & \mathsf{A}_{\bar{\boldsymbol{w}}}^T \\ \mathsf{A}_{\boldsymbol{x}} & \mathsf{A}_{\bar{\boldsymbol{w}}} & 0 \end{bmatrix} \begin{pmatrix} \boldsymbol{p}_{\boldsymbol{x}} \\ \boldsymbol{p}_{\bar{\boldsymbol{w}}} \\ \boldsymbol{p}_{\boldsymbol{\lambda}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_{\boldsymbol{x}} \\ \boldsymbol{b}_{\bar{\boldsymbol{w}}} \\ \boldsymbol{b}_{\boldsymbol{\lambda}} \end{pmatrix}, \tag{3.1}$$

where the subscripts on the Hessian and constraint Jacobian blocks denote the variables by which the Lagrangian and the IDF constraints are (total) differentiated, respectively. In addition, we have introduced $\bar{\boldsymbol{w}} \in \mathbb{R}^l$ for the vector of IDF coupling variables; in the two-discipline case this vector would be $\bar{\boldsymbol{w}}^T = [\bar{\boldsymbol{u}}^T \bar{\boldsymbol{v}}^T]$.

For our first step toward the preconditioner, we drop all the Hessian terms from K except for $\mathsf{H}_{\boldsymbol{xx}} = \nabla^2_{\boldsymbol{xx}}\mathcal{L}$, which is replaced by a symmetric positive-definite approximation B:

$$\begin{bmatrix} \mathsf{B} & 0 & \mathsf{A}_{\boldsymbol{x}}^T \\ 0 & 0 & \mathsf{A}_{\bar{\boldsymbol{w}}}^T \\ \mathsf{A}_{\boldsymbol{x}} & \mathsf{A}_{\bar{\boldsymbol{w}}} & 0 \end{bmatrix} \begin{pmatrix} \boldsymbol{p}_{\boldsymbol{x}} \\ \boldsymbol{p}_{\bar{\boldsymbol{w}}} \\ \boldsymbol{p}_{\boldsymbol{\lambda}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_{\boldsymbol{x}} \\ \boldsymbol{b}_{\bar{\boldsymbol{w}}} \\ \boldsymbol{b}_{\boldsymbol{\lambda}} \end{pmatrix}, \tag{3.2}$$

The invertibility of $\mathsf{A}_{\bar{\boldsymbol{w}}}$, which we will discuss later, allows us to perform a block factorization of the matrix in (3.2). Applying this block factorization, we arrive at the initial version of our preconditioner in Alg. 2.

To see how the IDF preconditioner mimics MDF, recall that the IDF formulation removes variables responsible for coupling between disciplines out of the linearized MDA, and instead solves those variables within the optimization algorithm via coupling constraints. However, these coupling constraints are guaranteed to be satisfied only at the optimum solution, and the intermediate solutions are not

Algorithm 2: Steps for the IDF preconditioner application.

Data: vector to be preconditioned $\boldsymbol{b} = (\boldsymbol{b_x}, \boldsymbol{b_{\bar{w}}}, \boldsymbol{b_\lambda})$
Result: preconditioned vector $\boldsymbol{p} = (\boldsymbol{p_x}, \boldsymbol{p_{\bar{w}}}, \boldsymbol{p_\lambda})$

1 solve $\mathsf{A}_{\bar{\boldsymbol{w}}}^T \boldsymbol{p_\lambda} = \boldsymbol{b_{\bar{w}}}$ for $\boldsymbol{p_\lambda}$
2 solve $\mathsf{B}\boldsymbol{p_x} = \boldsymbol{b_x} - \mathsf{A}_{\boldsymbol{x}}^T \boldsymbol{p_\lambda}$ for $\boldsymbol{p_x}$
3 solve $\mathsf{A}_{\bar{\boldsymbol{w}}} \boldsymbol{p_{\bar{w}}} = \boldsymbol{b_\lambda} - \mathsf{A}_{\boldsymbol{x}} \boldsymbol{p_x}$ for $\boldsymbol{p_{\bar{w}}}$

feasible with respect to the multidisciplinary problem. At these intermediate steps of the optimization, a solution with the IDF constraint Jacobian, $\mathsf{A}_{\bar{\boldsymbol{w}}} = d\mathcal{C}/d\bar{\boldsymbol{w}}$, effectively acts as a feasibility restoration on the coupling constraints, and can be used to recover solutions that satisfy the fully coupled linearized MDA.

For this reason, Steps 1 and 3 in Alg. 2 are essentially equivalent to solving the MDF adjoint and (linearized) state, respectively. For Step 2, the vector $\boldsymbol{b_x} - \mathsf{A}_{\boldsymbol{x}}^T \boldsymbol{p_\lambda}$ is analogous to the total gradient, with $\boldsymbol{b_x}$ playing the role of $\partial \mathcal{J}/\partial \boldsymbol{x}$ and the second term playing the role of $(\partial \mathcal{R}/\partial \boldsymbol{x})^T \boldsymbol{\Psi}$. Thus, in the MDF context, Step 2 finds the step in design space, where B would typically be some quasi-Newton approximation to the reduced Hessian. In a sense, it could be said that our preconditioner uses the coupling information already present in the MDF formulation to precondition the IDF formulation.

3.3 Invertibility of the IDF Constraint Jacobian

In the proposed IDF preconditioner Steps 1 and 3 hinge on the assumption that the IDF constraint Jacobian, $\mathsf{A}_{\bar{\boldsymbol{w}}} = d\mathcal{C}/d\bar{\boldsymbol{w}}$, is invertible. Therefore, in order to ensure a complete and generally applicable preconditioner for all IDF problems, we must also provide a general proof for this Jacobian's invertibility. To that end, we turn to the Theorem 1 below, which shows the IDF Jacobian is not only square but also nonsingular provided the MDF Jacobian and the discipline Jacobians are also nonsingular.

Theorem 1. *Let $\mathsf{K}_i = \partial \mathcal{R}_i/\partial \boldsymbol{w}_i$ be the Jacobian of the ith discipline's residual respect to its state variables. If K_i is nonsingular $\forall i$, then the IDF Jacobian $\mathsf{A}_{\bar{\boldsymbol{w}}} = d\mathcal{C}/d\bar{\boldsymbol{w}}$ is nonsingular if and only if the MDA Jacobian $\mathsf{A}_{\boldsymbol{w}} = \partial \mathcal{R}/\partial \boldsymbol{w}$ is nonsingular.*

The proof of Theorem 1 is a straightforward application of the Schur complement, which we will briefly review here. Given a $(p+q)\times(p+q)$ matrix,

$$\mathsf{M} = \begin{bmatrix} \mathsf{A} & \mathsf{B} \\ \mathsf{C} & \mathsf{D} \end{bmatrix}, \tag{3.3}$$

where $\mathsf{A} \in \mathbb{R}^{p\times p}$, $\mathsf{B} \in \mathbb{R}^{p\times q}$, $\mathsf{C} \in \mathbb{R}^{q\times p}$ and $\mathsf{D} \in \mathbb{R}^{q\times q}$ are the matrix sub-blocks, the Schur complement of M with respect to A or the $(1,1)$-block is defined as

$$\mathsf{M}/\mathsf{A} \equiv \mathsf{S}_{(1,1)} \coloneqq \mathsf{D} - \mathsf{C}\mathsf{A}^{-1}\mathsf{B}. \tag{3.4}$$

Similarly, the Schur complement of M with respect to D or the $(2,2)$ block is

$$\mathsf{M}/\mathsf{D} \equiv \mathsf{S}_{(2,2)} \coloneqq \mathsf{A} - \mathsf{B}\mathsf{D}^{-1}\mathsf{C}. \tag{3.5}$$

In this proof, we leverage the property that

$$\det(\mathsf{M}) = \det(\mathsf{A})\det(\mathsf{S}_{(1,1)}) = \det(\mathsf{D})\det(\mathsf{S}_{(2,2)}). \tag{3.6}$$

Consequently, if the diagonal sub-block D and the related Schur complement $\mathsf{S}_{(2,2)}$ are both invertible (i.e. $\det(\mathsf{D}) \neq 0$ and $\det(\mathsf{S}_{(2,2)}) \neq 0$), then M must also be invertible (i.e. $\det(\mathsf{M}) \neq 0$), and vice versa. The same relationship also holds true for the other diagonal sub-block A and the associated Schur complement $\mathsf{S}_{(1,1)}$.

Next, consider an MDA with d disciplines. Let the equation for the ith discipline be given by

$$\mathcal{R}_i\left(\boldsymbol{w}_i, \{\mathcal{E}_{ij}(\boldsymbol{w}_j) | j = 1,\ldots,d, j \neq i\}\right), \qquad \forall i = 1,\ldots,d, \tag{3.7}$$

where the set notation in the second argument indicates that the residual depends on the states of the other $d-1$ disciplines via the potentially nonlinear mappings $\mathcal{E}_{ij}$. Next, introduce the coupling variables, $\bar{\boldsymbol{w}}_{ij}$, which are defined by

$$\mathcal{C}_{ij}(\boldsymbol{w}_j, \bar{\boldsymbol{w}}_{ij}) \equiv \mathcal{E}_{ij}(\boldsymbol{w}_j) - \bar{\boldsymbol{w}}_{ij} = \boldsymbol{0}, \qquad \forall i, j = 1,\ldots,d,\ j \neq i. \tag{3.8}$$

Using the above conditions, the residual of the ith discipline can be expressed as a function of the coupling variables:

$$\mathcal{R}_i\left(\boldsymbol{w}_i, \{\bar{\boldsymbol{w}}_{ij} | j = 1, \ldots, d, j \neq i\}\right), \qquad \forall i = 1, \ldots, d. \tag{3.9}$$

Let $\boldsymbol{w}^T = [\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_d^T]$ and $\bar{\boldsymbol{w}}^T = [\bar{\boldsymbol{w}}_{12}^T, \ldots, \bar{\boldsymbol{w}}_{d,d-1}^T]$ denote the compound vectors containing all the states and coupling variables, respectively, and consider the compound residual given by

$$\begin{pmatrix} \mathcal{R}(\boldsymbol{w}, \bar{\boldsymbol{w}}) \\ \mathcal{C}(\boldsymbol{w}, \bar{\boldsymbol{w}}) \end{pmatrix} = \begin{pmatrix} \mathcal{R}_1(\boldsymbol{w}_1, \bar{\boldsymbol{w}}_{12}, \ldots, \bar{\boldsymbol{w}}_{1d}) \\ \vdots \\ \mathcal{R}_d(\boldsymbol{w}_d, \bar{\boldsymbol{w}}_{d1}, \ldots, \bar{\boldsymbol{w}}_{d,d-1}) \\ \mathcal{C}_{12}(\boldsymbol{w}_2, \bar{\boldsymbol{w}}_{12}) \\ \vdots \\ \mathcal{C}_{d,d-1}(\boldsymbol{w}_{d-1}, \bar{\boldsymbol{w}}_{d,d-1}) \end{pmatrix}. \tag{3.10}$$

The Jacobian of this compound residual, with respect to $\boldsymbol{w}$ and $\bar{\boldsymbol{w}}$, is the matrix

$$\mathsf{D} \equiv \begin{bmatrix} \frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} & \frac{\partial \mathcal{R}}{\partial \bar{\boldsymbol{w}}} \\ \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}} & \frac{\partial \mathcal{C}}{\partial \bar{\boldsymbol{w}}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} & \frac{\partial \mathcal{R}}{\partial \bar{\boldsymbol{w}}} \\ \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}} & -\mathsf{I} \end{bmatrix}. \tag{3.11}$$

The (2,2) block of D is the identity matrix, which is clearly invertible, and the (1,1) block of D is the block diagonal matrix $\mathsf{diag}(\mathsf{K}_1, \ldots, \mathsf{K}_d)$, which is also invertible by assumption that the K_i are invertible. Therefore, we can form the Schur complement of D with respect to either block.

The Schur complement with respect to the (2,2) block of D is the matrix

$$\mathsf{S}_{(2,2)} \equiv \frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} + \frac{\partial \mathcal{R}}{\partial \bar{\boldsymbol{w}}} \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}. \tag{3.12}$$

Now, the $(i,j)^{th}$ discipline-block of second term above is, for $i \neq j$,

$$\left(\frac{\partial \mathcal{R}}{\partial \bar{\boldsymbol{w}}} \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}\right)_{i,j} = \frac{\partial \mathcal{R}_i}{\partial \bar{\boldsymbol{w}}_{ij}} \frac{\partial \mathcal{E}_{ij}}{\partial \boldsymbol{w}_j} = \frac{\partial \mathcal{R}_i}{\partial \boldsymbol{w}_j}, \tag{3.13}$$

where the last equality follows from the chain rule. There is no contribution to the (i,i) block from the second term, since there is no coupling variable between a discipline and itself. Conversely, recall that the first term in the Schur complement $\mathsf{S}_{(2,2)}$ is $\mathsf{K}_i = \partial\mathcal{R}_i/\partial\boldsymbol{w}_i$, which has no off-diagonal blocks. Summarizing, the Schur complement with respect to the (2,2) block of D is the monolithic MDA Jacobian

$$\mathsf{S}_{(2,2)} = \left(\frac{\partial\mathcal{R}}{\partial\boldsymbol{w}} + \frac{\partial\mathcal{R}}{\partial\bar{\boldsymbol{w}}}\frac{\partial\mathcal{C}}{\partial\boldsymbol{w}}\right)_{i,j} = \frac{\partial\mathcal{R}_i}{\partial\boldsymbol{w}_j}. \tag{3.14}$$

Next, we consider the Schur complement of D with respect to the (1,1) block:

$$\mathsf{S}_{(1,1)} \equiv -\mathsf{I} - \frac{\partial\mathcal{C}}{\partial\boldsymbol{w}}\left[\frac{\partial\mathcal{R}}{\partial\boldsymbol{w}}\right]^{-1}\frac{\partial\mathcal{R}}{\partial\bar{\boldsymbol{w}}}. \tag{3.15}$$

Now, the direct sensitivities of $\boldsymbol{w}_i$ with respect to $\bar{\boldsymbol{w}}_{ij}$ can be found by taking the total derivative of $\mathcal{R}_i = \mathbf{0}$:

$$\begin{aligned} &\frac{d\mathcal{R}_i}{d\bar{\boldsymbol{w}}_{ij}} = \frac{\partial\mathcal{R}_i}{\partial\bar{\boldsymbol{w}}_{ij}} + \frac{\partial\mathcal{R}_i}{\partial\boldsymbol{w}_i}\frac{d\boldsymbol{w}_i}{d\bar{\boldsymbol{w}}_{ij}} = 0 \\ \Rightarrow \qquad &\frac{d\boldsymbol{w}_i}{d\bar{\boldsymbol{w}}_{ij}} = -\left[\frac{\partial\mathcal{R}_i}{\partial\boldsymbol{w}_i}\right]^{-1}\frac{\partial\mathcal{R}_i}{\partial\bar{\boldsymbol{w}}_{ij}}, \end{aligned} \tag{3.16}$$

where $j \neq i$ as before. Consequently, substituting this expression, the Schur complement with respect to the (1,1) block of D simplifies to

$$\mathsf{S}_{(1,1)} = -\mathsf{I} + \frac{\partial\mathcal{C}}{\partial\boldsymbol{w}}\frac{d\boldsymbol{w}}{d\bar{\boldsymbol{w}}} = \frac{\partial\mathcal{C}}{\partial\bar{\boldsymbol{w}}} + \frac{\partial\mathcal{C}}{\partial\boldsymbol{w}}\frac{d\boldsymbol{w}}{d\bar{\boldsymbol{w}}} = \frac{d\mathcal{C}}{d\bar{\boldsymbol{w}}}, \tag{3.17}$$

which is the total derivative of the coupling constraints with respect to the coupling variables, i.e. the IDF Jacobian.

The desired result now follows. The $(2,2)$ block of D is the identity matrix, and therefore invertible. If the MDA Jacobian, i.e. the Schur complement $\mathsf{S}_{(2,2)}$ is also invertible, then D must be invertible. If D is invertible, then the IDF Jacobian must be invertible, since it is the Schur complement $\mathsf{S}_{(1,1)}$. The reverse implication is analogous. This completes the proof.

3.4 Nested Linear Solutions

We have shown that the IDF constraint Jacobian is invertible. However, recall that the rows of $\mathsf{A}_{\bar{\boldsymbol{w}}}$ consist of total derivatives, so forming this matrix explicitly and inverting via direct methods is not feasible in practice. Therefore, to solve the linear systems in Steps 1 and 3 of Alg. 2, we again turn to Krylov iterative methods.

For simplicity, consider the system in Step 3 above in the context of a two-discipline problem. A Krylov method used to solve this system requires products between $\mathsf{A}_{\bar{\boldsymbol{w}}}$ and an arbitrary vector $\bar{\boldsymbol{z}}^T = [\bar{\boldsymbol{z}}_{\boldsymbol{u}}^T, \bar{\boldsymbol{z}}_{\boldsymbol{v}}^T]$. Such a product can be expressed as (recall $\mathsf{A}_{\bar{\boldsymbol{w}}} = d\mathcal{C}/d\bar{\boldsymbol{w}}$ where $\mathcal{C}$ is defined by (2.2))

$$\begin{aligned} \mathsf{A}_{\bar{\boldsymbol{w}}}\bar{\boldsymbol{z}} &= \begin{bmatrix} -\mathsf{I} & \frac{\partial \mathcal{E}_{\boldsymbol{v}}}{\partial \boldsymbol{u}} \ \frac{d\boldsymbol{u}}{d\bar{\boldsymbol{v}}} \\ \frac{\partial \mathcal{E}_{\boldsymbol{u}}}{\partial \boldsymbol{v}} \ \frac{d\boldsymbol{v}}{d\bar{\boldsymbol{u}}} & -\mathsf{I} \end{bmatrix} \begin{pmatrix} \bar{\boldsymbol{z}}_{\boldsymbol{u}} \\ \bar{\boldsymbol{z}}_{\boldsymbol{v}} \end{pmatrix} \\ &= \begin{bmatrix} -\bar{\boldsymbol{z}}_{\boldsymbol{u}} - \frac{\partial \mathcal{E}_{\boldsymbol{v}}}{\partial \boldsymbol{u}} \left(\frac{\partial \mathcal{R}_{\boldsymbol{u}}}{\partial \boldsymbol{u}} \right)^{-1} \frac{\partial \mathcal{R}_{\boldsymbol{u}}}{\partial \bar{\boldsymbol{v}}} \bar{\boldsymbol{z}}_{\boldsymbol{v}} \\ -\bar{\boldsymbol{z}}_{\boldsymbol{v}} - \frac{\partial \mathcal{E}_{\boldsymbol{u}}}{\partial \boldsymbol{v}} \left(\frac{\partial \mathcal{R}_{\boldsymbol{v}}}{\partial \boldsymbol{v}} \right)^{-1} \frac{\partial \mathcal{R}_{\boldsymbol{v}}}{\partial \bar{\boldsymbol{w}}} \bar{\boldsymbol{z}}_{\boldsymbol{w}} \end{bmatrix}, \end{aligned} \tag{3.18}$$

where we have replaced the total derivatives $d\boldsymbol{u}/d\bar{\boldsymbol{v}}$ and $d\boldsymbol{v}/d\bar{\boldsymbol{u}}$ with their definitions. The above expression shows that each product with $\mathsf{A}_{\bar{\boldsymbol{w}}}$ requires the inversion of the discipline Jacobians. While these inversions are decoupled from one another, their cost is unnecessarily high given the other approximations inherent to the preconditioner (e.g. approximating and discarding the Hessian blocks).

Instead, we replace the inverse Jacobians with approximate inversions induced by the discipline preconditioners; that is, we assume that each discipline has a preconditioner it uses to solve its PDE system, and that we can apply these preconditioners to arbitrary vectors. Thus, the products with $\mathsf{A}_{\bar{\boldsymbol{w}}}$ are approximated as

$$\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}\bar{\boldsymbol{z}} = \begin{bmatrix} -\bar{\boldsymbol{z}}_{\boldsymbol{u}} - \frac{\partial \mathcal{E}_{\boldsymbol{v}}}{\partial \boldsymbol{u}} \widetilde{\left(\frac{\partial \mathcal{R}_{\boldsymbol{u}}}{\partial \boldsymbol{u}} \right)^{-1}} \frac{\partial \mathcal{R}_{\boldsymbol{u}}}{\partial \bar{\boldsymbol{v}}} \bar{\boldsymbol{z}}_{\boldsymbol{v}} \\ -\bar{\boldsymbol{z}}_{\boldsymbol{v}} - \frac{\partial \mathcal{E}_{\boldsymbol{u}}}{\partial \boldsymbol{v}} \widetilde{\left(\frac{\partial \mathcal{R}_{\boldsymbol{v}}}{\partial \boldsymbol{v}} \right)^{-1}} \frac{\partial \mathcal{R}_{\boldsymbol{v}}}{\partial \bar{\boldsymbol{w}}} \bar{\boldsymbol{z}}_{\boldsymbol{w}} \end{bmatrix}, \tag{3.19}$$

where the tilde accent denotes an approximate matrix. We note that stationary PDE preconditioners are necessary in this context, otherwise the products with $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}$ will change from one iteration to the next and may cause the Krylov solver to fail.

A similar approach to the one described above is used to form the products

with $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}^T$, which are needed by Krylov iterative methods to solve the system in Step 1 of the IDF preconditioner. In addition, this general approach is used to form products with $\tilde{\mathsf{A}}_{\boldsymbol{x}}^T$ and $\tilde{\mathsf{A}}_{\boldsymbol{x}}$, which are needed in steps 2 and 3, respectively. For the transposed matrices, the only significant difference is that these products rely on the transpose of the PDE preconditioners.

To solve the systems involving $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}^T$ and $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}$ in Steps 1 and 3, we use unpreconditioned GMRES; unlike the KKT matrix in (2.15), our experience suggests that the IDF Jacobian is well conditioned and preconditioning is not necessary. Indeed, in the numerical experiments in Section 3.6 and Chapter 5, we find that approximately 5 iterations are typically needed to reduce the relative residual by a factor of 10^{-2} on problems with $\mathcal{O}(10^2)$ coupling constraints.

3.5 Preconditioner Overview

We have introduced a matrix-free preconditioner for the IDF formulation, which is given by the following steps:

1. Inexactly solve $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}^T \boldsymbol{p}_{\boldsymbol{\lambda}} = \boldsymbol{b}_{\bar{\boldsymbol{w}}}$ for $\boldsymbol{p}_{\boldsymbol{\lambda}}$ using GMRES, where the necessary matrix-vector products with $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}^T$ are evaluated using the transposed variant of (3.19).

2. Solve $\mathsf{B}\boldsymbol{p}_{\boldsymbol{x}} = \boldsymbol{b}_{\boldsymbol{x}} - \tilde{\mathsf{A}}_{\boldsymbol{x}}^T \boldsymbol{p}_{\boldsymbol{\lambda}}$ for $\boldsymbol{p}_{\boldsymbol{x}}$, where the product $\tilde{\mathsf{A}}_{\boldsymbol{x}}^T \boldsymbol{p}_{\boldsymbol{\lambda}}$ is evaluated using the $\tilde{\mathsf{A}}_{\boldsymbol{x}}^T$ variant of (3.19).

3. Solve $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}} \boldsymbol{p}_{\bar{\boldsymbol{w}}} = \boldsymbol{b}_{\boldsymbol{\lambda}} - \tilde{\mathsf{A}}_{\boldsymbol{x}} \boldsymbol{p}_{\boldsymbol{x}}$ for $\boldsymbol{p}_{\bar{\boldsymbol{w}}}$ using GMRES, where the matrix-vector products with $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}$ are evaluated using (3.19), and $\tilde{\mathsf{A}}_{\boldsymbol{x}} \boldsymbol{p}_{\boldsymbol{x}}$ is evaluated using the $\tilde{\mathsf{A}}_{\boldsymbol{x}}$ variant of (3.19).

The remaining unspecified detail of the IDF preconditioner is the approximate Hessian B appearing in Step 2. In the present work, we simply take $\mathsf{B} = \mathsf{I}$. More generally, a quasi-Newton approximation of the Hessian could be used for B, but we did not find any significant improvement using, e.g., BFGS in our experiments.

For implementation considerations, many modern open-source PDE solvers provide, or can be made to provide, the PDE preconditioner and Jacobian product subroutines necessary for the IDF preconditioner, e.g. Trilinos [57]. For some

legacy (linear) structural codes, it may be possible to use the exact stiffness matrix to compute products with $\tilde{\mathsf{A}}_{\bar{\boldsymbol{w}}}$ without significant cost. In other cases, computational cost may be mitigated by using approximate solutions with loose tolerances. More generally, however, we acknowledge that the operations required by the IDF preconditioner may prevent its use with some legacy solvers.

3.6 Numerical Examples

To test the efficacy of our matrix-free IDF preconditioner, we implement two low-fidelity multidisciplinary design optimization problems – a domain decomposition problem based on the 2-D Poisson equation, and a 2-D elastic nozzle problem. These problems are implemented in both the MDF and IDF formulations, and solved using inexact-Newton-Krylov algorithms. The MDF versions of these problems lack constraints; consequently, the appropriate Newton-Krylov algorithm is a Newton-CG method [24], utilizing second-order adjoint-based matrix-vector products of the Hessian. Meanwhile, the IDF versions are solved using the RSNK algorithm introduced in Chapter 2.

3.6.1 Domain Decomposition as a Model MDO Problem

Our first numerical example is an inverse problem based on the two-dimensional Laplace equation. The optimization problem is to determine the boundary value on the left edge that produces a target solution along the right edge. To mimic a multidisciplinary system, we use a domain decomposition approach in which the Laplace equation is solved independently on each subdomain. These independent subdomains are intended to mimic different "disciplines" in an MDO problem and are coupled together at the boundaries using consistency constraints imposed as Dirichlet conditions. Using this model MDO problem, we can investigate the IDF formulation with varying numbers of "disciplines".

The 2-dimensional Laplace equation on a unit domain is given by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad \forall\, (x, y) \in \Omega = [0, 1]^2 \,. \tag{3.20}$$

The boundary conditions for this test case are given by

$$\begin{aligned} u(x,y) &= 0, \quad \forall\ (x,y) \in \left(\partial\Omega|_{top} \cup \partial\Omega|_{bottom}\right), \\ u(0,y) &= d(y), \quad \forall\ y \in \partial\Omega|_{left}, \\ \frac{\partial u}{\partial x}\Big|_{x=1} &= 0, \quad \forall\ y \in \partial\Omega|_{right}, \end{aligned} \tag{3.21}$$

where the Dirichlet boundary on the left is defined by the control $d(y)$. The Laplace equation is discretized using a second-order accurate finite-difference scheme over a uniform grid. Based on this discretization, we use $\boldsymbol{d}_i = d(y_i)$ to denote the ith design variable associated with the Dirichlet condition imposed at the left boundary node with y-coordinate y_i.

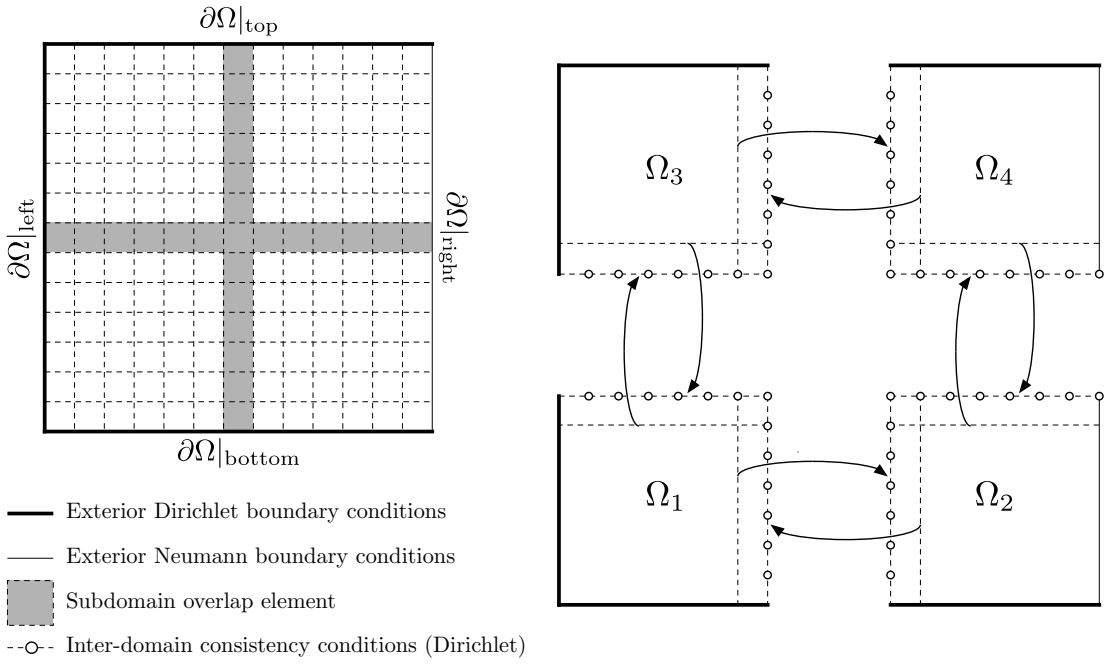


Fig. 3.1. Four-subdomain example of the domain decomposition used for the Laplace problem.

The unit domain with the boundary conditions specified in (3.21) is divided into equally sized subdomains, for example, as shown by the four-subdomain example in Fig. 3.1. As mentioned above, each subdomain is intended to mimic an individual "discipline" in a multidisciplinary problem. Decomposing the global

domain creates new inter-domain boundaries on which we must define boundary conditions, and it is through these conditions that we couple the "disciplines" together. Consequently, the definition of the boundary conditions depends on the MDAO formulation.

For the MDF architecture, the inter-domain boundary conditions are derived using the overlapping nodes shown in Fig. 3.1. The solution values of each domain that are one interval away from the edge are mapped onto the neighboring subdomain as a Dirichlet condition. An example expression of the inter-domain bounds is given as

$$\begin{aligned} u_1(x_1, y_{1,max}) &= u_3(x_1, y_{3,min} + \Delta y), \\ u_1(x_{1,max}, y_1) &= u_2(x_{2,min} + \Delta x, y_1), \end{aligned} \tag{3.22}$$

where Δx and Δy are the node spacing of the uniform grid, and the subscripts denote the domain numbers in the four-subdomain example from Fig. 3.1. In this instance, the row of nodes second from the bottom in Ω_3 has been mapped onto the top edge of Ω_1 as a Dirichlet condition. Similarly, the column of nodes second from the left in Ω_2 has been mapped onto the right edge of Ω_1. Together with the exterior boundary conditions defined in (3.21), Ω_1 now has all the information it needs for a well-posed solution of the Laplace equation. Again, these internal boundary conditions are intended to mimic the transfer of information between the coupled disciplines of a multidisciplinary system.

The multidisciplinary analysis (MDA) for the MDF architecture is then solved using a block-Jacobi approach. At each iteration of the MDA, the individual subdomains are solved independently, and the inter-domain Dirichlet conditions are updated before advancing to the next iteration. The multidisciplinary solution is considered converged when the maximum difference between overlapping nodes of neighboring subdomains falls below a specified tolerance.

Remark. *Stationary iterative methods, such as block-Jacobi, remain a popular solution method for multidisciplinary analysis [3], [58], [59] due to their ease of implementation when using disparate solvers. This choice is particularly appropriate for the present work, because MDF problems with block-Jacobi MDAs can be refor-*

mulated into IDF problems without necessitating any new solver infrastructure, and the two MDO architectures can be compared on equal footing in terms of implementation difficulty. We recognize that a monolithic Newton-Krylov approach can yield a more efficient solution of the multidisciplinary system [10], and we will explore this alternative further with our second test problem in Section 3.6.2.

In the IDF architecture, the inter-domain boundary conditions have the same form as MDF, but the values are prescribed by coupling variables; for example,

$$\begin{aligned} u_1(x_1, y_{1,max}) &= \bar{u}_{1,top}(x_1), \\ u_1(x_{1,max}, y_1) &= \bar{u}_{1,right}(y_1), \end{aligned} \tag{3.23}$$

where the IDF coupling variables, denoted by $\bar{u}$, are part of the optimization variables in the optimization problem. No multidisciplinary analysis is performed in the IDF formulation, and, instead, the multidisciplinary feasibility of the optimal solution is enforced through optimization constraints of the form

$$\begin{aligned} u_3(x_1, y_{3,min} + \Delta y) - \bar{u}_{1,top}(x_1) &= 0, \\ u_2(x_{2,min} + \Delta x, y_1) - \bar{u}_{1,right}(y_1) &= 0. \end{aligned} \tag{3.24}$$

Consequently, the subdomains in the IDF formulation do not communicate any boundary information to each other during the solution of the state equations, nor do they require communication during the adjoint solution.

3.6.1.1 Objective Function and Problem Setup

The objective function is the squared L^2 error between a target solution and the solution of the Laplace equation on the right edge of the full domain:

$$\mathcal{J} = \frac{1}{2} \int_0^1 (u(1, y) - u_{\text{targ}}(y))^2 dy, \tag{3.25}$$

where u_{targ} is the target solution along $\partial\Omega|_{\text{right}}$. Thus, the optimization seeks to recover the target solution through the control of the variables that define the Dirichlet boundary condition along the left edge. The target solution is the parabola $u_{\text{targ}}(y) = -4y(y-1)$, which has a maxima of 1 at $y = 0.5$ and complies with

zero Dirichlet boundary conditions at $y = 0$ and $y = 1$, to ensure the solution is continuous.

Table 3.1. Subdomain configurations tested with the Laplace problem.

Design Vars	Domains	Partition ($n_x \times n_y$)	State Vars	Coupling Vars
26	2	1×2	450	28
26	4	2×2	900	110
26	6	3×2	1350	192
39	6	2×3	1350	194
39	9	3×3	2025	332
39	12	4×3	2700	470

Table 3.1 lists all the domain-decomposition cases evaluated in the computational cost comparison. The variables n_x and n_y denote the number of subdomains along the x and y axes, respectively. Since the design variables are defined directly on the left-edge nodes, the number of design variables in the optimization problem changes with n_y. In the IDF formulation, the full size of the design space is expanded further by the coupling variables. The cases are designed such that the subdomain grid sizes are fixed at (15×15), including the overlapping nodes, for a total of 225 state variables for each "discipline". This ensures that the computational cost of an individual subdomain solution is a fixed constant for both architectures in all domain partitioning configurations, and the number of subdomain solutions can be used as a cost metric for optimization runs.

Table 3.2. Optimization parameters for the Laplace problem.

Parameter	Value
relative optimality tolerance, τ_p	10^{-5}
relative feasibility tolerance, τ_d	10^{-5}
initial penalty parameter, μ_0	10^5
initial Krylov tolerance, η_0	0.5
Krylov subspace size	20

Table 3.2 lists the algorithm parameters used for the solutions generated in this section. The initial and maximum trust radii vary for each test case to account for the differences in the size of the design space. The maximum radius is set as

$\Delta_{\max} = \sqrt{0.25\ n_d}$ and the initial radius as $\Delta_0 = \Delta_{\max}/8$ for both formulations, where n_d is the number of degrees of freedom for the optimization problem, including the coupling variables in the case of IDF.

3.6.1.2 IDF Preconditioner Results

Earlier in the chapter, we mentioned that the IDF preconditioner was critical in the practical application of the reduced-space Newton-Krylov algorithm to the IDF formulation. Without this preconditioner, the solution of the KKT system emerging from the IDF formulation stalls and fails to produce a useful search direction. To demonstrate this claim, Fig. 3.2 shows the convergence histories of the preconditioned and non-preconditioned FLECS solution of (2.15) in the four-subdomain (2×2) test case. The residual norm being measured is that of the linearized KKT conditions, and the Krylov subspace is capped at 20 vectors. The preconditioned solver is able to meet the tolerance target using only 10 iterations, while the non-preconditioned solver terminates at the iteration limit without any significant progress.

For the sample shown in Fig. 3.2, the preconditioned Krylov solution required 246 subdomain solution versus 160 for its non-preconditioned counterpart. However, the majority of the subdomain solutions in the preconditioned case are approximate. Each Krylov iteration for both cases requires one KKT-matrix-vector product, which in turn requires 8 subdomain solutions in this four-subdomain example (i.e.: 4 solutions for each second-order adjoint). This accounts for only 80 subdomain solutions in the preconditioned case. The remaining 166 subdomain solutions are, in fact, cheap approximations used in Steps 1 and 3 of the IDF preconditioner. We believe that this is a small price well worth paying, as this KKT system cannot be adequately solved otherwise.

3.6.1.3 Optimization Results

Fig. 3.3 shows the optimum solution for the four-subdomain case, with the overlap between subdomains shaded in gray. To plotting accuracy, both the MDF and IDF formulations recover the same optimum, matching the targeted parabola on the right edge of the domain at $x = 1$.

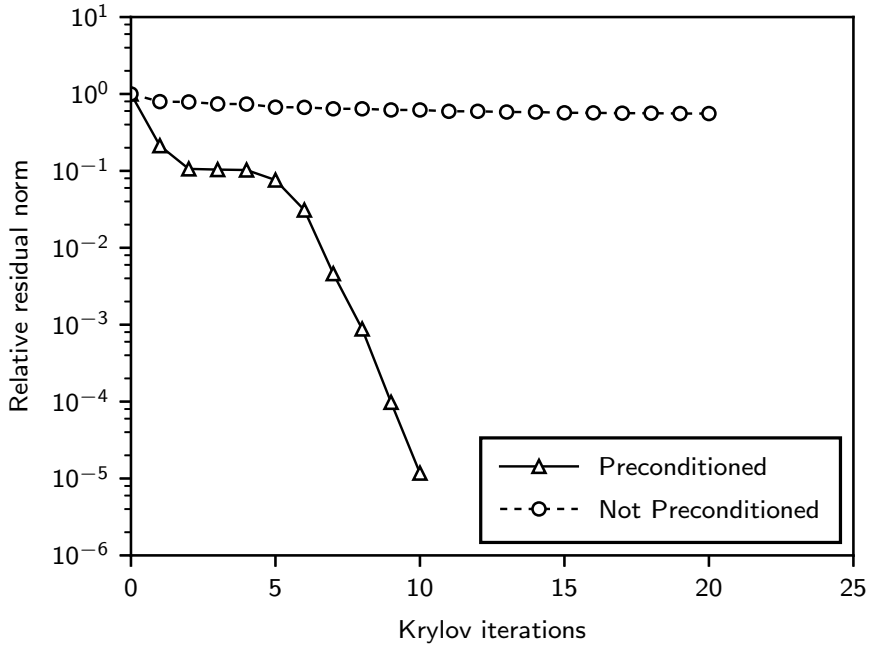


Fig. 3.2. IDF KKT system Krylov solution convergence with four-subdomain (2x2) decomposition.

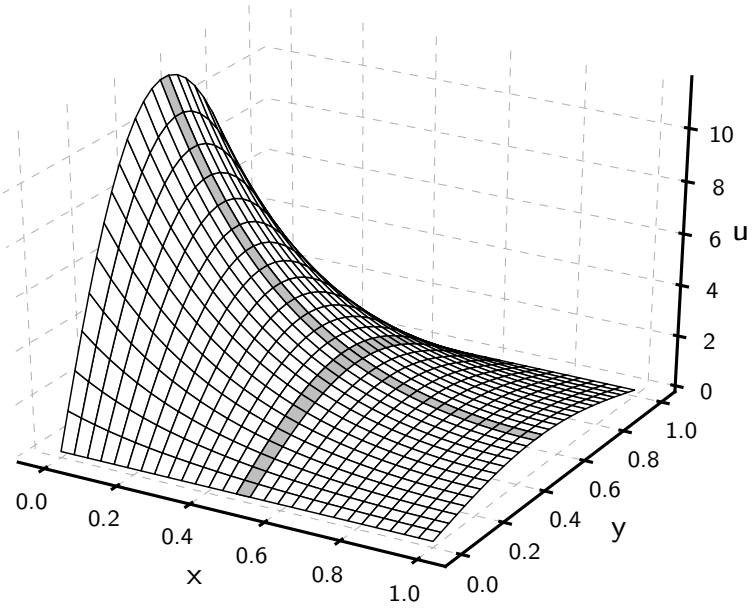


Fig. 3.3. Optimal solution with a four-subdomain (2x2) decomposition.

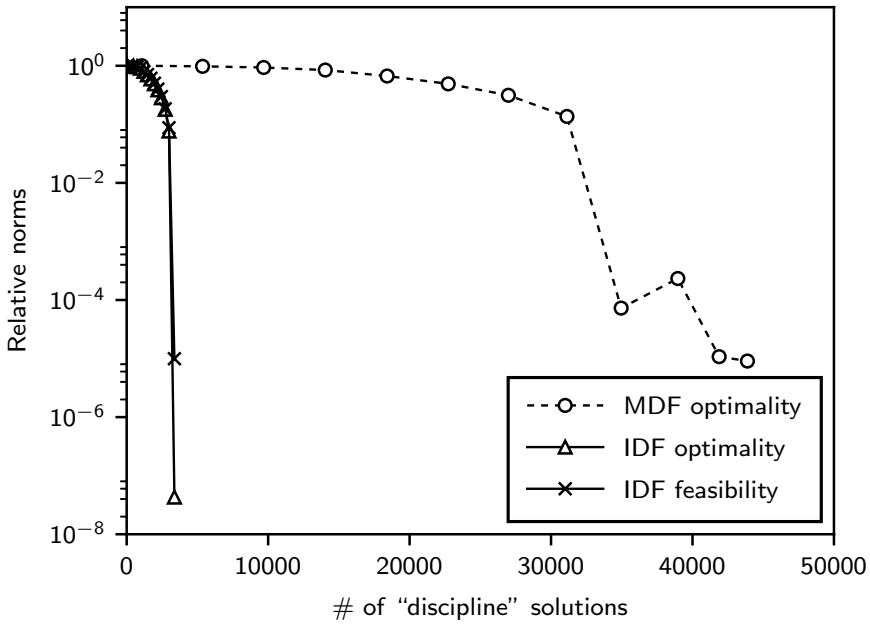


Fig. 3.4. Convergence histories for optimizations with four-subdomain (2x2) decomposition.

In order to demonstrate the superlienar convergence expected of Newton-Krylov algorithms, we plot the convergence history of the four-subdomain case in Fig. 3.4. The convergence metrics are the l^2-norms of the KKT conditions in (2.5), $\|d\mathcal{L}/d\boldsymbol{y}\|_2$ for optimality and $\|\mathcal{C}\|_2$ for feasibility, normalized by their initial values. The trends we present here are representative of all the subdomain configurations tested for this problem. The IDF formulation exhibits clear superlinear convergence on this problem, while the MDF formulation is close to superlinear except for the numerical artifacts at the end. This behavior is due to the dynamic Krylov tolerance that attempts to prevent oversolving near the optimum.

Additionally, we observe that the optimization for both architectures also takes approximately the same number of nonlinear iterations, but the overall computational cost of the MDF formulation for this problem is an order of magnitude more expensive than IDF. This is due to the forward and adjoint solutions in the MDF formulation requiring expensive block-Jacobi iterations to achieve full multidisciplinary convergence.

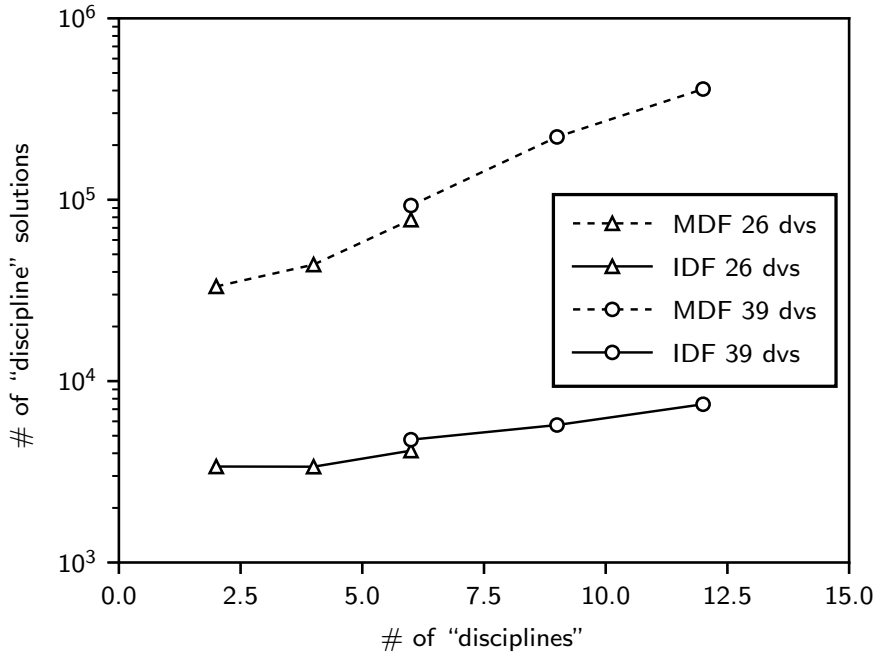


Fig. 3.5. Cost scaling with number of subdomain "disciplines" under IDF and MDF architectures.

The computational cost scaling against the number of subdomains is shown in Fig. 3.5. The trend-lines are separated per coupling architecture and number of design variables. It is clear that the MDF formulation is not only significantly more expensive on this problem, but it also scales poorly with increasing number of "disciplines".

3.6.2 Aero-Structural Optimization of a 2-D Elastic Nozzle

Our second test case is another inverse design problem, this time based on the quasi-one-dimensional flow in an elastic nozzle. The optimization problem is to find the (deformed) nozzle shape that recovers a prescribed pressure distribution. The flow is modeled using the quasi-one-dimensional Euler equations, and the structural deformations in the nozzle wall are modeled using combined beam-bending and axial stiffness.

3.6.2.1 Quasi-1D Flow Solver

The quasi-one-dimensional Euler equations are given by

$$\frac{\partial \mathcal{F}}{\partial x} - \mathcal{G} = 0, \quad \forall\, x \in [0,1], \tag{3.26}$$

where flux and source terms are defined by

$$\begin{aligned} \mathcal{F}^T &= \Big(\rho v A, \quad (\rho v^2 + p)A, \quad v(e+p)A\Big) \\ &\qquad\qquad \text{and} \\ \mathcal{G}^T &= \Big(0, \quad p\tfrac{dA}{dx}, \quad 0\Big), \end{aligned} \tag{3.27}$$

respectively. The state variables in the Euler equations are the density, ρ, momentum per unit volume, ρv, and energy per unit volume, e. The pressure is defined using the ideal gas law, $p = (\gamma - 1)(e - \frac{1}{2}\rho v^2)$. The flow solver is coupled to the structural solver through the nozzle area $A(x)$.

The boundary conditions are prescribed at the inlet, $x = 0$, and outlet, $x = 1$, using the Area-Mach-number relations, with a stagnation temperature of 300K and a pressure of 100 kPa. The specific gas constant and the critical nozzle area are set as 287 J/(kg K) and 0.8, respectively.

The spatial derivatives in (3.26) are discretized with a third-order accurate summation-by-parts operator [60], [61], with boundary conditions imposed weakly using penalty terms [62], [63]. The solution is stabilized with scalar third-order accurate artificial dissipation [64]. The resulting set of nonlinear algebraic equations are denoted by

$$\mathcal{R}_F(\boldsymbol{w}, \boldsymbol{u}) = \mathbf{0}, \tag{3.28}$$

where $\boldsymbol{w}$ are the flow states and the $\boldsymbol{u}$ are the structure deformations (defined below) that determine the nozzle area $A(x)$.

3.6.2.2 Linear Elastic Structural Solver

The structural deformations in the elastic nozzle wall are modeled using the direct stiffness method [65], [66]. The local stiffness system for frame elements is

given by

$$\mathsf{K}_{e'}\boldsymbol{u}_{e'} = \boldsymbol{f}_{e'}, \tag{3.29}$$

where the local stiffness matrix and degrees of freedom are defined as,

$$\underbrace{\frac{wtE}{l}\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & \frac{12t^2}{l^2} & \frac{6t^2}{l} & 0 & -\frac{12t^2}{l^2} & \frac{6t^2}{l} \\ 0 & \frac{6t^2}{l} & 4t^2 & 0 & -\frac{6t^2}{l} & 2t^2 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -\frac{12t^2}{l^2} & -\frac{6t^2}{l} & 0 & \frac{12t^2}{l^2} & -\frac{6t^2}{l} \\ 0 & \frac{6t^2}{l} & 2t^2 & 0 & -\frac{6t^2}{l} & 4t^2 \end{bmatrix}}_{\mathsf{K}_{e'}} \underbrace{\begin{pmatrix} u_{x',1} \\ u_{y',1} \\ u_{\theta,1} \\ u_{x',2} \\ u_{y',2} \\ u_{\theta,2} \end{pmatrix}}_{\boldsymbol{u}_{e'}}, \tag{3.30}$$

In (3.30), w is the cross-section width, t is the cross-section thickness, l is the axial length, and E is the Young's modulus of the element. The subscripts 1 and 2 denote the information that belongs to the left and right nodes of the element, respectively. The degrees of freedom $\boldsymbol{u}_{e'} = (u_{x'}, u_{y'}, u_\theta)^T$ are the local axial, local transverse and rotational deformations, respectively. Here, the local axial direction is parallel to the nozzle wall, while the local transverse direction is perpendicular.

(3.29) is coupled to the flow solver through the local forcing vector,

$$\boldsymbol{f}_{e'} = \begin{pmatrix} f_{x',1} & f_{y',1} & m_1 & f_{x',2} & f_{y',2} & m_1 \end{pmatrix}^T. \tag{3.31}$$

In the elastic nozzle problem, the only forces exerted on the structure stem from the pressure of the flow through the nozzle. Since pressure forces are always normal to the surface, the local axial forces $f_{x'}$ and applied moments m are always zero. The normal forces $f_{y'}$ are defined as

$$\begin{pmatrix} f_{y',1} \\ f_{y',2} \end{pmatrix} = -\frac{wl}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}, \tag{3.32}$$

where p_1 and p_2 are the flow pressures on the left and right nodes of the element.

The local stiffness system is transformed into the global Cartesian coordinate system to obtain

$$\mathsf{T}^T \mathsf{K}_{e'} \mathsf{T} \boldsymbol{u}_{e'} = \mathsf{T} \boldsymbol{f}_{e'}, \tag{3.33}$$

using the transformation matrix

$$\mathsf{T} = \begin{bmatrix} \cos\phi & \sin\phi & 0 & 0 & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\phi & \sin\phi & 0 \\ 0 & 0 & 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.34}$$

where ϕ is the angle between the frame element's axial centerline x' and the global x axis. The element systems in (3.33) are then assembled into a global system,

$$\mathcal{R}_S(\boldsymbol{u}, \boldsymbol{w}) = \mathsf{K}\boldsymbol{u} - \boldsymbol{f}(\boldsymbol{w}) = \boldsymbol{0}, \tag{3.35}$$

where $\boldsymbol{u}$ contains the deformations of the nozzle at each node, defined in the global (x, y, θ) directions, and $\boldsymbol{w}$ are the flow states defined earlier.

One of the assumptions inherent to the flow solver is that the areas are always defined at fixed locations along the nozzle. Therefore, in order to simplify the transfer of information between the two disciplines, we fix the global x-displacements for all nodes. Since the inlet and outlet areas are also fixed, we impose simple-support boundary conditions at the left and right ends of the nozzle, that is

$$u_x(0) = u_x(L) = u_y(0) = u_y(L) = 0.$$

3.6.2.3 Monolithic MDA and Adjoint Solutions

It is clear that the quasi-one-dimensional nozzle flow and structural displacement are coupled: the pressure determines the displacement, which changes the nozzle shape, which determines the pressure. To solve this coupled MDA, which

is necessary in MDF, we use a monolithic Newton-Krylov method. This iterative root-finding method solves the nonlinear discrete residual, denoted by the coupled discrete residual

$$\mathcal{R}(\boldsymbol{w}, \boldsymbol{u}) = \begin{pmatrix} \mathcal{R}_F(\boldsymbol{w}, \boldsymbol{u}) \\ \mathcal{R}_S(\boldsymbol{u}, \boldsymbol{w}) \end{pmatrix} = \mathbf{0}. \tag{3.36}$$

Applying Newton's method to (3.36) produces

$$\begin{bmatrix} \frac{\partial \mathcal{R}_F}{\partial \boldsymbol{w}} & \frac{\partial \mathcal{R}_F}{\partial \boldsymbol{u}} \\ \frac{\partial \mathcal{R}_S}{\partial \boldsymbol{w}} & \frac{\partial \mathcal{R}_S}{\partial \boldsymbol{u}} \end{bmatrix} \begin{pmatrix} \Delta \boldsymbol{w} \\ \Delta \boldsymbol{u} \end{pmatrix} = \begin{pmatrix} -\mathcal{R}_F \\ -\mathcal{R}_S \end{pmatrix}, \tag{3.37}$$

which is solved at each Newton iteration using FGMRES. This linear solution is preconditioned with a block-Jacobi approach, where the $(1, 1)$ flow block is solved using an LU factorization of a first-order accurate discretization based on nearest-neighbors, while the $(2, 2)$ structural block is preconditioned using nested FGMRES with an absolute tolerance of 10^{-5}. The nested FGMRES itself is preconditioned using the Gauss-Seidel method with 10 iterations. The MDA is converged to an absolute tolerance of 10^{-8}.

The adjoint system uses the transpose of the matrix in (3.37) and is solved using FMGRES with a relative tolerance of 10^{-8}. This solution is preconditioned with the transposed version of the block-Jacobi approach described above.

The IDF version of the problem solves the two disciplines independently. The discipline residuals are modified such that

$$\begin{pmatrix} \mathcal{R}_F(\boldsymbol{w}, \bar{\boldsymbol{u}}) \\ \mathcal{R}_S(\boldsymbol{u}, \bar{\boldsymbol{w}}) \end{pmatrix} = \mathbf{0}, \tag{3.38}$$

where $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{w}}$ are the target state variables prescribed by the optimization algorithm. In this particular problem, $\bar{\boldsymbol{u}}$ only consists of the transverse displacements at each node, $\boldsymbol{u}_y$, which are used to compute the nozzle areas for the flow solver. Similarly, $\bar{\boldsymbol{w}}$ only consists of the pressure at each nozzle node, $\boldsymbol{p}$, which are used to compute the structural forces. Under IDF, the disciplines are decoupled from

each other and can be evaluated independently. Forward and adjoint linear systems for IDF use only the diagonal blocks of the matrix in (3.37), and their associated preconditioners. Multidisciplinary feasibility is enforced in IDF using constraints,

$$\mathcal{E}_{\boldsymbol{w}}(\boldsymbol{u}) - \bar{\boldsymbol{u}} = \boldsymbol{0} \quad \text{and} \quad \mathcal{E}_{\boldsymbol{w}}(\boldsymbol{u}) - \bar{\boldsymbol{w}} = \boldsymbol{0}, \tag{3.39}$$

where $\mathcal{E}_{\boldsymbol{w}}(\boldsymbol{u}) = \boldsymbol{u}_y$ and $\mathcal{E}_{\boldsymbol{u}}(\boldsymbol{w}) = \boldsymbol{p}$.

3.6.2.4 Nozzle Geometry

The nozzle area, $A(x)$, is parameterized using a cubic b-spline with an open uniform knot vector. The interior b-spline control points are used as design variables in the optimization. The nozzle areas at the inlet and the outlet are fixed to be $A(0) = 2$ and $A(1) = 1.75$ respectively. The initial design variables are chosen such that they produce a linear nozzle area between the inlet and the outlet.

The displaced areas at each x-coordinate are calculated under the assumption of a rectangular nozzle cross-section with fixed width $w = 1\ m$ and varying height $h = 2[1 - (y_0 + \boldsymbol{u}_y)]$, where y_0 is the initial y-coordinate of each structural node and $\boldsymbol{u}_y$ are the vertical deformations computed by the structural solver. Finally, the nozzle lenght is fixed at $l = 1$ m and the Young's Modulus at $E = 10^9$ Pa.

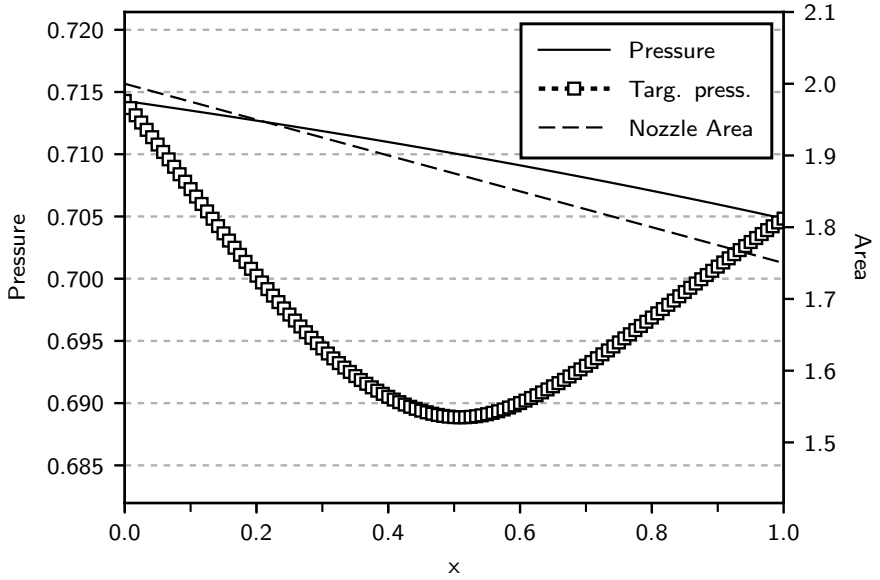
3.6.2.5 Objective Function and Problem Setup

The objective function for the nozzle inverse design problem is

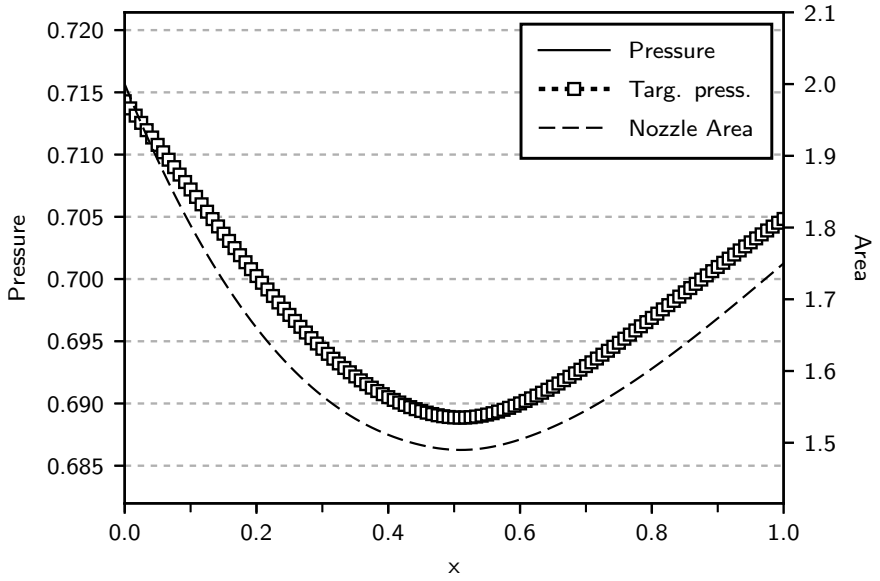
$$\mathcal{J} = \frac{1}{2}\int_0^1 (p - p_{targ})^2 dx, \tag{3.40}$$

where p_{targ} is the target pressure distribution. The discrete integral is calculated with the same fourth-order accurate SBP quadrature used in the flow solver [67]. The target pressures are calculated from the MDA solution where the initial (i.e., unloaded) nozzle has a cubic dependence on x, such that the minimum area, $A(0.5) = 1.5$ occurs at the midpoint of the nozzle. Fig. 3.6 shows the initial and optimal nozzle shapes, together with the corresponding pressure distributions. The target pressure is also provided and is seen to match the computed pressure on the optimal

nozzle.



(a) initial



(b) optimized

Fig. 3.6. Deformed nozzle areas and corresponding pressure distributions.

The nozzle problem was solved using the MDF and IDF formulations for varying numbers of design variables ranging from 10 to 40 in increments of 2. Note

that the x-axis in Fig. 3.8 only shows the number of b-spline control points: the IDF formulation for this problem has 242 additional coupling variables, and just as many Lagrange multipliers, that are also variables in the optimization algorithm.

A complete list of optimization parameters used in this problem is provided in Table 3.3.

Table 3.3. Optimization parameters for the elastic nozzle problem.

Parameter	Value
relative optimality tolerance, τ_p	10^{-5}
relative feasibility tolerance, τ_d	10^{-5}
initial trust radius, Δ_0	2.0
maximum trust radius, $\Delta_{\max}$	4.0
initial penalty parameter, σ_0	0.1
initial Krylov tolerance, η_0	0.5
Krylov subspace size	15

3.6.2.6 IDF Preconditioner Results

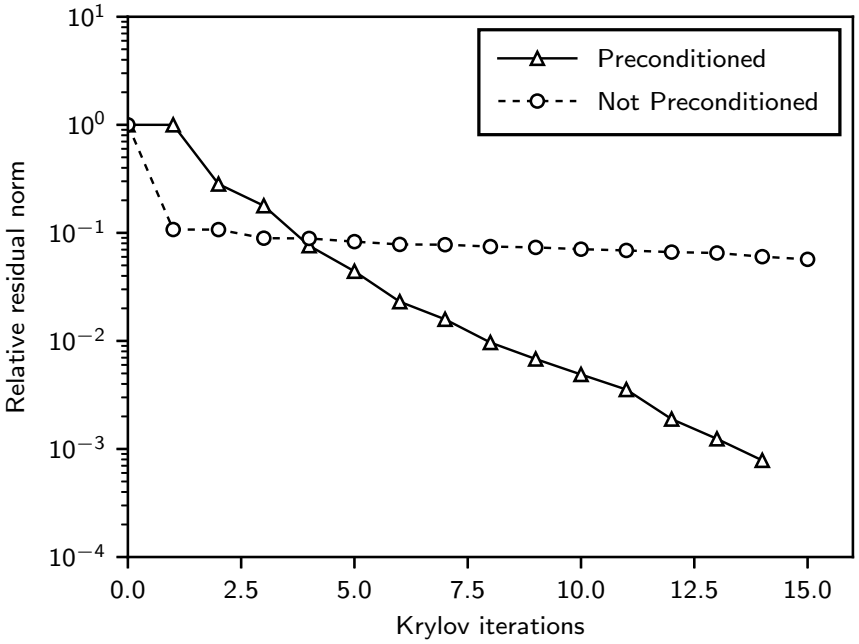


Fig. 3.7. A sample of FLECS convergence histories for the IDF problem with 20 design variables.

We begin the numerical experiments by demonstrating the efficacy of the IDF preconditioner, as we did for the domain-decomposition problem. Fig. 3.7 shows the convergence histories of the FLECS Krylov solver with 20 design variables, with and without the preconditioner. The sample presented here is representative of convergence histories typically observed throughout the optimization. Once again, the non-preconditioned solution stalls while the preconditioned solver is able to converge the relative KKT residual down three orders of magnitude for this particular nonlinear iteration. This mirrors our observations from the previous Laplace domain-decomposition problem. Without the preconditioner, the Krylov solver cannot produce useful Newton steps, and the optimization algorithm cannot make progress toward an optimum for the IDF problem.

3.6.2.7 Optimization Results

Fig. 3.8 shows the cost scaling of the two MDO architectures. The computational cost is measured by recording the number of flow solver preconditioner applications and normalizing this by the number of flow solver preconditioner applications required for the MDA on the target geometry.

The cost scaling analysis for the elastic nozzle problem reveals a number of important findings. First, the Newton-Krylov MDA makes the MDF formulation considerably more efficient than what we observed with the block-Jacobi MDA of the domain-decomposition problem. Consequently, the MDF formulation here costs 8.3 fewer equivalent MDA evaluations, on average, relative to IDF – a 15.5% decrease. However, the inexact-Newton-Krylov strategy appears to exhibit better cost scaling with the IDF formulation than it does with MDF. The trust-region Newton-CG algorithm used for the MDF problem experiences an increase in cost at more than 30 design variables. Overall, the IDF formulation remains competitive in cost while offering significant advantages in terms of implementation.

We evaluate optimization convergence properties with sample convergence histories taken from the 20 and 40 design-variable cases, shown in Fig. 3.9. The inexact-Newton-Krylov method for IDF exhibits the expected superlinear convergence for both cases, and this is typical of the entire range of design variables we have tested.

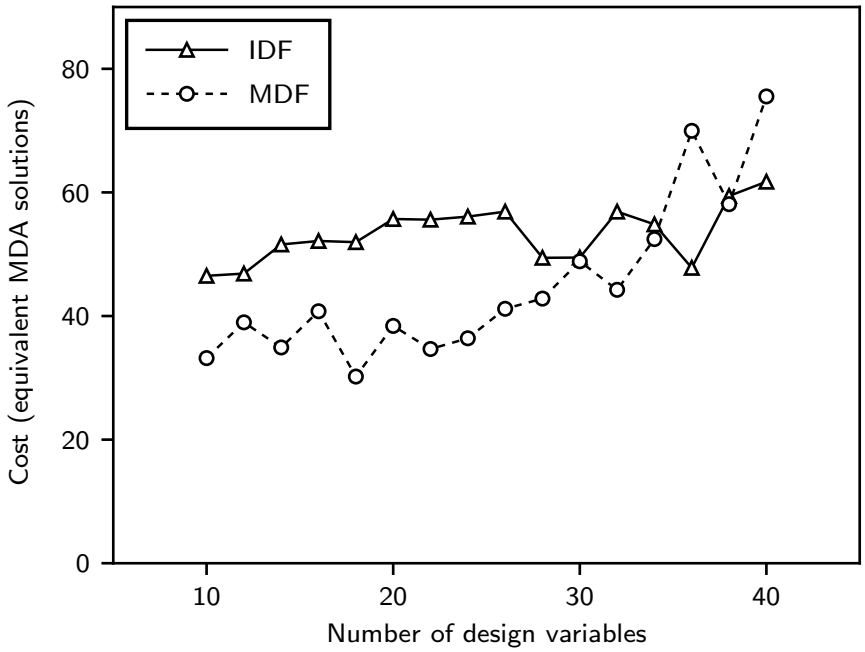
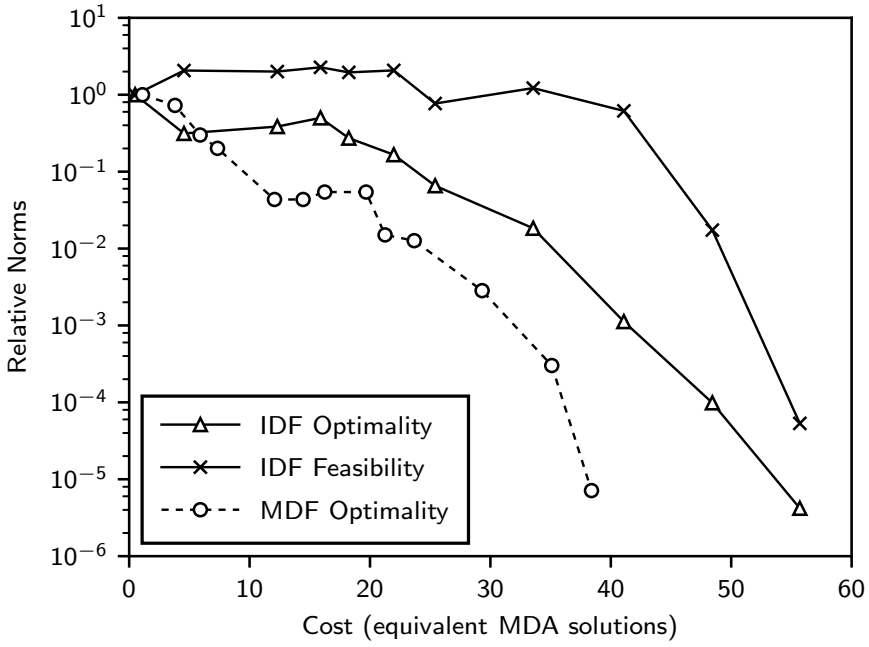


Fig. 3.8. Computational cost of the optimization with varying numbers of design variables.

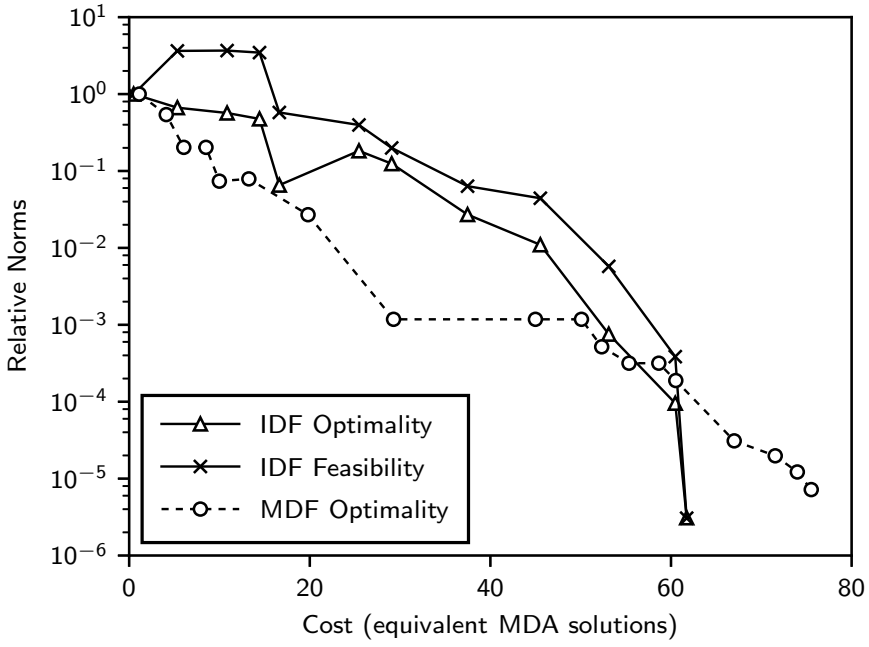
Convergence for the MDF formulation, however, loses superlinearity at higher numbers of design variables, indicating that the quality of the Newton step produced by the Krylov iterative method is degrading with the increasing size of the MDF optimization problem.

3.7 Summary

The RSNK algorithm we have introduced in Chapter 2 aims to solve reduced-space PDE-governed optimization problems by computing the Newton step by solving the KKT system with a Krylov solver. The advantage of this approach is that the solution can be performed matrix-free, without requiring the computationally expensive explicit assembly of a total constraint Jacobian. However, the KKT system that arises from applying the Newton's method to the first-order optimality conditions is an ill conditioned saddle-point system. Efficient matrix-free solution of this system requires an effective matrix-free preconditioner.



(a) 20 design variables



(b) 40 design variables

Fig. 3.9. Optimization convergence histories for the elastic nozzle problem.

In this chapter, we have taken inspiration from the full-space optimization community and developed a matrix-free preconditioner for the IDF formulation. Our preconditioner approximately inverts the IDF coupling constraint Jacobian, which is analogous to approximating the IDF coupling variables via an MDF analysis. This is accomplished through a three step process described in Section 3.2. Invertibility of the coupling constraint Jacobian is proven in Section 3.3, and implementation details of the nested Krylov solutions discussed in Section 3.4.

We have demonstrated the effectiveness of the IDF preconditioner on two low-fidelity test problems in Section 3.6. Our results indicate that the preconditioner is critical for the computation of high-quality descent directions in the IDF formulation. Without it, the Krylov solutions for the associated KKT systems do not converge. In Chapter 5, we will investigate whether this still holds true on a high-fidelity large-scale problem with thousands of coupling variables and constraints.

CHAPTER 4
PARALLEL-AGNOSTIC OPTIMIZATION LIBRARY

4.1 Introduction

In the present work, we have developed a reduced-space inexact-Newton-Krylov (RSNK) algorithm intended to make the modular IDF formulation a viable and efficient choice for multidisciplinary design optimization. However, the matrix-free KKT matrix-vector product introduced in Chapter 2 and the IDF preconditioner introduced in Chapter 3 involve PDE Jacobian products and linearized solutions that are often parallelized in high-fidelity applications. Since the choice of data structures and parallelization schemes differ from one PDE solver to another, any implementation of this RSNK algorithm must remain agnostic to the underlying data structures and parallelization schemes in order to maintain modularity.

In this chapter, we will introduce a parallel-agnostic optimization library, called Kona, that aims to preserve the sought-after modularity. The high-level design of Kona can be thought of in three components: the PDE solver interface, the optimization algorithm interface, and an abstraction layer that bridges the two together. It is important to stress that the optimization side never directly interacts with the solver side and vice versa, instead going through the abstraction layer. Our motivation for such a separation is to allow the development of new optimization algorithms and the integration of new PDE solvers independently from one another.

Figure 4.1 shows a minimal UML diagram for Kona. Here, `ISolver`, `IAllocator` and `IVector` define the solver side interfaces. `IAlgorithm` and `IHessian` define the optimization side interfaces. `Optimizer` is the top level optimization controller users are intended to interact with. Finally, `KonaMemory`, `VectorFactory`, `KonaVector` and `KonaMatrix` form the reverse-communication abstraction layer.

Portions of this chapter previously appeared as: A. Dener, P. Meng, J. E. Hicken, G. J. Kennedy, J. Hwang, and J. S. Gray, "Kona: A parallel optimization library for engineering-design problems," presented at the 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conf., San Diego, CA, USA, 2016

Below we will describe the design of each block individually, explaining in detail the components listed in the UML diagram. To facilitate this discussion, we introduce a generic equality-constrained, PDE-governed optimization problem statement.

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \mathcal{F}(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) \\ \text{subject to} \quad & \mathcal{C}(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) = 0 \\ \text{governed by} \quad & \mathcal{R}(\boldsymbol{x}, \boldsymbol{u}) = 0 \end{aligned} \tag{4.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ are the design variables and $\boldsymbol{u} \in \mathbb{R}^s$ are the state variables. The objective function, $\mathcal{F} : \mathbb{R}^n \times \mathbb{R}^s \to \mathbb{R}$, and constraints, $\mathcal{C} : \mathbb{R}^n \times \mathbb{R}^s \to \mathbb{R}^m$, are assumed to be C^2 continuous functions. In this reduced-space formulation, the state variables are defined as implicit functions of the design variables via a discretized set of PDEs, denoted here by $\mathcal{R}(\boldsymbol{x}, \boldsymbol{u})$. As in Chapter 2, this general optimization statement can also represent a multidisciplinary problem, with a set of coupled PDEs combined under a single multidisciplinary residual. However, defining the implementation on this general statement ensures the broad applicability of the optimization library to a wide range of problems.

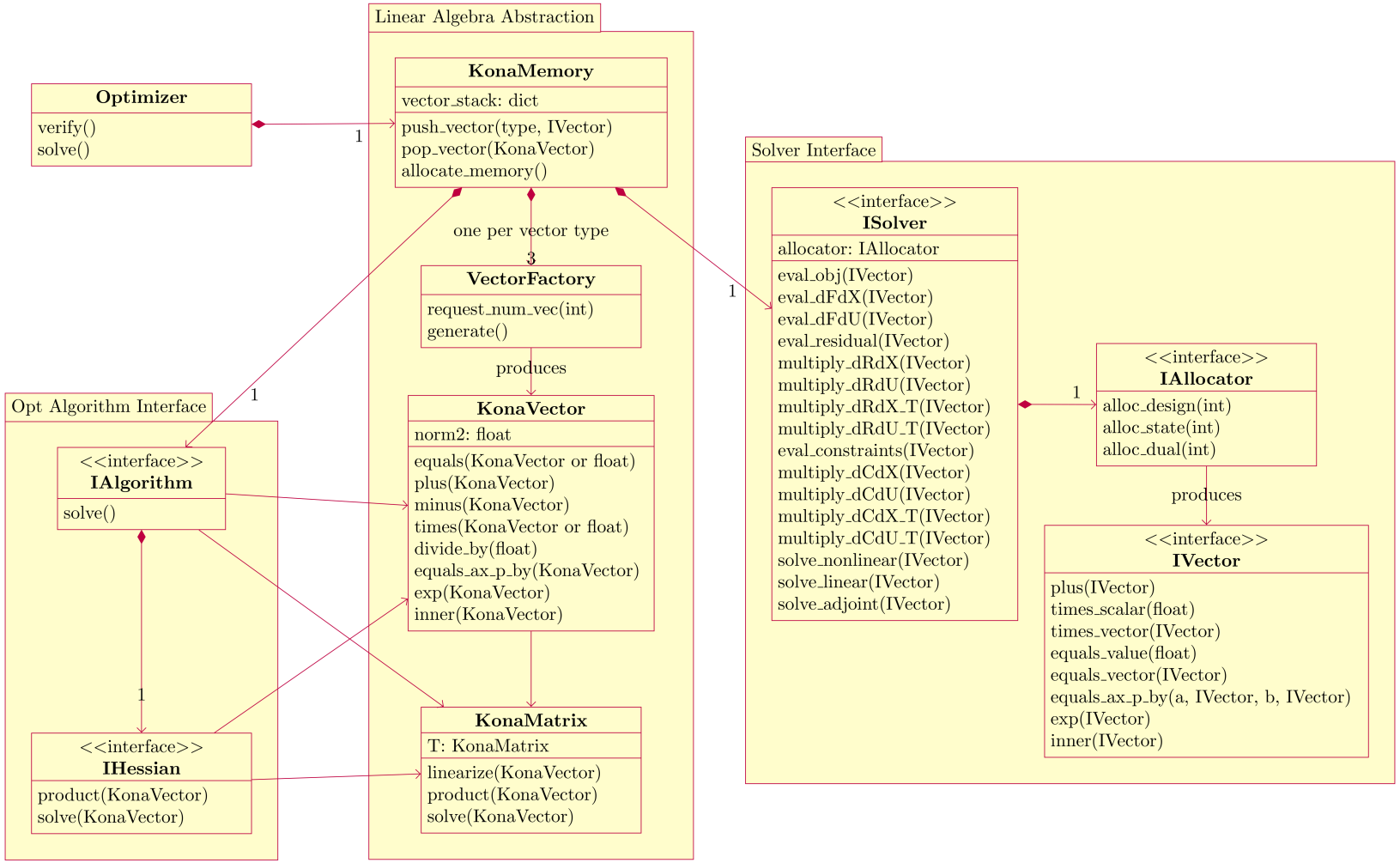


Fig. 4.1. Minimal UML diagram for Kona, showing only high-level associations.

4.2 PDE-Solver Interface

Kona's interaction with a PDE solver is inspired by a "reverse communication" feature first proposed by Ashby and Seager [69] in the context of iterative linear system solvers. The goal of reverse communication, as stated by Ashby and Seager, is to perform algebraic operations while avoiding data structures entirely, such that the user has the freedom to implement the data storage in whatever architecture they prefer, and adopt any parallelization scheme. Consequently, an algorithm using reverse communication never interacts with data directly. Instead, it sends requests to the user to perform certain linear algebra tasks on data stored in particular locations. This ensures the algorithm's operability with both MPI- and GPGPU-style parallelism, with no scheme-specific modifications to the optimization algorithms.

Kona adopts this reverse communication model by requiring users to create three objects that follow a predetermined interface.

- `IVector` – A state vector object implementing ten basic algebra tasks on the underlying data set.
- `IAllocator` – A memory allocation tool that will initialize a requested number of `IVector` objects in a given vector space.
- `ISolver` – A solver wrapper with a predetermined set of member functions performing function evaluations, partial-derivative evaluations, Jacobian-vector products, and linear and non-linear system solutions on `IVector` objects.

4.2.1 Abstract Vectors

User-created vectors are required to be an implementation of the `IVector` interface shown in the UML diagram in Figure 4.1. Currently, Kona supports vector abstraction only for state vectors. Design and dual vectors are internally implemented as serial arrays, under the assumption that their relative sizes in target applications are considerably smaller than the state space.

The `IVector` interface consists of eight specific algebra operations on the underlying data structure referenced within the abstract vector. These operations are:

- `plus(IVector)` – In-place summation with the given vector.

- `times_scalar(float)` – In-place multiplication with a scalar.
- `times_vector(IVector)` – In-place element-wise multiplication with the given vector.
- `equals_vector(IVector)` – Value assignment to the given vector.
- `equals_value(float)` – Value assignment to a scalar.
- `equals_ax_p_by(a, IVector, b, IVector)` – Value assignment to the scaled sum of two given vectors.
- `inner(IVector)` – In-place inner product with the given vector.

By default, the Kona library provides a basic serial implementation of these operations based on `NumPy ndarray` data storage. For high-performance applications, the choice of data structure within the abstract-vector object is left up to the user. However, it is strongly recommended to store and manipulate the data with a compiled language, in a parallelized environment, consistent with the PDE solver's data structure. This is facilitated by Python's interoperability with other languages; for example, Python can leverage compiled Fortran code using F2Py [70], or it can be extended to C/C++ using the native Python C API, Cython [71], or external libraries such as SWIG [72] and Boost.Python [73].

4.2.2 Vector Allocator

Once the abstract vector object is defined, Kona then requires a way to allocate the underlying data structure in bulk. This is achieved using an `IAllocator` interface with three member functions, one for each vector space, that produce standard Python arrays containing the requested number of user vectors.

As before, Kona provides a basic serial implementation of this called `BaseAllocator` that produces `BaseVectors` sized according to the requested vector space. The user is expected to define allocators for the vector objects associated with any user-implemented parallel data structure.

4.2.3 Solver Wrapper

The `ISolver` wrapper interface defines most of Kona's reverse communication scheme with the PDE solver. It contains a reference to the `IAllocator` implementation, and a set of predetermined member functions that perform various linear algebra tasks on `IVector` implementations.

Table 4.1 outlines all `ISolver` operations that must be implemented by the user. These methods are called upon by Kona's reverse-communication abstraction layer to perform various optimization tasks. For instance, when an algorithm needs to evaluate the total design derivative of the objective function, the abstraction layer will first evaluate the partial state derivative using `eval_dFdU` and then use this as the negative right-hand-side vector in a `solve_adjoint` call in order to calculate adjoint variables, as per (2.13). The matrix-vector product between the adjoint variables and $(\partial\mathcal{R}/\partial\boldsymbol{x})^T$, evaluated using `multiply_dRdX_T`, will then be added to the partial design derivative of the objective function, evaluated using `eval_dFdX`. The vector summations are performed via the algebra tools, also defined by the user, under the `IVector` implementation.

Kona provides an empty base class implementation of this interface, relying on the aforementioned base implementations for vector and allocator objects, intended to assist in the rapid development of simple test problems.

4.3 Linear Algebra Abstraction

The core functionality of Kona revolves around the idea of wrapping the user-provided `ISolver` and `IVector` implementations into abstract vectors and matrices, which are then used to write optimization algorithms with a simple, easy-to-read syntax. In order to accomplish this, the reverse-communication abstraction layer implements a memory manager that contains a "stack" of user-defined vector objects and factories that wrap these user-vectors into linear algebra objects used by optimization algorithms.

A very similar abstraction layer, named Thyra, has been developed as part of the Trilinos framework [74]. Thrya implements abstract vectors in different vector spaces and provides vector factories for their construction just as Kona does. In

Table 4.1. Operations defined by the `ISolver` interface.

Operation	Interface Calls
Function evaluations: evaluate the objective function, discrete PDE residual, and constraints at the design-state pair $(\boldsymbol{x}, \boldsymbol{u})$	
$\mathcal{F} = \mathcal{F}(\boldsymbol{x}, \boldsymbol{u}) \quad \mathcal{F} \in \mathbb{R}$ $\mathcal{R} = \mathcal{R}(\boldsymbol{x}, \boldsymbol{u}) \quad \mathcal{R} \in \mathbb{R}^s$ $\mathcal{C} = \mathcal{C}(\boldsymbol{x}, \boldsymbol{u}) \quad \mathcal{C} \in \mathbb{R}^m$	`eval_obj`, `eval_residual`, `eval_constraints`
Objective derivatives: evaluate the objective function partial derivatives at the design-state pair (x, u)	
$\boldsymbol{y} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{x}} \quad \boldsymbol{y} \in \mathbb{R}^n$	`eval_dFdX`,
$\boldsymbol{v} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{u}} \quad \boldsymbol{v} \in \mathbb{R}^s$	`eval_dFdU`
Matrix-vector products: evaluate Jacobian-vector products, linearized about (x, u)	
$\boldsymbol{v} = \dfrac{\partial \mathcal{R}}{\partial \boldsymbol{x}} \boldsymbol{y}, \quad \boldsymbol{y} = \left(\dfrac{\partial \mathcal{R}}{\partial \boldsymbol{x}} \right)^T \boldsymbol{v},$ $\boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{v} \in \mathbb{R}^s$	`multiply_dRdX`, `multiply_dRdX_T`
$\boldsymbol{v} = \dfrac{\partial \mathcal{R}}{\partial \boldsymbol{u}} \boldsymbol{w}, \quad \boldsymbol{w} = \left(\dfrac{\partial \mathcal{R}}{\partial \boldsymbol{u}} \right)^T \boldsymbol{v},$ $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^s$	`multiply_dRdU`, `multiply_dRdU_T`
$\boldsymbol{z} = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{x}} \boldsymbol{y}, \quad \boldsymbol{y} = \left(\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{x}} \right)^T \boldsymbol{z},$ $\boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{z} \in \mathbb{R}^m$	`multiply_dCdX`, `multiply_dCdX_T`
$\boldsymbol{z} = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{u}} \boldsymbol{w}, \quad \boldsymbol{w} = \left(\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{u}} \right)^T \boldsymbol{z},$ $\boldsymbol{w} \in \mathbb{R}^s, \boldsymbol{z} \in \mathbb{R}^m$	`multiply_dCdU`, `multiply_dCdU_T`
PDE solves: solve the following systems for u, w, and ϕ	
$R(x, u) = 0, \quad \left(\dfrac{\partial R}{\partial u} \right) w = u, \quad \left(\dfrac{\partial R}{\partial u} \right)^T \phi = v$ $v, w, u, \phi \in \mathbb{R}^s$	`solve_nonlinear`, `solve_linear`, `solve_adjoint`

place of abstract matrices, however, Thrya opts to implement linear operators that define matrix-related operations instead. This choice makes no significant difference in practice. Kona's abstract matrix objects do not contain any actual matrix data. They only house matrix-vector operations such as products and solutions (inverse products) which are ideally implemented matrix-free under the `ISolver` interface

4.3.1 Kona Memory Manager

`KonaMemory` is a memory manager object that contains a reference to the user-provided `ISolver` implementation, and it uses the `IAllocator` implementation within the solver to create a stack of `IVector` instances. These preallocated user-defined vectors are then wrapped into specialized Kona vectors and served to the optimization algorithms as needed. Neither the PDE solver nor the optimization algorithms are intended to interact with this memory manager. It is instantiated by the top level optimization controller, and entirely hidden from the users on either side of the abstraction layer.

`KonaMemory` implements three functions used to manage the stack:

- `push_vector(vec_type, IVector)` – Pushes the given user-defined `IVector` object onto the stack associated with the given vector space.
- `pop_vector(vec_type)` – Pops and returns a user-defined `IVector` object from the stack associated with the requested vector space.
- `allocate_memory()` – Uses the `IAllocator` implementation provided by the user to populate the vector stacks with a predetermined number of `IVector` objects.

It could be said that the memory manager simulates dynamic memory allocation for the optimization algorithms, while still using preallocated solver memory in practice. Whenever a new `KonaVector` is created, a user-defined vector of the appropriate vector space is popped off the memory stack and embedded into the Kona vector. Conversely, whenever an existing Kona vector is destroyed, the embedded user-defined vector is pushed onto the memory stack for later use. This scheme

emerges from the motivation to make the optimization algorithm syntax appear like pseudo-code, while also respecting good HPC coding practices by avoiding a large numbers of unnecessary and costly memory allocations and deallocations.

However, this design choice introduces a difficulty in counting the number of user-defined vectors that need to be preallocated ahead of the optimization. To solve this problem, we implement vector factories.

4.3.2 Vector Factories

Kona's vector factories are objects that serve two purposes: 1) tallying up vector allocation requirements for each vector space, and 2) generating specialized Kona vectors on-demand using preallocated user-defined vectors from the memory stack. Kona's memory manager creates three factories in total, one for each vector space – design, state and dual. These factories are then passed onto the optimization algorithm, allowing the algorithm and all its components to report their total data space requirements to the memory manager during initialization. The same factories are later used by the algorithm to "dynamically" produce Kona vectors as needed.

4.3.3 Kona Vectors

Kona internally implements a generalized vector object, `KonaVector`, that is used as a base class for all specialized vectors. This class contains fundamental algebra methods such as vector summation and subtraction, scalar multiplication and division, and inner products, common to all user-vector spaces. Each `KonaVector` produced by a vector factory also contains a unique `IVector` object taken off the internal memory stack. The algebra operations are then linked to the algebra implementation provided by the user as part of the `IVector` interface, allowing all `KonaVector` objects to perform vector operations without direct access to the underlying data, or any awareness of the storage and parallelization scheme adopted by the user.

A `KonaVector` by itself is not sufficient to perform optimization tasks. Three other classes, `DesignVector`, `StateVector` and `DualVector`, inherit from `KonaVector` and add specialized methods for their respective vector spaces. These specialized methods go through the memory manager and call upon solver methods,

implemented by the user as part of the `ISolver` interface, to perform various optimization tasks. Consequently, these three vectors are the common vector objects used by all optimization algorithms implemented in Kona.

Within the abstract vector module, Kona also introduces the concept of a composite vector. As the name suggests, these objects are composites of the three core vector objects described above. Unlike core vectors, composite vectors do not possess their own unique data space. Instead, they contain references to component vectors that make up the composite. For instance, a `ReducedKKTVector` in Kona is defined by the composite of a `DesignVector` and a `DualVector`. Any operations defined on this composite vector modifies the data space of the underlying core vectors. This mimics the mathematical notation of the RSNK algorithm we have developed in Chapter 2, where we perform linear algebra operations on primal-dual vectors of the form $\boldsymbol{z} = \left[\boldsymbol{x}^T, \boldsymbol{\lambda}^T\right]^T$, where $\boldsymbol{x}$ are the design variables and $\boldsymbol{\lambda}$ are the Lagrange multipliers.

4.3.4 Kona Matrices

In addition to abstract vectors, Kona also implements abstract matrices that allow optimization algorithms to use Jacobian-vector products and linear solves. As with vectors, Kona defines a base matrix object called `KonaMatrix` that the core set of matrices all inherit from. These core matrices are the residual Jacobians, `dRdX` and `dRdU`, and the constraint Jacobians, `dCdX` and `dCdU`.

Unlike vectors, however, the base `KonaMatrix` object does not implement any useful methods on its own. Instead, it is used to define an interface that all matrix objects in Kona must adhere to. This interface requires, at minimum, a transpose attribute `.T`, as well as `linearize()` and `product()` member functions. The state-Jacobian of the residual, `dRdU`, additionally implements a `solve()` method as well. These methods are all connected to the appropriate Jacobian-vector products and linear solves defined in the `ISolver` interface, and implemented by the user.

It is important to note here that the Jacobian matrices defined in this module do not possess any data. They are containers for `ISolver` methods, which in turn are recommended to be implemented in a matrix-free fashion. In such matrix-

free implementations, the `linearize()` method does not involve any actual matrix linearizations. Instead, it simply updates references to the design and state vectors about which linear solves and matrix-vector products should be performed. These references are used when calling the appropriate Jacobian-vector product or linear solve method in `ISolver`. However, to efficiently support legacy or matrix-explicit solver implementations, this method also includes a call-back to `ISolver` to trigger the necessary linearizations and factorizations, so that optimization algorithms that perform multiple products or solutions with the same matrix can avoid unnecessary and potentially expensive re-assembly operations.

4.4 Optimization Tools and Algorithms

All data manipulation required by optimization algorithms in Kona is provided by the reverse-communication abstraction layer described above. However, we still need additional tools that build on this abstraction layer in order to be able to perform numerical optimization. Consequently, the algorithm layer of Kona has two purposes: provide basic optimization utilities used by most optimization algorithms, and then build on top of these tools a common interface for performing the optimization.

In this thesis, we will not go into the details of the optimization utilities. These tools are simply Kona-specific versions of well-known algorithms, adapted to operate on `KonaVector` and `KonaMatrix` objects. Instead, we provide a brief list of what the library has currently implemented:

- **Merit Functions** – Three types: the objective function, l_2 merit function, and the augmented Lagrangian merit function.
- **Line-Search Algorithms** – Two methods: a back-tracking line search and a line search satisfying the strong Wolfe conditions.
- **Krylov Solvers** – Four iterative system solvers: Steihaug-Toint Conjugate Gradient (`STCG`) [50], Flexible Generalized Minimum Residual (`FGMRES`) [44], Flexible Generalized Conjugate Residual with Outer Truncation (`GCROT`) [75],

and the Flexible Equality-Constrained Subproblem (`FLECS`) solver [45] which we introduced in Chapter 2.

The optimization interface, which we will describe in detail, is separated into two complementary components: `IHessian`, a `KonaMatrix`-like interface defining a Hessian matrix, and `IAlgorithm`, an algorithm interface that uses the Hessian definition to perform the optimization. This separation grants one optimization algorithm the flexibility to operate with different Hessian approximations and contributes to the symbolic math syntax of the library.

4.4.1 Optimization Algorithms

Optimization algorithms in Kona implement the outer/nonlinear optimization iterations within which `KonaVector`, `KonaMatrix` and `IHessian` objects are used to perform various optimization steps. The `IAlgorithm` interface that defines these objects has very few requirements: they must be initialized using vector factories, and they must implement a `solve()` method that triggers the optimization. Alg. 1 is one such algorithm implemented in Kona using this interface.

One of the important functions of these algorithm objects, besides performing the outer optimization iterations, is to use the vector factories to initialize all the underlying tools required for optimization. At minimum this involves initializing the Hessian approximation, but it could also include setting up line searches, merit functions and Krylov solvers. Performing the initializations using the same vector factories allows the memory manager to correctly determine the memory requirements of the entire optimization process.

Kona currently implements several gradient-based optimization algorithms: unconstrained reduced-space quasi-Newton, unconstrained STCG-based reduced-space Newton-CG, and two equality constrained reduced-space Newton-Krylov (RSNK) algorithms (composite-step [40] and FLECS-based, the latter of which was described in Chapter 2, specifically in Alg. 1).

4.4.2 Hessian Approximations

Kona's `IHessian` interface is modeled after the `KonaMatrix` objects described before as part of the reverse-communication abstraction layer. This Hessian interface requires mandatory `product()` and `solve()` methods that operate on various `KonaVector` objects. Here, the `solve()` method is equivalent to a product with the approximate inverse Hessian. Optionally, some approximations might require a `linearize()` method as well.

Kona currently implements several Hessian objects:

- `LimitedMemoryBFGS`
- `LimitedMemorySR1`
- `ReducedHessian`
- `ReducedKKTMatrix`
- `AugmentedKKTMatrix`
- `LagrangianHessian`
- `TotalConstraintJacobian`

Our reduced-space quasi-Newton algorithm implementation can operate interchangeably with either the L-BFGS or the L-SR1 quasi-Newton approximations, while the remaining five matrices represent reduced-space KKT systems or KKT matrix subblocks used in our RSNK methods. These KKT systems implement the second-order adjoint-based matrix-vector products described in Section 2.3.1.

4.5 Verification Tests

In order to test, debug and verify Kona, we construct three analytical test problems with known optima, intended to exercise the various algorithms implemented in our optimization library. Optimization statements for all three problems are shown in Table 4.2.

Table 4.2. Optimization statements for Kona verification problems.

Unconstrained Spiral problem with state equations

$$\begin{aligned}
\underset{x,\boldsymbol{u}(x)}{\text{minimize}} \quad & \mathcal{J}(x,\boldsymbol{u}(x)) = \frac{1}{2}(x^2+u_1^2+u_2^2) \\
\text{governed by} \quad & \mathcal{R}(x,\boldsymbol{u}) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} - \begin{pmatrix} x^2\cos(\alpha) \\ x^2\sin(\alpha) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\
\text{where} \quad & \theta = \frac{1}{2}(x+\pi) \quad \text{and} \quad \alpha = \frac{1}{2}(x-\pi)
\end{aligned}$$

2-D Rosenbrock function

$$\underset{x,y}{\text{minimize}} \quad \mathcal{J}(x,y) = (1-x)^2 + 100(y-x)^2$$

Equality constrained Sphere problem

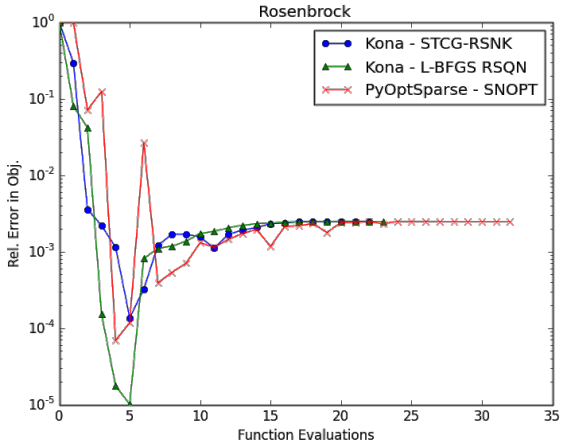
$$\begin{aligned}
\underset{x,y,z}{\text{minimize}} \quad & \mathcal{J}(x,y,z) = x+y+z \\
\text{subject to} \quad & 3-(x^2+y^2+z^2) = 0
\end{aligned}$$

As part of this work, we have also integrated Kona into OpenMDAO 1.x Alpha [76] [3] as an optimization driver. Developed at the NASA Glenn Research Center, OpenMDAO is a platform for development of coupled multidisciplinary models with analytic derivatives. OpenMDAO provides a unified interface for a number of optimizers, including several popular SQP algorithms. Kona's integration links our algorithms to a range of problems implemented as part of OpenMDAO, and allows easy comparison against other SQP methods without having to re-implement a given problem for each optimizer.

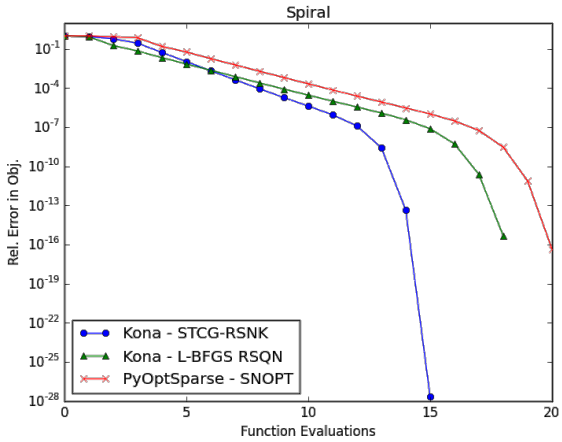
The three verification problems are tested with the following algorithms in Kona:

- **L-BFGS RSQN** – Reduced-space quasi-Newton with limited-memory BFGS updates (unconstrained).
- **STCG-RSNK** – Reduced-space Newton-CG (unconstrained).

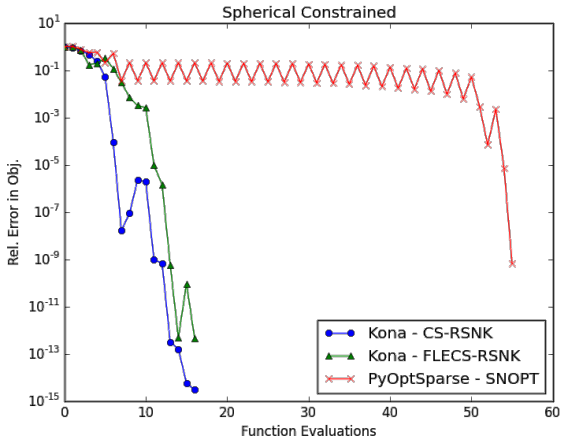
[3] http://openmdao.org/



(a) Rosenbrock function optimization.



(b) Spiral problem optimization.



(c) Sphere problem optimization.

Fig. 4.2. Relative error in the objective value across function evaluations for all verification problems.

- **CS-RSNK** – Composite-step reduced-space Newton-Krylov (constrained).
- **FLECS-RSNK** – FLECS-based reduced-space inexact-Newton-Krylov (constrained).

In addition, the same problems are also solved with SNOPT [18], a robust and popular quasi-Newton SQP library available in OpenMDAO through pyOptSparse [77]. The SNOPT solution provides a verification and validation benchmark for Kona.

Fig. 4.2 shows the objective function evaluations across nonlinear iterations

for all three test problems. The measured quantity is the difference between the numerical objective value and the analytical optimum, normalized by the initial error. All solutions shown are converged to at least 10^{-5} in optimality and feasibility (where applicable).

We can observe that the STCG-RSNK algorithm, which utilizes a second-order adjoint-based matrix-vector product for the Hessian, offers faster convergence on the Spiral problem than either of the quasi-Newton algorithms. This problem features state equations and an objective function that has a nonlinear dependence on the state variables. The ability of the STCG-RSNK algorithm to capture the full curvature information of the problem without any approximations proves advantageous.

Another important observation lies in the convergence of the Sphere problem, named after the spherical equality constraint designed to exercise the globalization methods implemented in Kona. The optimization is deliberated started near the stationary point at $(1, 1, 1)^T$, which is a local maximum that Newton methods would converge to in the absence of globalization. The trust region method used in CS-RSNK and FLECS-RSNK is effective in preventing this, and converging to the correct optimum at $(-1, -1, -1)^T$. SNOPT also recovers the correct optimum; however, the it appears to zig-zag around the local maximum for a number of steps before converging. While the reason for this is not immediately clear, we suspect that it may be related to either the line search method used in SNOPT for step acceptance, or alternatively the need for the quasi-Newton approximation to accumulate sufficient curvature information before it can produce high quality steps.

4.6 Summary

In this chapter, we have described the software design of a Python optimization library, called Kona, that aims to perform parallel optimization of large-scale PDE-governed engineering systems. This is achieved via a reverse-communication abstraction layer that allows optimization algorithms implemented in Kona to perform all necessary optimization tasks without direct access to data. This separation makes the optimization algorithms agnostic to distributed data structures and parallelization schemes implemented by the underlying "PDE solver".

Kona available on GitHub[4] for public use under GNU Lesser General Public License. As part of this work, Kona was also integrated into NASA Glenn Research Center's OpenMDAO framework [76], a platform for development of coupled multi-disciplinary models with analytic derivatives. We have leveraged this integration in verifying and validating Kona's optimization algorithms against SNOPT [18], a popular quasi-Newton SQP library available in OpenMDAO through pyOptSparse [77].

[4]https://github.com/OptimalDesignLab/Kona

CHAPTER 5
AN AERO-STRUCTURAL DESIGN OPTIMIZATION

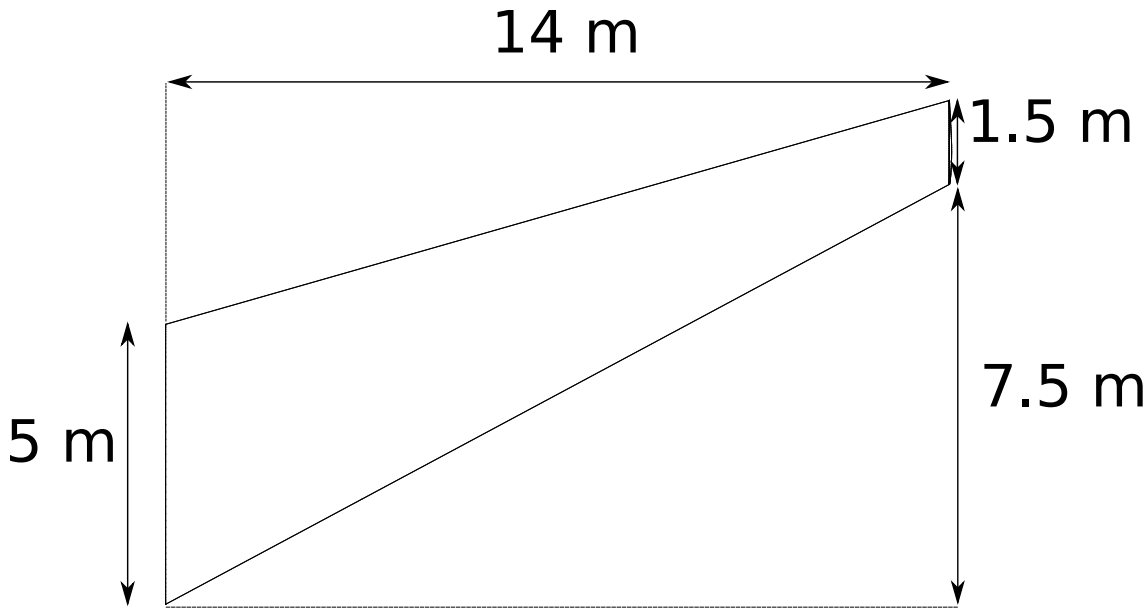
5.1 Introduction

Having introduced the building blocks of the RSNK algorithm, we will now exercise it on a large-scale aero-structural design optimization problem. The test case is based on a tapered swept wing with the RAE 2822 airfoil, shown in Fig. 5.1. The sizing of the problem is derived from the Boeing 717-200HGW, with maximum take-off weight (MTOW) of $54,884$kg, service ceiling of $11,000$m, and cruise speed of 822km/h.

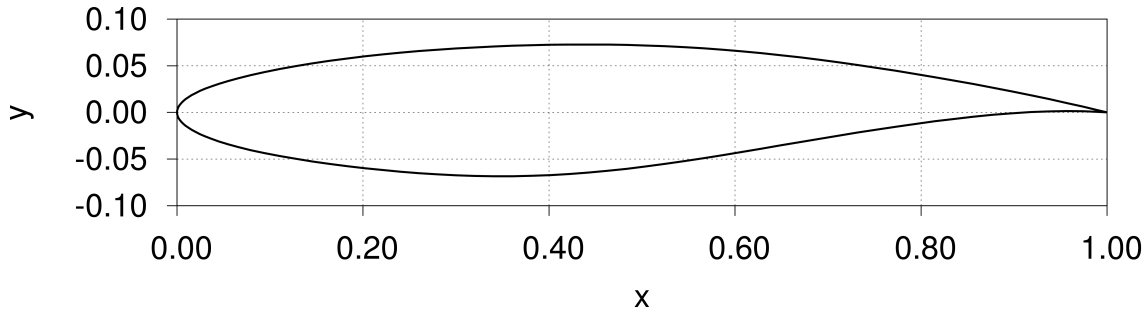
In this high-fidelity test case, the optimization algorithm manipulates twist variables along the wingspan and the size of structural elements in order to minimize a weighted combination of drag and wing mass under lift and stress constraints. As before, we will analyze the efficacy of the IDF preconditioner, compare the RSNK convergence characteristics and computational cost of both the MDF and IDF architectures, and benchmark the RSNK algorithm against SNOPT on the MDF formulation.

Portions of this chapter have been submitted to: A. Dener, J. E. Hicken, G. K. Kenway, and J. R. Martins, "Modular Matrix-free Aero-structural Optimization," in *19th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Atlanta, GA, 2018.

[4]Image source: Prof. Joaquim R. R. A. Martins/University of Michigan, Ann Arbor



(a) Planform view[5].



(b) RAE 2822 airfoil [78].

Fig. 5.1. Geometry of the wing used in the aero-structural optimization.

5.2 Optimization Problem Statement and Setup

The optimization statement for this problem is

$$\begin{aligned}
\underset{\boldsymbol{x}}{\text{minimize}} \quad & \frac{1}{2}\frac{D(\boldsymbol{x},\boldsymbol{u}(\boldsymbol{x}))}{D_0} + \frac{1}{2}\frac{m(\boldsymbol{x})}{m_0}, \\
\text{subject to} \quad & L(\boldsymbol{x},\boldsymbol{u}(\boldsymbol{x})) = \frac{MTOW}{2}, \\
& KS_{ribs,spars} = 1, \\
& KS_{skin,upper} = 1, \\
& KS_{skin,lower} = 1, \\
\text{governed by} \quad & \mathcal{R}_{Aero} = \mathbf{0},\ \mathcal{R}_{Struct} = \mathbf{0}.
\end{aligned} \tag{5.1}$$

The objective function is an equally weighted combination of the scaled drag, $D(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x}))/D_0$, and scaled wing mass, $m(\boldsymbol{x})/m_0$; the drag and mass are scaled by their initial values, D_0 and m_0, respectively. The optimizer manipulates 6 twist variables (at 0%, 20%, 40%, 60%, 80% and 100% along the half-span) and 108 structural sizing variables (thicknesses of the spars, ribs, and skin elements) for a total of 114 design variables.

The lift, $L(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x}))$, is constrained to be half the maximum take-off weight (MTOW) to account for a single-wing solution. The stress constraints are separated into three structural groups: the ribs and spars, the upper skin, and the lower skin. For each group, the scaled Von Mises stresses, σ_v/σ_{yield}, are aggregated using the Kreisselmeier-Steinhauser (KS) function [79]. For a vector of constraints, $\boldsymbol{c} \in \mathbb{R}^K$, this function is defined as

$$KS(\boldsymbol{c}) = \frac{1}{\rho} \ln \left(\sum_{k=1}^{K} \exp(\rho \boldsymbol{c}_k) \right), \tag{5.2}$$

where ρ is a user-defined parameter that must be carefully selected to balance the smoothness of the KS function with the accuracy of the aggregation. In general, higher ρ values increase the accuracy of the KS function in estimating $\boldsymbol{c}_{\max}$, but also produce sharper changes in the gradient. For the present work, we have adopted a recommended value of $\rho = 50$ [80]–[82].

The KS function is considered to be a "conservative" aggregation method, such that

$$\boldsymbol{c}_{\max} \leq KS(\boldsymbol{c}) \leq \boldsymbol{c}_{\max} + \frac{\ln(K)}{\rho}. \tag{5.3}$$

In this application, we set the aggregated KS constraints to be equal to 1, which signifies the boundary of the Von Mises yield criterion where $\sigma_v = \sigma_{yield}$. The overestimation of the Von Mises stresses by the KS aggregation acts as a built-in factor of safety on the structural design, ensuring that the maximum stresses in the structure never exceed the yield stress.

The IDF implementation of this problem introduces an additional $8,400$ coupling variables and coupling constraints: $4,200$ accounting for aerodynamic forces

on the wing surface, and $4,200$ accounting for the structural deformations of the wing surface. The optimization problem sizes for both MDF and IDF are listed in Table 5.1, and the complete list of optimization parameters used in this problem is provided in Table 5.2.

Table 5.1. Size of the aero-structural optimization problem for each MDO architecture.

	MDF	**IDF**
Aerodynamic design variables	6	6
Structural design variables	108	108
Coupling variables	0	8400
Total number of design variables	114	8514
Aerodynamic constraints	1	1
Structural constraints	3	3
Coupling constraints	0	8400
Total number of state-based constraints	4	8404

Table 5.2. Optimization parameters for the aero-structural test case.

Parameter	**Value**
relative optimality tolerance, τ_p	10^{-5}
relative feasibility tolerance, τ_d	10^{-5}
initial trust radius, Δ_0	0.5
maximum trust radius, $\Delta_{\max}$	4.0
initial penalty parameter, μ_0	1.0
initial Krylov tolerance, η_0	0.5
Krylov subspace size	20

5.2.1 CFD Solver

The flow solver used in the multidisciplinary analysis is ADflow, formerly known as SUMad or SUmd [54]. We previously used this tool in the aerodynamic shape optimization problem in Chapter 2. For the aero-structural optimization case, the flow is modeled using the compressible Euler equations and solved on a

structured grid with $96,767$ cells. The Mach number and the angle of attack are fixed at $M = 0.77$ and $\alpha = 2.0°$, respectively.

5.2.2 Structural Solver

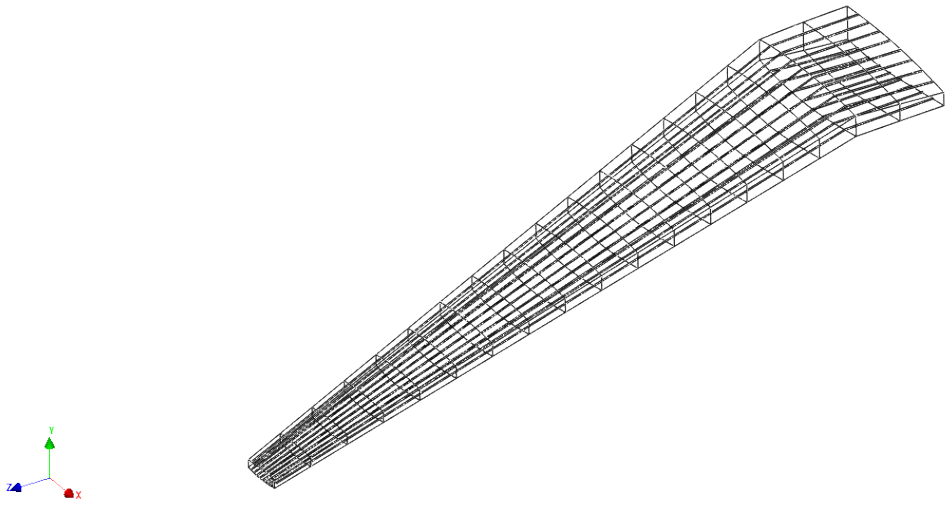


Fig. 5.2. Structural wing box used in the aero-structural problem.

The elastic structural deformations under aerodynamic loads are analyzed using the Toolkit for the Analysis of Composite Structures (TACS) [83]. The wing box, shown in Fig. 5.2, consists of $7,632$ quadrilateral plate elements. The material of the wing box is set as Aluminum 2024, with a density of $\rho = 2780 g/m^3$, Young's modulus of $E = 73.1 GPa$, and yield stress of $\sigma_{yield} = 326 MPa$.

5.2.3 Multidisciplinary Solution for MDF

The coupled aero-structural analysis is performed using nonlinear block-Gauss-Seidel with Aitken acceleration [84], where the aerodynamic forces and structural displacements are transferred using a rigid link between the structural elements and the aerodynamic surface [85]. The mesh movement is performed by the same analytic inverse distance method [55] previously used in the single-discipline aerodynamic shape optimization test case from Chapter 2.

Coupled linear adjoint and forward solutions are implemented as monolithic Krylov solutions. Both the coupled forward and coupled adjoint problems are preconditioned via block-Jacobi, which re-uses the independent discipline preconditioners for the discipline forward and adjoint problems.

5.3 IDF Preconditioner Results

In contrast with the low-fidelity problems considered in Chapter 3, the aerostructural optimization problem features thousands more IDF coupling variables and constraints. Consequently, it is important to examine the efficacy of the IDF preconditioner once again, considering that the size of the nested linear systems are significantly larger.

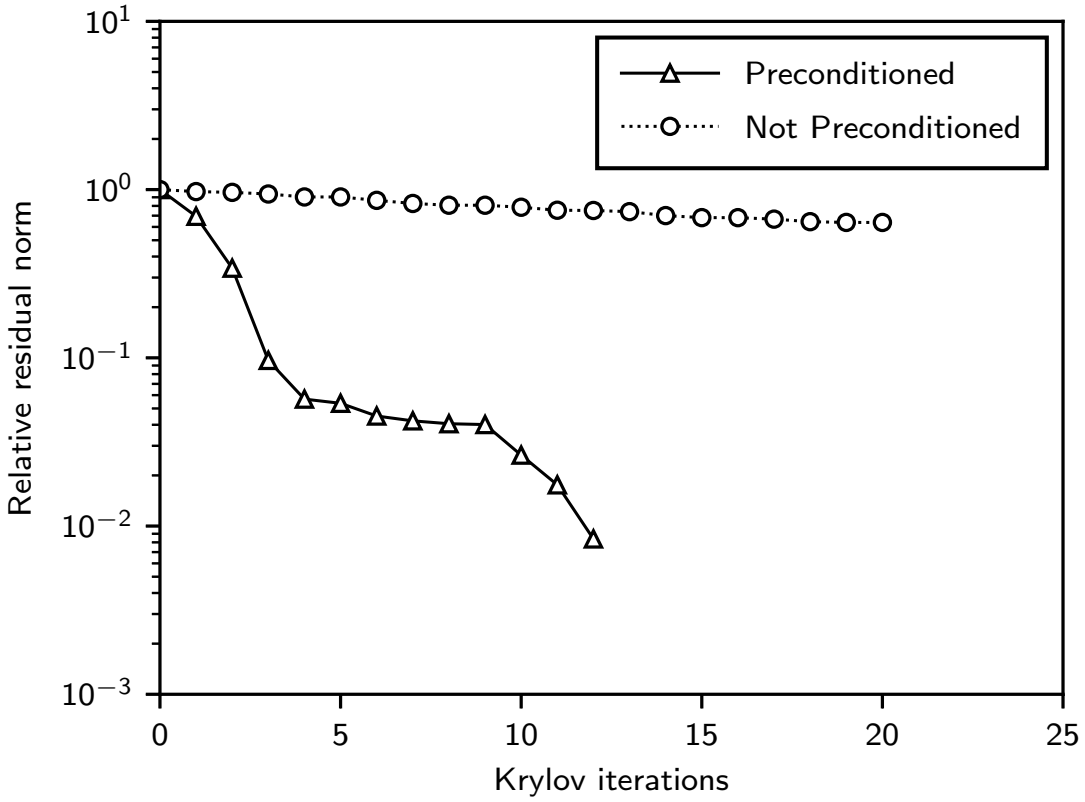


Fig. 5.3. Krylov convergence history for the KKT system at a representative nonlinear iteration.

Fig. 5.3 shows the Krylov convergence history for the solution of the IDF KKT system, with and without the IDF preconditioner. This particular solution is representative of the typical Krylov convergence on this problem. We can see that the IDF preconditioner is once again critically important to the quality of the Newton step. Without the preconditioner, the KKT system cannot be solved to the required tolerance. This result demonstrates that the IDF preconditioner maintains its effectiveness even with thousands of coupling constraints present in the problem.

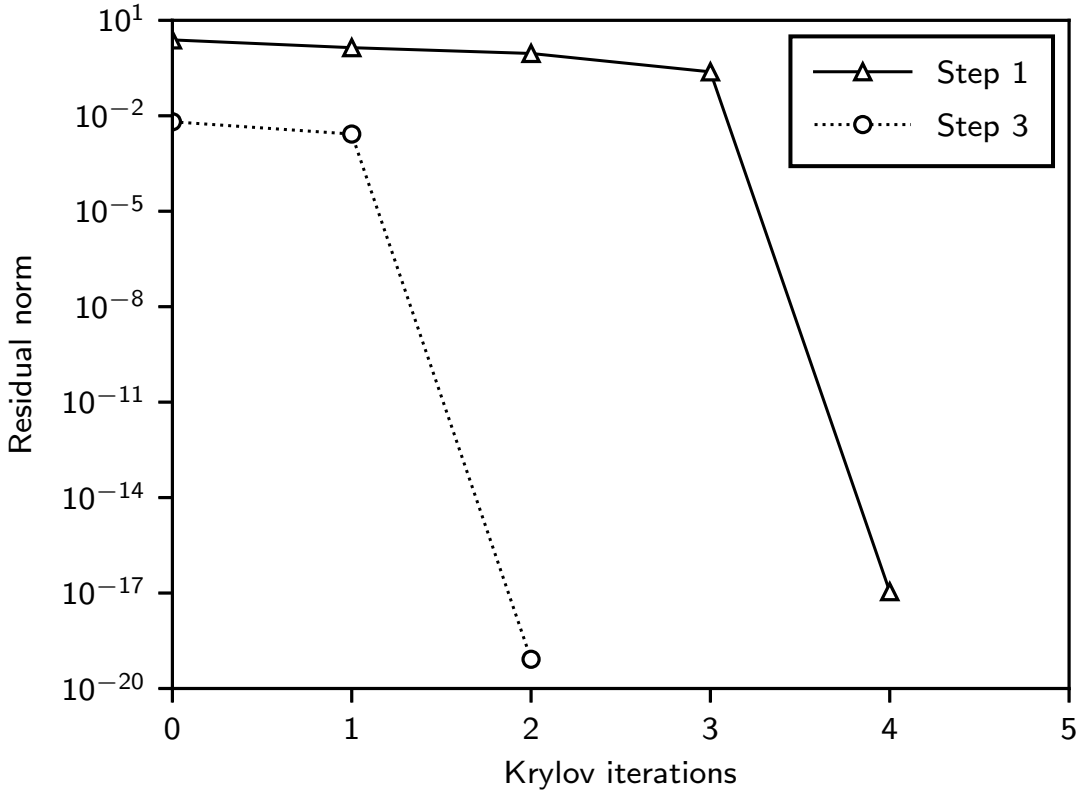


Fig. 5.4. Krylov convergence history of the nested solutions within the IDF preconditioner.

The observation that the IDF preconditioner has maintained its effectiveness despite the considerably larger size of the nested linear problems prompts us to investigate Steps 1 and Step 3 in Alg. 1 more closely. Fig. 5.4 shows a sample convergence history for these nested solutions, representative of the typical nested solutions at all optimization iterations.

It is immediately obvious that the nested solutions converge very rapidly, indicating that the IDF constraint Jacobian is very well conditioned (note that steps 1 and 2 are solved without preconditioning). To understand why, recall that the IDF coupling constraints take the following form for a two-discipline problem:

$$\mathcal{C} = \begin{pmatrix} \mathcal{C}_{\bar{\boldsymbol{u}}} \\ \mathcal{C}_{\bar{\boldsymbol{v}}} \end{pmatrix} = \begin{pmatrix} \mathcal{E}_{\boldsymbol{v}}(\boldsymbol{u}) - \bar{\boldsymbol{u}} \\ \mathcal{E}_{\boldsymbol{u}}(\boldsymbol{v}) - \bar{\boldsymbol{v}} \end{pmatrix} = \mathbf{0}, \tag{5.4}$$

where $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ are coupling variables, and $\boldsymbol{u}$ and $\boldsymbol{v}$ are state variables. The Jacobian of these constraints, with respect to the coupling variables, is

$$\frac{d\mathcal{C}}{d\boldsymbol{w}} = \begin{bmatrix} \frac{d\mathcal{C}_{\bar{\boldsymbol{u}}}}{d\bar{\boldsymbol{v}}} & \frac{d\mathcal{C}_{\bar{\boldsymbol{u}}}}{d\bar{\boldsymbol{v}}} \\ \frac{d\mathcal{C}_{\bar{\boldsymbol{v}}}}{d\bar{\boldsymbol{u}}} & \frac{d\mathcal{C}_{\bar{\boldsymbol{v}}}}{d\bar{\boldsymbol{v}}} \end{bmatrix} = \begin{bmatrix} -\mathsf{I} & \frac{d\mathcal{C}_{\bar{\boldsymbol{u}}}}{d\bar{\boldsymbol{v}}} \\ \frac{d\mathcal{C}_{\bar{\boldsymbol{v}}}}{d\bar{\boldsymbol{u}}} & -\mathsf{I} \end{bmatrix} \tag{5.5}$$

where $\bar{\boldsymbol{w}} = (\bar{\boldsymbol{u}}^T, \bar{\boldsymbol{v}}^T)^T$ is the vector of combined coupling variables. The rapid convergence of the nested solution with this Jacobian suggests that the identity matrix blocks dominate. We also observed the consistent trend that the initial residual norm for the Step 3 solution is approximately two orders magnitude lower than Step 1, and converges more quickly as well. The reason for this is unknown, and requires further study. Ultimately, however, the IDF preconditioner's success on such a problem with 8400 coupling variables and constraints suggests that it can be effectively scaled up to larger problems in the future.

5.4 Optimization Results

The MDF and IDF algorithms recover approximately the same solution for the aero-structural problem, producing the same optimum coefficient of drag and mass to the 4^{th} decimal place. This optimum offers a 74.4% reduction in mass from the initial design, with a 32.2% penalty in the total drag, for an overall 21.1% improvement in the objective.

Initial and optimal lift and spanwise twist distributions are shown in Fig. 5.5 and Fig. 5.6, respectively. These results are plotted only for the RSNK solution of the MDF problem; however, the SNOPT solution of the MDF problem and the

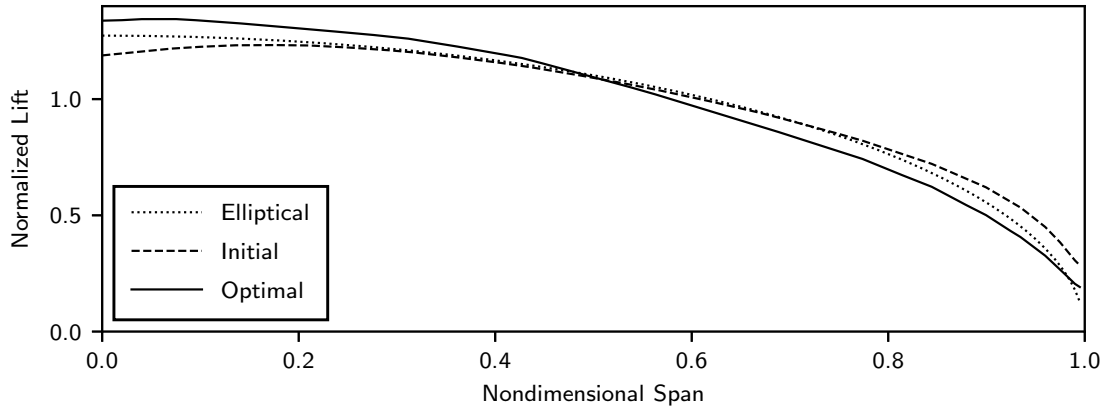


Fig. 5.5. Normalized lift distribution for the aero-structural problem.

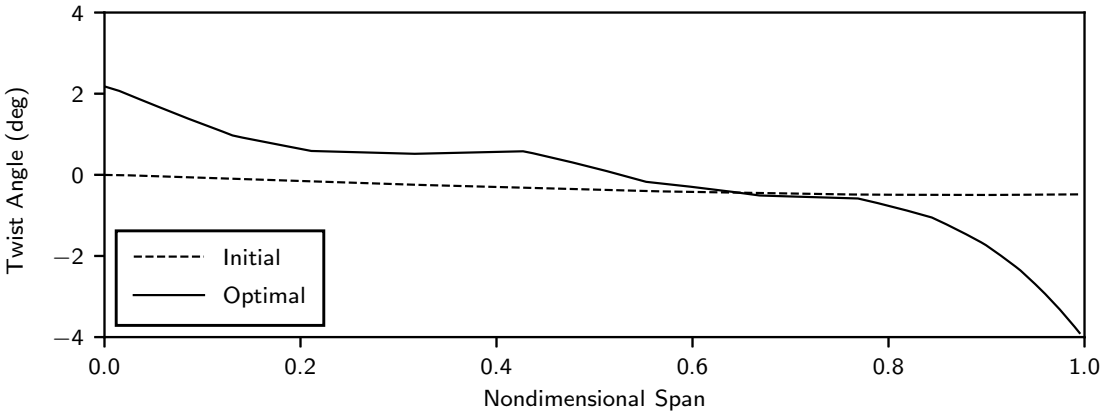
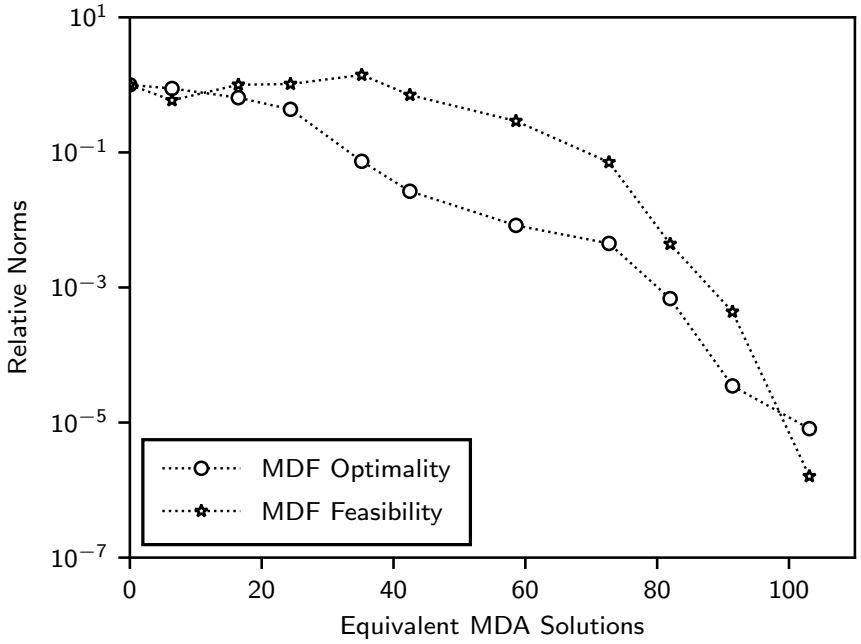


Fig. 5.6. Twist angle along the wing for the aero-structural problem.

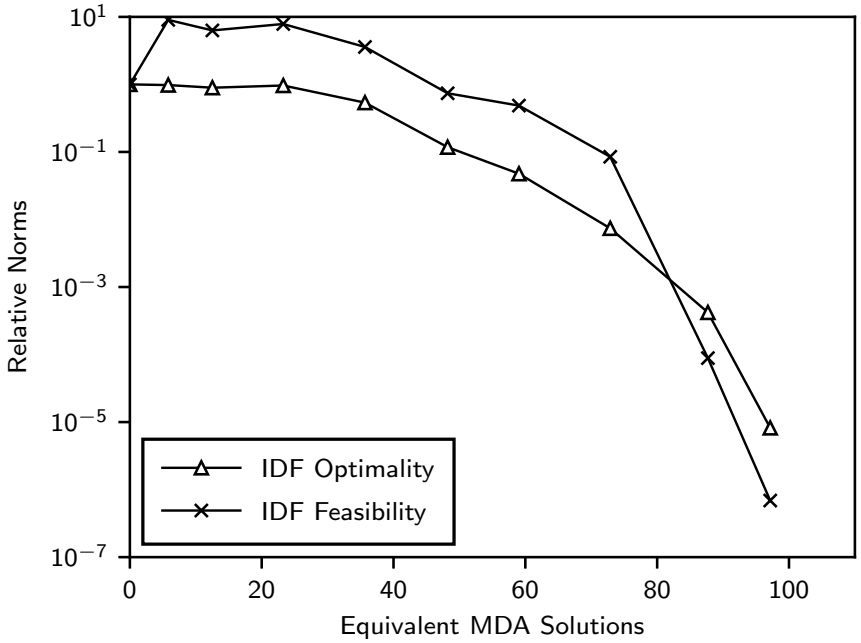
RSNK solution of the IDF problem were both visually indistinguishable from this result.

The initial lift distribution is close to elliptical, which is the pure aerodynamic optimum that minimizes induced drag. The multidisciplinary aero-structural optimum, however, increases the twist angle at the root and decreases it at the wingtip, thereby producing a wash-out in the lift distribution that significantly reduces the stresses in the wing box and permits a lower mass structure.

Fig. 5.7 shows the RSNK convergence histories for the MDF and IDF implementations of the aero-structural problem. The RSNK algorithm exhibits the



(a) MDF formulation.



(b) IDF formulation.

Fig. 5.7. RSNK convergence histories for the aero-structural problem.

expected superlinear asymptotic convergence in both cases. The computational cost for these optimization runs is measured in CPU time, and normalized by the time it takes to perform a single multidisciplinary analysis at the initial design. The results demonstrate that the RSNK algorithm is able to solve the IDF problem at least as efficiently as MDF.

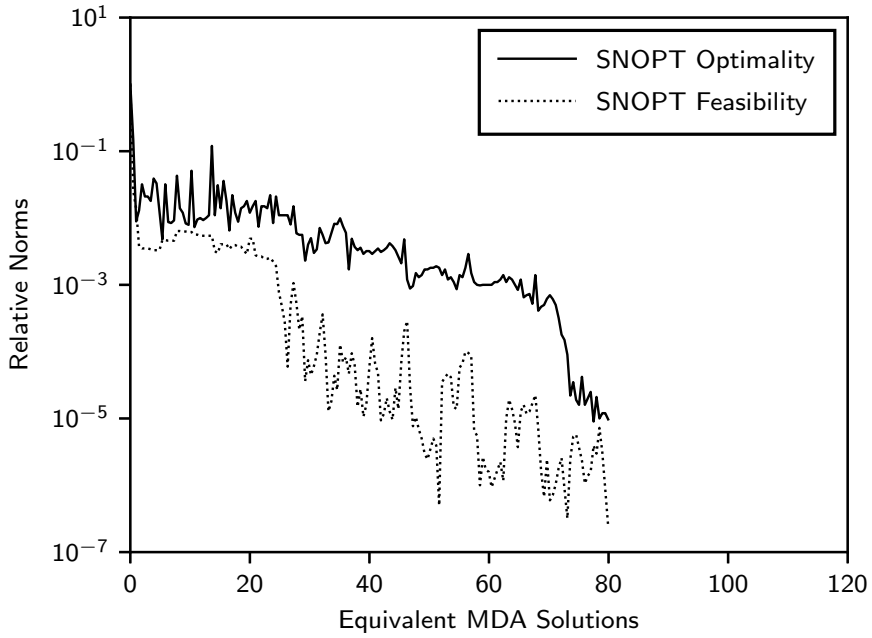


Fig. 5.8. SNOPT convergence history for the MDF aero-structural problem.

As a final benchmark, we solve the MDF problem using SNOPT [18], the quasi-Newton SQO algorithm we previously used in Chapter 2 to solve a single-discipline aerodynamic shape optimization problem. The convergence history for this optimization is shown in Fig. 5.8. SNOPT is able to solve this MDF problem, with 114 design variables, approximately 22% faster than RSNK. This observation is in line with the cost scaling comparison we have performed in Chapter 2. Based on our studies, we expect the RSNK algorithm to become more competitive on large problems, with more design variables and state-based constraints than is present in this aero-structural case.

5.5 Summary

We solved a challenging large-scale aero-structural optimization problem using RSNK implemented with both the MDF and IDF formulations. To the best of our knowledge, the present work is the first application of the IDF architecture to a PDE-governed multidisciplinary problem.

Our results demonstrate that the matrix-free reduced-space inexact-Newton-Krylov approach we have developed in this thesis can successfully solve a large-scale IDF problem at least as efficiently as its MDF counterpart. This is a significant finding, as IDF offers valuable advantages in the modularity of the underlying PDE solvers and ease of implementation in the discipline coupling. Addressing the computational cost challenges associated with the added coupling variables and constraints promotes the IDF architecture as a practical and viable alternative to the commonly used MDF formulation.

It is important to note that the matrix-free IDF preconditioner we have developed in Chapter 3 has been instrumental in achieving this result. We had previously established its efficacy using low-fidelity 2-D test problems featuring on the order of a hundred coupling constraints. This aero-structural problem demonstrates that the IDF preconditioner remains effective with $8,400$ coupling constraints. This increases our confidence in the preconditioner's general applicability to high-fidelity large-scale multidisciplinary optimization problems. To the best of our knowledge, this is the first effective preconditioner developed for the IDF formulation, and represents a significant novel contribution of this thesis work.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

In the last decade, an exponential increase in access to computing power has prompted engineers and researchers alike to develop simulation tools for increasingly more complex multidisciplinary problems. With increasing complexity, however, comes increasing difficulty in performing engineering design solely through intuition and manual iteration. Multidisciplinary design optimization (MDO) is a powerful methodology that can help inform the design of such systems. The motivation of this thesis was to help proliferate the use of MDO in engineering applications that involve the computationally expensive solution of coupled partial differential equations (PDEs). To that end, we have developed a reduced-space inexact-Newton-Krylov algorithm for the efficient solution of multidisciplinary problems formulated with the individual discipline feasible (IDF) architecture.

IDF provides modularity in the underlying PDE solvers, such that the state equations (e.g. discretized PDEs) for each discipline can be solved independently of each other at each optimization iteration. This enables the use of existing stand-alone PDE solvers and their specialized solution methods. However, off-loading the multidisciplinary coupling to the optimization problem introduces large numbers of additional design variables and nonlinear state-based constraints. This poses a challenge to conventional gradient-based sequential quadratic optimization algorithms which typically require the constraint Jacobian to be provided explicitly for factorization purposes.

The RSNK algorithm presented in this thesis overcomes these challenges by solving the Newton step in matrix-free fashion, using a Krylov solver. In this approach, the matrix-vector product with the Karush-Kuhn-Tucker (KKT) matrix is computed using second-order adjoints. This comes at the cost of two linear solves with the PDE Jacobian, independent of the number of design variables and state-based constraints. The Krylov solution is then preconditioned via a novel matrix-free preconditioner tailored for the IDF problem. The resulting Newton step is globalized

in a trust region framework, using a simple filter for step acceptance.

We have demonstrated the RSNK algorithm on a wide range of problems in both MDF and IDF formulations, and studied the effectiveness of the novel matrix-free IDF preconditioner. Our results establish that RSNK exhibits the superlinear asymptotic convergence expected of Newton-type methods, offers favorable cost scaling with the size of the design space, and does not require the expensive explicit computation of the constraint Jacobian. Most importantly, we have shown that the IDF preconditioner is critically important to the successful solution of the Newton step; without it, the rate of convergence of the Krylov solution is too slow to be practically viable. In fact, our numerical experiments with restrictions on the number of Krylov iterations failed to converge to an optimum without the IDF preconditioner.

Finally, a large-scale aero-structural optimization application revealed that the RSNK algorithm, with the help of the IDF preconditioner, can solve IDF problems at least as efficiently as their MDF counterparts. Consequently, our approach enables engineers to take advantage of the modularity and ease of implementation benefits offered by the IDF architecture, but avoid suffering a penalty in the computational cost of the optimization relative to the commonly used MDF architecture.

6.1 Future Research Areas

In closing, we present a number of recommendations for future research. These are based on the experience and results obtained throughout the course of this thesis.

6.1.1 Theory-Based Update of the FLECS Penalty Parameter

The iterative solution of the KKT system in the RSNK algorithm uses a specialized Krylov solver called FLECS that extends FGMRES to handle nonconvexity. This is done through a quadratic penalty function in the primal subproblem, which involves a penalty parameter, μ that needs to be controlled from one nonlinear iteration to the next.

The currently implemented approach to determining μ values in RSNK is heuristic, and it is based on our intuition and experience with numerical experiments. In the interest of robustness and ease of use, however, we recommend developing a

mathematical foundation for an update that promotes optimal convergence rates in the nonlinear problem.

The quadratic penalty term in FLECS, $\mu(\mathsf{A}\boldsymbol{p} + \mathcal{C})^T(\mathsf{A}\boldsymbol{p} + \mathcal{C})/2$, is related to augmented Lagrangian methods [86]. In augmented Lagrangian methods, the constrained optimization problem is converted into a sequence of unconstrained problems through a penalty function. This penalty function, like the one in FLECS, uses a penalty parameter that is updated in outer iterations, and fixed in inner iterations, which is analogous to the updates we perform in our nonlinear RSNK iterations while keeping μ fixed through linear Krylov iterations. Literature on augmented Lagrangian methods do offer some theoretical guidance on the update of these penalty parameters that could potentially be adapted for use in RSNK.

6.1.2 Extending RSNK to Inequality Constraints

In this thesis, we have developed the RSNK algorithm exclusively for equality constraints and have not attempted to address inequality constraints. Consequently, our test problems throughout were carefully chosen or adapted to include only equality constraints. This was a significant challenge throughout this work, particularly in the case of single-discipline aerodynamic shape optimization. The absence of support for inequality constraints prevented us from imposing thickness constraints on the wing geometry, and necessitated the use of a regularization term in the objective function that would discourage very thin trailing edges.

On the other hand, existing methods in literature for solving inequality constraints in sequential quadratic optimization all involve converting the inequality constraints into equality constraints, or eliminating them altogether via a penalty function that combines the objective and constraints. The RSNK algorithm we have developed and presented in this thesis can accommodate all of these approaches.

Linear or bound constraints can be accounted for via barrier functions [24], [87]. For nonlinear or state-based inequality constraints, the active set method [24] offers the most straightforward approach in terms of the software implementation, with the caveat that it requires element-wise data access into dual vectors and constraint Jacobians. In exploratory work that is not included in this thesis, we

have also had some success experimenting with interior-point methods [88].

6.1.3 Matrix-Free Constraint Jacobians in Quasi-Newton SQO Methods

Throughout this thesis, one of the key reasons why we have sought to avoid conventional quasi-Newton SQO methods is that they typically require the computationally expensive assembly of explicit constraint Jacobians. This has been a major motivation behind developing our matrix-free approach, and computing matrix-vector products with the KKT matrix using second-order adjoints.

An analogous second-order adjoint-based matrix-vector product exists for the stand-alone constraint Jacobian instead of the complete KKT matrix. One notable difference with this product is that the right-hand-side of the second-order adjoint system does not involve any terms computed with a forward-difference approximation. This raises the natural question of whether such a product can be used within a quasi-Newton SQO algorithm in order to avoid the finite difference step size dilemma entirely.

For a potential answer, we turn to the SQO review in Chapter 2 where we discussed the different ways in which traditional SQO algorithms solve constrained problems. One of the cited methods is the range-space method [23], [24], where the KKT system in (2.7) is rearranged into

$$(\mathsf{A}_k \mathsf{W}_k^{-1} \mathsf{A}_k^T) \boldsymbol{\lambda}_{k+1} = \mathsf{A}_k \mathsf{W}_k^{-1} \frac{d\mathcal{J}_k}{d\boldsymbol{x}} - \mathcal{C}_k, \tag{6.1a}$$

$$\mathsf{W}_k \boldsymbol{p}_k = - \frac{d\mathcal{J}_k}{d\boldsymbol{x}} + \mathsf{A}_k^T \boldsymbol{\lambda}_{k+1}. \tag{6.1b}$$

In the conventional approach, the dual system in (6.1a) is solved with a direct method (i.e. factorization). However, it is possible to solve it iteratively using a Krylov solver provided that W_k^{-1} is cheaply available. This approach avoids the requirement to construct the left-hand-side matrix explicitly.

The matrix-free product with the left-hand-side matrix of (6.1a) is straightforward to compute. The inverse of the Hessian, W_k^{-1}, is cheaply approximated using a quasi-Newton method, while products with the constraint Jacobian, A_k, can be provided via a second-order adjoint-based formulation. Once the dual system is solved

for $\boldsymbol{\lambda}_{k+1}$, it is only a matter of another quasi-Newton approximation application to compute the primal step, $\boldsymbol{p}_k$.

It is an open ended question whether the Krylov solution of the dual system will require preconditioning, and how to construct an effective preconditioner. The operation ($\mathsf{A}\mathsf{A}^\mathsf{T}$) squares the condition number of A; therefore, in the presence of large numbers of general nonlinear constraints, we could expect slow convergence rates. However, our numerical experiments with the IDF formulation indicate that the IDF constraint Jacobian is well conditioned and Krylov solutions of its associated systems converge quickly without the need for preconditioning. Consequently, this matrix-free quasi-Newton SQP approach could be a viable alternative to the RSNK algorithm for IDF problems with only a few additional state-based constraints, with the added benefit of not requiring any preconditioning.

REFERENCES

[1] A. Dener and J. E. Hicken, "Matrix-free algorithm for the optimization of multidisciplinary systems," *Structural and Multidisciplinary Optimization*, vol. 56, no. 6, pp. 1429–1446, Jun. 2017.

[2] N. P. Tedford and J. R. R. A. Martins, "Benchmarking multidisciplinary design optimization algorithms," *Optimization and Eng.*, vol. 11, no. 1, pp. 159–183, Mar. 2009.

[3] J. R. R. A. Martins and A. B. Lambe, "Multidisciplinary design optimization: A survey of architectures," *AIAA J.*, vol. 51, no. 9, pp. 2049–2075, Sep. 2013.

[4] R. T. Haftka, "Simultaneous analysis and design," *AIAA J.*, vol. 23, no. 7, pp. 1099–1103, Jul. 1985.

[5] S. Ta′asan, G. Kuruvila, and M. Salas, "Aerodynamic design and optimization in one shot," presented at the 30th Aerospace Sciences Meeting and Exhibit, Reno, NV, USA, 1992.

[6] J. Herskovits, P. Mappa, E. Goulart, and C. M. Mota Soares, "Mathematical programming models and algorithms for engineering design optimization," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 194, no. 30–33, pp. 3244–3268, Aug. 2005.

[7] G. Biros and O. Ghattas, "Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization — Part I: The Krylov-Schur solver," *SIAM J. on Scientific Comput.*, vol. 27, no. 2, pp. 687–713, Jan. 2005.

[8] G. Biros and O. Ghattas, "Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization — Part II: The Lagrange-Newton solver and its application to optimal control of steady viscous flows," *SIAM J. on Scientific Comput.*, vol. 27, no. 2, pp. 714–739, Jan. 2005.

[9] E. Özkaya and N. R. Gauger, "Single-step one-shot aerodynamic shape optimization," in *Optimal Control of Coupled Systems of Partial Differential Equations*. Basel, Switzerland: Birkhäuser, 2009, pp. 191–204.

[10] G. J. Kennedy and J. R. R. A. Martins, "Parallel solution methods for aerostructural analysis and design optimization," presented at the 13th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conf., Fort Worth, TX, USA, 2010.

[11] K. Maute, M. Nikbay, and C. Farhat, "Coupled analytical sensitivity analysis and optimization of three-dimensional nonlinear aeroelastic systems," *AIAA J.*, vol. 39, no. 11, pp. 2051–2061, Nov. 2001.

[12] K. Maute, M. Nikbay, and C. Farhat, "Sensitivity analysis and design optimization of three-dimensional non-linear aeroelastic systems by the adjoint method," *Int. J. for Numerical Methods in Eng.*, vol. 56, no. 6, pp. 911–933, Dec. 2003.

[13] J. R. R. A. Martins, J. J. Alonso, and J. J. Reuther, "High-fidelity aerostructural design optimization of a supersonic business jet," *J. of Aircraft*, vol. 41, no. 3, pp. 523–530, May 2004.

[14] G. K. W. Kenway, G. J. Kennedy, and J. R. R. A. Martins, "Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and adjoint derivative computations," *AIAA J.*, vol. 52, no. 5, pp. 935–951, May 2014.

[15] R. T. Haftka, J. Sobieszczanski-Sobieski, and S. L. Padula, "On options for interdisciplinary analysis and design optimization," *Structural Optimization*, vol. 4, no. 2, pp. 65–74, Jun. 1992.

[16] E. J. Cramer, J. E. Dennis, Jr., P. D. Frank, R. M. Lewis, and G. R. Shubin, "Problem formulation for multidisciplinary optimization," *SIAM J. on Optimization*, vol. 4, no. 4, pp. 754–776, Nov. 1994.

[17] K. Schittkowski, "NLPQL: A Fortran subroutine solving constrained nonlinear programming problems," *Ann. of Operations Res.*, vol. 5, no. 1–4, pp. 485–500, May 1986.

[18] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Rev.*, vol. 47, no. 1, pp. 99–131, Jan. 2005.

[19] S. G. Nash and J. Nocedal, "A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization," *SIAM J. on Optimization*, vol. 1, no. 3, pp. 358–372, Aug. 1991.

[20] A. Dener, G. K. W. Kenway, J. E. Hicken, and J. R. R. A. Martins, "Comparison of inexact- and quasi-Newton algorithms for aerodynamic shape optimization," presented at the 53rd AIAA Aerospace Sciences Meeting, Kissimmee, FL, USA, 2015.

[21] J. R. Bunch and L. Kaufman, "Some stable methods for calculating inertia and solving symmetric linear systems," *Math. of Comput.*, vol. 31, no. 137, pp. 163–163, Jan. 1977.

[22] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear," *ACM Trans. on Math. Software*, vol. 9, no. 3, pp. 302–325, Sep. 1983.

[23] P. E. Gill, N. I. M. Gould, W. Murray, M. A. Saunders, and M. H. Wright, "A weighted Gram-Schmidt method for convex quadratic programming," *Math. Programming*, vol. 30, no. 2, pp. 176–195, Oct. 1984.

[24] S. J. Wright and J. Nocedal, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[25] P. E. Gill and W. Murray, "Numerically stable methods for quadratic programming," *Math. Programming*, vol. 14, no. 1, pp. 349–372, Dec. 1978.

[26] W. C. Davidon, "Variable metric method for minimization," *SIAM J. on Optimization*, vol. 1, no. 1, pp. 1–17, Feb. 1991.

[27] R. Fletcher, *Practical Methods of Optimization*. Hoboken, NJ, USA: John Wiley & Sons, Ltd, May 2000.

[28] R. H. Byrd, H. F. Khalfan, and R. B. Schnabel, "Analysis of a symmetric rank-one trust region method," *SIAM J. on Optimization*, vol. 6, no. 4, pp. 1025–1039, Nov. 1996.

[29] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms 1. general considerations," *IMA J. of Appl. Math.*, vol. 6, no. 1, pp. 76–90, Mar. 1970.

[30] R. Fletcher, "A new approach to variable metric algorithms," *The Comput. J.*, vol. 13, no. 3, pp. 317–322, Mar. 1970.

[31] D. Goldfarb, "A family of variable-metric methods derived by variational means," *Math. of Comput.*, vol. 24, no. 109, pp. 23–23, Jan. 1970.

[32] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization," *Math. of Comput.*, vol. 24, no. 111, pp. 647–647, Sep. 1970.

[33] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Math. of Comput.*, vol. 35, no. 151, pp. 773–773, Sep. 1980.

[34] J.-L. Lions, *Contrôle optimal de systèmes gouvernés par des équations aux dérivées partielles* (Etudes Mathématiques). Paris, France: Dunod/Gauthier-Villars, 1968.

[35] A. Jameson, "Aerodynamic design via control theory," *J. of Scientific Comput.*, vol. 3, no. 3, pp. 233–260, Sep. 1988.

[36] S. Arreckx, A. B. Lambe, J. R. R. A. Martins, and D. Orban, "A matrix-free augmented Lagrangian algorithm with application to large-scale structural design optimization," *Optimization and Eng.*, vol. 17, no. 2, pp. 359–384, Oct. 2015.

[37] R. H. Byrd, F. E. Curtis, and J. Nocedal, "An inexact SQP method for equality constrained optimization," *SIAM J. on Optimization*, vol. 19, no. 1, pp. 351–369, Jan. 2008.

[38] M. Heinkenschloss and D. Ridzal, "An inexact trust-region SQP method with applications to PDE-constrained optimization," in *Numerical Mathematics and Advanced Applications*. New York, NY, USA: Springer, 2008, pp. 613–620.

[39] R. H. Byrd, F. E. Curtis, and J. Nocedal, "An inexact Newton method for nonconvex equality constrained optimization," *Math. Programming*, vol. 122, no. 2, pp. 273–299, Oct. 2008.

[40] M. Heinkenschloss and D. Ridzal, "A matrix-free trust-region SQP method for equality constrained optimization," *SIAM J. on Optimization*, vol. 24, no. 3, pp. 1507–1541, Jan. 2014.

[41] D. Knoll and D. Keyes, "Jacobian-free Newton–Krylov methods: A survey of approaches and applications," *J. of Comput. Physics*, vol. 193, no. 2, pp. 357–397, Jan. 2004.

[42] Z. Wang, I. M. Navon, F. X. Le Dimet, and X. Zou, "The second order adjoint analysis: Theory and applications," *Meteorology and Atmospheric Physics*, vol. 50, no. 1–3, pp. 3–20, Mar. 1992.

[43] E. Nielsen, R. Walters, W. Anderson, and D. Keyes, "Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code," presented at the 12th Computational Fluid Dynamics Conf., San Diego, CA, USA, 1995.

[44] Y. Saad, "A flexible inner-outer preconditioned GMRES algorithm," *SIAM J. on Scientific Comput.*, vol. 14, no. 2, pp. 461–469, Mar. 1993.

[45] J. E. Hicken and A. Dener, "A flexible iterative solver for nonconvex, equality-constrained quadratic subproblems," *SIAM J. on Scientific Comput.*, vol. 37, no. 4, pp. A1801–A1824, Jan. 2015.

[46] J. J. Moré and D. C. Sorensen, "Computing a trust region step," *SIAM J. on Scientific and Statistical Comput.*, vol. 4, no. 3, pp. 553–572, Sep. 1983.

[47] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust Region Methods*. Philadelphia, PA, USA: Soc. for Ind. & Appl. Math., Jan. 2000.

[48] R. Fletcher and S. Leyffer, "Nonlinear programming without a penalty function," *Math. Programming*, vol. 91, no. 2, pp. 239–269, Jan. 2002.

[49] S. C. Eisenstat and H. F. Walker, "Choosing the forcing terms in an inexact Newton method," *SIAM J. on Scientific Comput.*, vol. 17, no. 1, pp. 16–32, Jan. 1996.

[50] T. Steihaug, "The conjugate gradient method and trust regions in large scale optimization," *SIAM J. on Numerical Anal.*, vol. 20, no. 3, pp. 626–637, Jun. 1983.

[51] J. Vassberg, M. Dehaan, M. Rivers, and R. Wahls, "Development of a common research model for applied CFD validation studies," presented at the 26th AIAA Applied Aerodynamics Conf., Honolulu, HI, USA, 2008.

[52] Z. Lyu, G. K. W. Kenway, and J. R. R. A. Martins, "Aerodynamic shape optimization investigations of the common research model wing benchmark," *AIAA J.*, vol. 53, no. 4, pp. 968–985, Apr. 2015.

[53] G. K. W. Kenway, G. J. Kennedy, and J. R. R. A. Martins, "A CAD-free approach to high-fidelity aerostructural optimization," presented at the 13th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conf., Fort Worth, TX, USA, 2010.

[54] E. van der Weide, G. Kalitzin, J. Schluter, and J. J. Alonso, "Unsteady turbomachinery computations using massively parallel platforms," presented at the 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, USA, 2006.

[55] E. Luke, E. Collins, and E. Blades, "A fast mesh deformation method using explicit interpolation," *J. of Comput. Physics*, vol. 231, no. 2, pp. 586–601, Jan. 2012.

[56] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta Numerica*, vol. 14, pp. 1–137, May 2005.

[57] M. A. Heroux, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring *et al.*, "An overview of the Trilinos project," *ACM Trans. on Math. Software*, vol. 31, no. 3, pp. 397–423, Sep. 2005.

[58] C. L. Bloebaum, "Coupling strength-based system reduction for complex engineering design," *Structural Optimization*, vol. 10, no. 2, pp. 113–121, Oct. 1995.

[59] S. Kodiyalam and J. Sobieszczanski-Sobieski, "Multidisciplinary design optimization — Some formal methods, framework requirements, and application to vehicle design," *Int. J. of Vehicle Des.*, vol. 25, no. 1/2, pp. 3–22, 2001.

[60] H. O. Kreiss and G. Scherer, "Finite element and finite difference methods for hyperbolic partial differential equations," in *Mathematical Aspects of Finite Elements in Partial Differential Equations*. New York, NY, USA: Academic Press, 1974, pp. 195–212.

[61] B. Strand, "Summation by parts for finite difference approximations for d/dx," *J. of Comput. Physics*, vol. 110, no. 1, pp. 47–67, Jan. 1994.

[62] D. Funaro and D. Gottlieb, "A new method of imposing boundary conditions in pseudospectral approximations of hyperbolic equations," *Math. of Comput.*, vol. 51, no. 184, pp. 599–599, Oct. 1988.

[63] M. H. Carpenter, D. Gottlieb, and S. Abarbanel, "Time-stable boundary conditions for finite-difference schemes solving hyperbolic systems: Methodology and application to high-order compact schemes," *J. of Comput. Physics*, vol. 111, no. 2, pp. 220–236, Apr. 1994.

[64] K. Mattsson, M. Svärd, and J. Nordström, "Stable and accurate artificial dissipation," *J. of Scientific Comput.*, vol. 21, no. 1, pp. 57–79, Aug. 2004.

[65] M. Turner, *The Direct Stiffness Method of Structural Analysis*. Seattle, WA, USA: Boeing Airplane Company, 1959.

[66] C. Felippa, "A historical outline of matrix structural analysis: A play in three acts," *Comput. & Structures*, vol. 79, no. 14, pp. 1313–1324, Jun. 2001.

[67] J. E. Hicken and D. W. Zingg, "Summation-by-parts operators and high-order quadrature," *J. of Comput. and Appl. Math.*, vol. 237, no. 1, pp. 111–125, Jan. 2013.

[68] A. Dener, P. Meng, J. E. Hicken, G. J. Kennedy, J. Hwang, and J. S. Gray, "Kona: A parallel optimization library for engineering-design problems," presented at the 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conf., San Diego, CA, USA, 2016.

[69] S. F. Ashby and M. K. Seager, "A proposed standard for iterative linear solvers," Lawrence Livermore Nat. Lab., Livermore, CA, USA, Tech. Rep. UCRL-102860, 1990.

[70] P. Peterson, "F2PY: A tool for connecting Fortran and Python programs," *Int. J. of Comput. Sci. and Eng.*, vol. 4, no. 4, p. 296, Nov. 2009.

[71] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Comput. in Sci. & Eng.*, vol. 13, no. 2, pp. 31–39, Mar. 2011.

[72] D. M. Beazley and P. S. Lomdahl, "Feeding a large-scale physics application to Python," presented at the 6th Int. Python Conf., San Jose, CA, USA, 1997.

[73] D. Abrahams and R. W. Grosse-Kunstleve, "Building hybrid systems with Boost. Python," *C++ Users J.*, vol. 21, no. 7, pp. 29–36, May. 2003.

[74] R. A. Bartlett, "Thyra linear operators and vectors," Sandia Nat. Labs., Albuquerque, NM, USA, Tech. Rep. SAND2007-5984, Aug. 2010.

[75] J. E. Hicken and D. W. Zingg, "A simplified and flexible variant of GCROT for solving nonsymmetric linear systems," *SIAM J. on Scientific Comput.*, vol. 32, no. 3, pp. 1672–1694, Jan. 2010.

[76] J. S. Gray, T. A. Hearn, K. T. Moore, J. Hwang, J. R. R. A. Martins, and A. Ning, "Automatic evaluation of multidisciplinary derivatives using a graph-based problem formulation in OpenMDAO," presented at the 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conf., Atlanta, GA, USA, 2014.

[77] J. Hwang, G. K. W. Kenway, and J. R. R. A. Martins, "Geometry and structural modeling for high-fidelity aircraft conceptual design optimization," presented at the 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conf., Atlanta, GA, USA, 2014.

[78] E. Iuliano and D. Quagliarella, "Proper orthogonal decomposition, surrogate modelling and evolutionary optimization in aerodynamic design," *Comput. & Fluids*, vol. 84, pp. 327–350, Sep. 2013.

[79] G. Kreisselmeier and R. Steinhauser, "Systematic control design by optimizing a vector performance index," in *Computer Aided Design of Control Systems*. Oxford, United Kingdom: Pergamon, 1980, pp. 113–117.

[80] C. Raspanti, J. Bandoni, and L. Biegler, "New strategies for flexibility analysis and design under uncertainty," *Comput. & Chemical Eng.*, vol. 24, no. 9–10, pp. 2193–2209, Oct. 2000.

[81] M. A. Akgün, R. T. Haftka, K. C. Wu, J. L. Walsh, and J. H. Garcelon, "Efficient structural optimization for multiple load cases using adjoint sensitivities," *AIAA J.*, vol. 39, no. 3, pp. 511–516, Mar. 2001.

[82] N. M. K. Poon and J. R. R. A. Martins, "An adaptive approach to constraint aggregation using adjoint sensitivity analysis," *Structural and Multidisciplinary Optimization*, vol. 34, no. 1, pp. 61–73, Dec. 2006.

[83] G. J. Kennedy and J. R. Martins, "A parallel finite-element framework for large-scale gradient-based design optimization of high-performance structures," *Finite Elements in Anal. and Des.*, vol. 87, pp. 56–73, Sep. 2014.

[84] B. M. Irons and R. C. Tuck, "A version of the Aitken accelerator for computer iteration," *Int. J. for Numerical Methods in Eng.*, vol. 1, no. 3, pp. 275–277, Jul. 1969.

[85] S. A. Brown, "Displacement extrapolations for CFD+CSM aeroelastic analysis," presented at the 38th Structures, Structural Dynamics, and Materials Conf., Kissimmee, FL, USA, 1997.

[86] M. R. Hestenes, "Multiplier and gradient methods," *J. of Optimization Theory and Appl.*, vol. 4, no. 5, pp. 303–320, Nov. 1969.

[87] R. J. Vanderbei, *Linear Programming* (Int. Series in Operations Res. & Management Sci.). New York, NY, USA: Springer, 2014.

[88] F. A. Potra and S. J. Wright, "Interior-point methods," *J. of Comput. and Appl. Math.*, vol. 124, no. 1–2, pp. 281–302, Dec. 2000.

APPENDIX A
SUPPLEMENTARY RESULTS FOR AERODYNAMIC SHAPE OPTIMIZATION

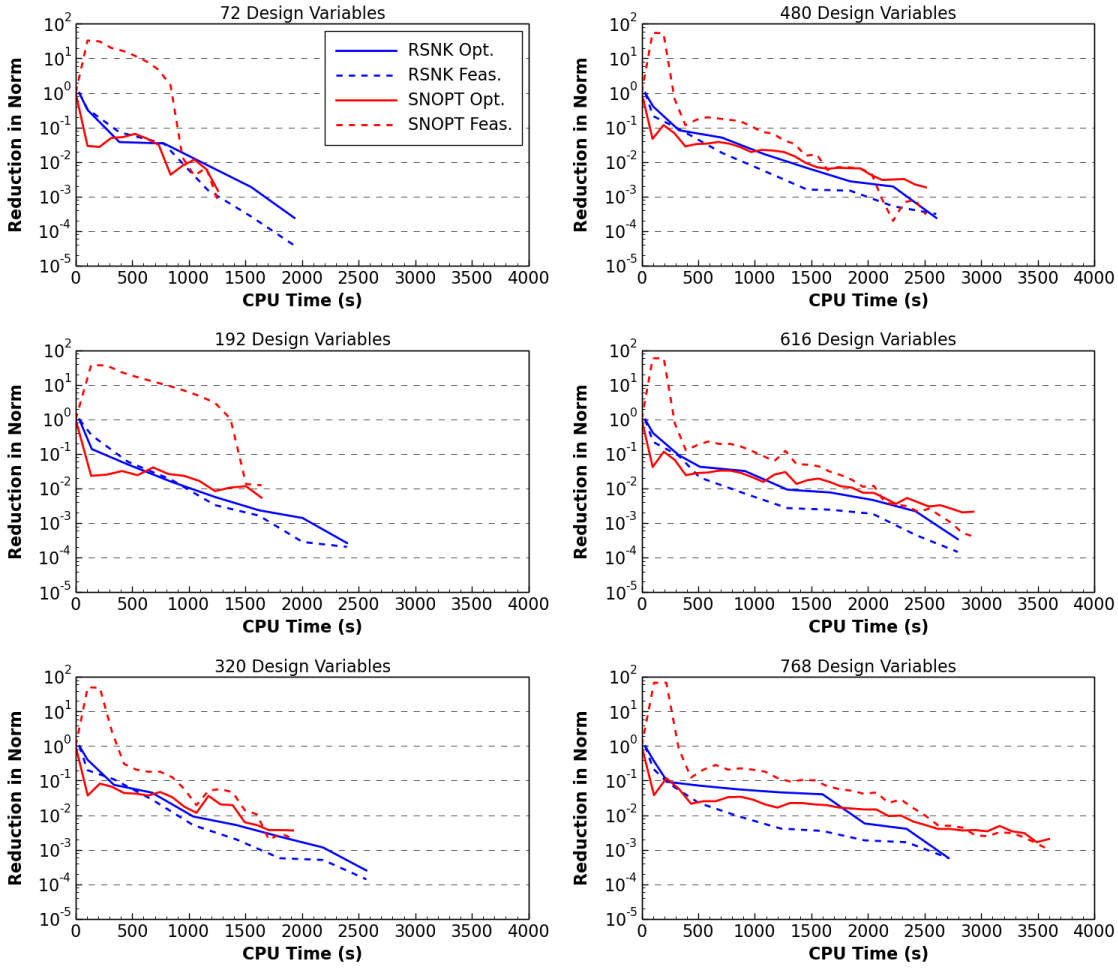


Fig. A.1. Optimality and feasibility convergence history of RSNK and SNOPT at dierent design space sizes.