EXTENSION OF THE ADJOIN Approach TO A LAMINAR NAVIER–STOKES Solver

by

Cody Paige

A thesis submitted in conformity with the requirements for the degree of Masters of Applied Science
Graduate Department of Institute for Aerospace Studies
University of Toronto

Copyright © 2013 by Cody Paige
Abstract

Extension of the ADjoint Approach to a Laminar Navier–Stokes Solver

Cody Paige

Masters of Applied Science

Graduate Department of Institute for Aerospace Studies

University of Toronto

2013

The use of adjoint methods is common in computational fluid dynamics to reduce the cost of the sensitivity analysis in an optimization cycle. The forward mode ADjoint is a combination of an adjoint sensitivity analysis method with a forward mode automatic differentiation (AD) and is a modification of the reverse mode ADjoint method proposed by Mader et al.\cite{Mader}. A colouring acceleration technique is presented to reduce the computational cost increase associated with forward mode AD. The forward mode AD facilitates the implementation of the laminar Navier–Stokes (NS) equations. The forward mode ADjoint method is applied to a three-dimensional computational fluid dynamics solver. The resulting Euler and viscous ADjoint sensitivities are compared to the reverse mode Euler ADjoint derivatives and a complex-step method to demonstrate the reduced computational cost and accuracy. Both comparisons demonstrate the benefits of the colouring method and the practicality of using a forward mode AD.
Acknowledgements

I want to thank all of the members of my lab group for their support and motivation. In particular Dr. Charles Mader and Gaetan Kenway for showing me the ropes and helping me with every question I had for them. Their insights and patience were invaluable to my work and truly made this thesis a possibility. I also would like to thank my family and friends for their understanding in this endeavour, for pushing me to continue and forgiving my absence and my frustrations throughout. Finally I would like to thank my supervisor Dr. Martins whose encouragement through an expectation of greatness pushed me to accomplish more.
# Contents

1 Introduction  
  1.1 Motivation ............................................................ 2  
  1.2 Literature Review ..................................................... 3  
  1.3 Thesis Overview ...................................................... 8  
  1.4 Research Contributions ........................................... 8  

2 SUmb Flow Solver and the MDO Framework  
  2.1 SUmb ................................................................. 12  
  2.2 pyWarp ............................................................... 16  
  2.3 DVGeometry .......................................................... 16  
  2.4 pySNOPT ............................................................. 17  
  2.5 MDO Framework Summary ....................................... 17  

3 The ADjoint Method  
  3.1 Adjoint Equations .................................................... 19  
  3.2 Automatic Differentiation .......................................... 21  
    3.2.1 Forward Mode .................................................. 23  
    3.2.2 Reverse Mode .................................................. 25  
    3.2.3 AD For Sensitivity Analysis .................................. 26  
  3.3 Colouring Method ................................................... 27  
  3.4 The ADjoint .......................................................... 31
List of Figures

1.1 Optimum elliptic and aerostructural lift distributions\textsuperscript{[10]} ........................................... 4

2.1 MDO for Aircraft Configuration with High-Fidelity (MACH) flowchart. An aerostructural gradient-based optimization using the ADjoint approach. 10

2.2 Subroutine call graph for the SUmb flow solver. ............................... 13

2.3 Schematic of the adjoint-based optimization algorithm ......................... 14

2.4 Schematic of the adjoint-based optimization algorithm ......................... 18

3.1 Framework of $\partial R/\partial w$ computation using AD. ................................. 27

3.2 Stencil of affected cells in an Euler ADjoint calculation ............................ 28

3.3 Perturbation colouring scheme for a one-dimensional grid, where each point indicates a perturbation\textsuperscript{[7]} ......................................................... 29

3.4 Residual computation for the forward mode Euler ADjoint. ....................... 30

3.5 Colouring used for forward mode Euler ADjoint $\partial R/\partial w$. ........................ 31

3.6 Original SUmb solver residual computation framework. ............................ 35

3.7 Residual computation for the forward mode Euler ADjoint. ....................... 36

3.8 $\partial R/\partial x$ colour convention and stencil. ................................................. 37

3.9 Colouring used for forward mode ADjoint $\partial R/\partial x$. .......................... 38

4.1 Residual computation for the forward mode ADjoint method extended to the laminar NS equations. ................................................................. 43

4.2 $5 \times 5 \times 5$ stencil with dense inner $3 \times 3$ stencil used for viscous colouring. 46
4.3 $5 \times 5 \times 5$ stencil with dense inner $3 \times 3$ stencil used for viscous colouring. 47
4.4 Colouring used for forward mode Euler ADjoint $\partial R/\partial w$. 48
4.5 Verification routine call graph. 50

5.1 Single block bump test case computational domain 52
5.2 Euler case surface density distribution. 53
5.3 $l^2$ norm of the error in the density state of the flux Jacobians. 56
5.4 Viscous case surface density distribution. 58
5.5 $l^2$ norm of the error in the density state of the flux Jacobians. 60

A.1 $l^2$ norm of the velocity state in the x-direction of the flux Jacobians. 68
A.2 $l^2$ norm of the velocity state in the y-direction of the flux Jacobians. 69
A.3 $l^2$ norm of the velocity state in the z-direction of the flux Jacobians. 70
A.4 $l^2$ norm of the velocity state in the x-direction of the flux Jacobians. 70
A.5 $l^2$ norm of the velocity state in the y-direction of the flux Jacobians. 71
A.6 $l^2$ norm of the velocity state in the z-direction of the flux Jacobians. 71
List of Tables

5.1 Accuracy validations ........................................ 54
5.2 Timing benchmarks .......................................... 57
5.3 Accuracy validations .......................................... 59
5.4 Timing benchmarks .......................................... 61
Chapter 1

Introduction

Airliners have had a tube-shaped fuselage, swept wing and podded engine design for over 50 years. As environmental considerations become more prevalent and the economy pushes for cheaper, more efficient aircraft, there is pressure for considering designs other than conventional. The push for unconventional aircraft has led organizations including NASA and Boeing to look into designs such as the blended wing body (BWB), the double-bubble concept, and a variety of wing designs to achieve the required improvements in performance. Unconventional aircraft, such as the BWB, are difficult to design since no historical data is available. High-fidelity multidisciplinary design optimization (MDO) facilitates the design of unconventional aircraft, since it relies heavily on physics-based models instead of statistical data.

In the development of a successful optimizer, certain factors must be taken into consideration before making an appropriate choice. In large-scale engineering systems numerous disciplines must be considered. There are multiple physical systems and large numbers of inputs, and thus, long computation times. In aircraft design in particular, the multidisciplinary systems are complex, with potential for millions of simulation requirements to take all of the aircraft configurations, flight conditions, loadings, etc., into account. The adjoint method is particularly useful in the sensitivity computations for gradient-
based design optimization because it is an extremely efficient approach for computing the sensitivity of a single function of interest with respect to many parameters\textsuperscript{[1–3]}.

One of the main difficulties with an adjoint solver is in the differentiation of the system. The derivatives can be done by hand, but often require simplifying assumptions during the formulation of the adjoint and become prohibitively difficult to implement with more complex systems. For complex models, such as those encountered in high-fidelity aerostructural optimization, an efficient method of sensitivity analysis is essential for an effective method of optimization\textsuperscript{[4]}.

1.1 Motivation

MDO for Aircraft Configuration with High-Fidelity (MACH) is a framework for aerostructural gradient-based optimization that uses a coupled adjoint approach. MACH uses a parallel finite-element code — Toolkit for the Analysis of Composite Structures (TACS)\textsuperscript{[5]} — for the structural analysis. This structural solver is coupled to the aerodynamic solver SUmb. The aerodynamic sensitivity analysis is done using the ADjoint method\textsuperscript{[1]}. The ADjoint is an adjoint approach that uses automatic differentiation (AD). This approach is a good alternative to hand differentiation, finite-difference methods and complex variables in deriving the partial derivatives of the adjoint for the aerodynamic sensitivity analysis.

The ADjoint method overcomes the complexities of the differentiation by using Tape-nade, an AD tool, to implement the computation of the partial derivatives in the adjoint solver. The original ADjoint method developed by Mader\textsuperscript{[1]} used reverse mode AD to calculate the residuals and force derivatives. It was successfully tested for the Euler equations using hundreds of design variables\textsuperscript{[6]}. The next step in the development of the ADjoint method is the extension to the laminar Navier–Stokes (laminar NS) equation. The reverse mode AD method, though highly efficient, was difficult to develop further.
In order to simplify the development of the code, a forward AD method is implemented using a colouring acceleration technique\cite{7} to increase efficiency. This technique not only reduces implementation time but also shows a decrease in computational cost compared to the reverse mode ADjoint.

In extending the ADjoint method to the laminar NS equation, viscous drag can be considered in wing optimizations, including the planform area, thus adding a penalty for wing area, which would otherwise be ignored. A penalty is also added for reducing the size of the chord, for example at the tip, by taking into account the additional viscous drag due to smaller Reynolds numbers. The goal of this project is to develop an efficient implementation of the ADjoint method for a discrete laminar NS adjoint equation for the ADjoint.

1.2 Literature Review

Traditionally, the optimization of an aircraft is done by first optimizing one discipline, for example aerodynamics, then passing this aerodynamic optimum to a second discipline, such as structures, where adjustments will be made to optimize for the structure of the aircraft. This will then be returned for a new aerodynamic optimization taking into account the adjustments made for structures. This type of sequential discipline optimization cannot always converge to the true optimum of the coupled system. Wakayama\cite{8} showed, for example, that in order to obtain an optimal planform design of wings and wing-tail configurations, multiple disciplines had to be included in the optimization. As well, multiple real-world constraints such as induced, profile and compressibility drag, bending stress and buckling, static aeroelasticity, etc., needed to be included to obtain a true optimum design. Martins \textit{et al.}\cite{9} and Chittick and Martins\cite{10} showed this by looking at the Breguet range formula for jet-powered aircraft. The Breguet range equation demonstrates a real-world objective function that expresses a trade off between the drag
and the empty weight of the aircraft. An aerostructural optimization was performed constraining the stress and thickness of the wing being optimized and maximizing for range. Both the aerodynamic and structural optima then are considered simultaneously and a maximum is reached which exceeds that computed using a sequential optimization. This is shown in Figure 1.1 in which the multidisciplinary feasible (MDF) method shows the true optimum compared to the step-wise sequential method.

**Figure 1.1:** Optimum elliptic and aerostructural lift distributions

MDO uses a large number of design variables and thus the optimization algorithm used must be chosen with this in mind. Gradient-based methods use sensitivity information to identify a search direction within the design space. A one-dimensional search in that direction is then performed to find the new search direction. The use of a gradient greatly reduces the number of function evaluations required to determine the optimum solution. A gradient-based method was used by Hicken and Zingg for aero-
dynamic shape optimization. The flow solver incorporated into the optimization algo-

rithm used a second-order-accurate finite-difference discretization and a Newton–Krylov
solution strategy for the calculation of the sensitivities.

The calculation of these sensitivities can often be the most costly step in the optimization cycle, particularly with the large number of design variables seen in aerostructural optimizations. When determining a sensitivity analysis method it is important to consider the ratio of the number of outputs to the number of inputs, as well as the computational efficiency of the method. The adjoint method is a commonly used method for sensitivity analysis in shape optimization. The adjoint is accurate and efficient but does not depend on the number of design variables. As will be described in more detail in Chapter 3, the adjoint approach takes advantage of partial sensitivity calculations to maintain the low computational cost. This method was first developed for computational fluid dynamic (CFD) models by Pironneau\textsuperscript{[14]} to demonstrate the minimization of drag over a body submersed in a viscous fluid. Since then the adjoint method has become increasingly popular in aerodynamic shape optimization. It has been used on airfoil and wing shape optimizations in Euler flows\textsuperscript{[15,16]}, viscous flows\textsuperscript{[17,18]}, and laminar-turbulent flow prediction\textsuperscript{[19]} alike. Because of its relative independence of number of design variables it has been effectively used to optimize complex aircraft configurations including multiple design points, subject to geometric constraints\textsuperscript{[20,21]}.

While adjoint methods are commonly used in aerodynamic shape optimization, they can be particularly difficult to implement as they require knowledge of the governing equations. It is also important to consider the human effort required in the implementation of the method.

The finite-difference method is a traditional method used to estimate sensitivities and can be relatively simple to implement. Consider the first-order forward difference approximation given by

$$\frac{df (x_j)}{dx_j} = \frac{f (x_j + h) - f (x_j)}{h} + O(h)$$ \hspace{1cm} (1.1)

where $h$ is the finite-difference interval, or step. The truncation error is $O(h)$. For each perturbation of $x_j$, $f$ must be recalculated, thus the cost is proportional to the
number of design variables for a first-order approximation. In order to achieve more accuracy, higher-order approximations must be used that require more function evaluations. Finally, the step size must be small in order to minimize truncation error, resulting in subtractive cancellation. A loss in speed and accuracy also occurs when using finite-difference methods\(^6,22\). Despite this, it is a useful benchmarking tool in the implementation of more complex methods.

A second option is the complex step derivative approximation, which uses complex calculus to obtain a second order approximation by perturbing the imaginary part of the function by a pure imaginary step making it robust to changes in step size\(^22\). However, the computational cost is still proportional to the number of design variables. Although the cost is much higher due to its use of complex mathematics, the complex step method is also a useful benchmarking tool, since it is as accurate as the original computation. This high degree of accuracy is required in order to locate all of the errors when developing the new code. Martins et al.\(^23\) developed an automated complex step method for aerodynamic sensitivity analysis that is used to simplify the implementation and testing of alternative methods.

The difficulty of implementation can be overcome by combining an adjoint method with automatic differentiation (AD). AD determines the derivatives by the use of the chain rule by propagating a variation due to a single input through the algorithm. AD can be run in forward mode (a forward propagation of the perturbation) or reverse mode (a backward propagation). The reverse mode is independent of design variables but is more difficult to implement. These difficulties will be discussed further along with the adjoint method in Chapter 3.

The original ADjoint method employed reverse mode AD with an adjoint method to generate the discrete adjoint operator. This was developed by Mader\(^6\). The methodology of applying AD selectively to an adjoint code is discussed in Marta et al.\(^24\) presenting the benefits of the method in reducing long implementation times. The accuracy
and efficiency of the code are demonstrated on multiple test cases and used for a wing optimization in Mader et al.\textsuperscript{[1]} In Mader and Martins\textsuperscript{[25]} the ADjoint method was extended to the computation of static, dynamic and transient aircraft stability derivatives. The computational cost reduction and accuracy of the time-spectral ADjoint method are demonstrated on an optimization of an oscillating ONERA M6 wing in Mader and Martins\textsuperscript{[26]}. Although this combination of methods is extremely efficient for sensitivity analysis\textsuperscript{[26]}, it does introduce some complications in implementation due to the nature of reverse mode AD as is discussed in Chapter 3. Forward mode AD has been shown by Bischof et al.\textsuperscript{[27]} to be an accurate and efficient method to obtain derivatives for a 3D, thin-layer Navier–Stokes, multigrid flow solver.

Forward mode AD is relatively easy to implement and very similar to the way the complex step method works. However, it is dependent on the number of design variables. In order to reduce the computational cost while still maintaining the benefits of AD, a secondary method called \textit{colouring} is used. The partial derivatives of the gradients are sparse and colouring takes advantage of this fact. It uses a stencil of affected cells to reduce the number of forward mode evaluations required to populate the entire Jacobian\textsuperscript{[7]}. When the dimension of the matrix is large and the evaluation of each gradient is expensive, it is necessary to take the sparsity structure of the matrix into account. Several methods for doing this have been developed including the estimation of Jacobian and Hessian matrices arising in the finite-difference approximation of partial differential equations in Goldfarb and Toint\textsuperscript{[28]}. Both symmetric and non-symmetric matrices were considered using ‘computational molecules’ or stencils of the finite-difference operator associated with the Jacobian matrix to determine the groupings used to reduce the number of function vector differences. A similar approach is developed by Nielsen and Kleb\textsuperscript{[7]}, in which node colouring is used to establish a stencil which may be simultaneously perturbed in a complex-step adjoint formulation for use in computational fluid dynamics (CFD). By using a colouring method, the efficiency of the forward mode ADjoint improves and
the difficulties of implementing the reverse mode ADjoint are eliminated.

1.3 Thesis Overview

In Chapter 2 we discuss the implementation of the laminar NS ADjoint sensitivities in the CFD solver. This chapter also discusses how the solver fits into MACH and describes the components of MACH being used, including the geometry and grid manipulation tools. Chapter 3 starts with a description of the forward mode Euler ADjoint solver and an overview of the concepts of the adjoint method and of AD. The basic concept of the colouring method is then introduced. The colouring method’s implementation into the ADjoint is developed using the reverse mode ADjoint method as a baseline for comparison. The addition of the viscous fluxes as well as viscous boundary conditions is described in Chapter 4. The colouring using a viscous stencil is outlined here, as well as the implementation of the new method into the ADjoint. Chapter 5 presents a simple test case that is used to compare the forward mode ADjoint with a colouring acceleration technique to the reverse mode ADjoint method. The viscous additions, facilitated by the colouring method, are checked against the original complex step solver routines also using the simple test case with viscous boundary conditions. The capabilities of the forward mode laminar NS ADjoint are demonstrated by the test case. Finally, the conclusions drawn from the benchmarking, validations and the implementations are presented in Chapter 6.

1.4 Research Contributions

There are two main contributions of this work. The first is the simplification of the original reverse mode ADjoint code to facilitate revisions and additions using a colouring method and a forward AD approach. By using a forward AD approach, the adjoint code matches the original solver much more closely, thus allowing for a simplified implementa-
tion of laminar NS and, in the future, the RANS equations. The use of this method also improved computational efficiency while maintaining the accuracy of the original reverse mode ADjoint method. The second contribution was the implementation and testing of the laminar NS equation into the ADjoint code. The test cases chosen demonstrate the usefulness of the laminar NS implementation, while outlining the benefits of the colouring method used.
Chapter 2

SUmb Flow Solver and the MDO Framework

The MDO of aircraft configurations with high-fidelity (MACH) is a multi-disciplinary optimization method and is capable of optimizing aerodynamics and structures simultaneously. MACH’s overall structure is shown in Figure 2.1. MACH currently uses SUmb

![Figure 2.1: MDO for Aircraft Configuration with High-Fidelity (MACH) flowchart. An aerostructural gradient-based optimization using the ADjoint approach.](image)

(Stanford University multi-block), a three-dimensional CFD flow solver, to develop the residual of the aerodynamic system. Work is also under way to integrate Diablo, a par-
allel CFD code that solves the NS equations, incorporates turbulence models and has
adjoint capability\cite{29}. The structural solver is the Toolbox for the Analysis of Composite Structures pyTACS, a parallel finite-element solver\cite{30}, shown in Figure 2.1. This work focused only on the aerodynamic solver and so TACS is not discussed here. A complete description of TACS can be found in Kennedy and Martins\cite{5}. The geometry and its derivatives are handled by DVGeometry. pyWarp, an algebraic warping module, is used to generate and manipulate the mesh according to the effects of perturbations. pySNOPT is the gradient based optimization algorithm\cite{31}.

The majority of the routines used in pySUmb, pyWarp, DVGeometry and pySNOPT are written in Fortran. For a variety of useful features and a number of useful tools, which are described in detail in Alonso et al.\cite{32}, these routines are all wrapped in Python. This is described in more detail for pySNOPT in Perez et al.\cite{33}. Since TACS is written in C++ and can also be wrapped in Python, this is particularly useful in integrating the different languages into the same optimization method. A tool called f2py, an automatic interface generator, allows a coder to transfer data and call functions between Python and Fortran with little or no knowledge of the coding details necessary to do so\cite{32}. f2py works by taking the Fortran source code and generating a signature file with the extension pyf that contains all of the information about the data, functions and their arguments in the Fortran source. This can then be edited by the user so that when single Fortran files are added to the source code the pyf file does not need to be recreated. f2py then creates a C code wrapper and compiles the final .so file that is read by the top level Python routines. The forward mode ADjoint method routines were written in Fortran and wrapped in Python in this manner.

The aerodynamic components of MACH are discussed here since the forward mode ADjoint is a component of the aerodynamic solver used in MACH.

When considering the total sensitivity equation for the entire aerodynamic framework
Chapter 2. SUmb Flow Solver and the MDO Framework

The total sensitivity can be written as,

$$\frac{dI}{dx_{DV}} = \frac{dx_{GEO}}{dx_{DV}} \left( \left[ \frac{dx_{CFD_s}}{dx_{CFD_v}} \right] \left[ \frac{\partial I}{\partial x_v} \right] + \left[ \frac{dx_{CFD_s}}{dx_{CFD_v}} \right] \left[ \frac{\partial R}{\partial x_v} \right] \psi^T \right)$$

(2.1)

where $s$ represents the surface derivatives, $v$ the volume derivatives and $DV$ the total design variable derivatives. $I$ is the function of interest, $x$ the design variables and $w$ represents the states. $R$ are the residuals and $\psi$ is the adjoint vector. Each component can be directly related to a part of the framework. $\partial I/\partial x_v$, $\partial R/\partial x_v$ and $\psi^T$ are all solved using the ADjoint in pySUmb, $dx_{CFD_s}/dx_{CFD_v}$ is computed using the warping algorithm, pyWarp, and $dx_{GEO_s}/dx_{DV}$ uses the geometry code, DVGeometry.

The total aerodynamic derivative is developed in Chapter 3 and is shown here for reference,

$$\frac{dC_D}{dM_\infty} = \frac{\partial C_D}{\partial M_\infty} + \psi^T \frac{\partial R}{\partial M_\infty}.$$  

or in more general terms,

$$\frac{dI}{d \alpha} = \frac{\partial I}{\partial \alpha} + \psi^T \frac{\partial R}{\partial \alpha}.$$ 

where $\alpha$ is the $x$ component representing the aerodynamic design variables such as $M_\infty$, angle of attack, twist, etc.

The total aerodynamic derivative is solved entirely by pySUmb and its implementation is described in Chapter 3. The total surface derivative also follows the described implementation, however, in order to include the shape parameterization and mesh movement pyWarp and DVGeometry are required.

Each component used in the sensitivity analysis in MACH will be described along with the implementation.

2.1 SUmb

SUmb, developed at Stanford University, is a finite-volume, cell-centered multi-block solver set up for the RANS equations (steady, unsteady and time-spectral) and has options for multiple turbulence models with one and two equations. The subroutine call
graph for SUmb is shown in Figure 2.2. SUmb is used to determine a flow solution as well as the components that are used for the sensitivity calculations in the ADjoint method. Figure 2.3 shows a reduced flowchart of the components of the optimization method where SUmb is involved. The flow solver is called using a Python driver routine. The flow is initialized and solved following the original SUmb solver routines with specified user inputs defining the equation type, input parameters, function of interest, etc. Once the solver has converged on a solution the volume solution file and surface solution files are saved. The flow solution, \( w \) in Figure 2.3, is then passed to the ADjoint solver. The ADjoint is initialized setting the design variables from the user inputs. If the function of interest is an aerodynamic objective the right hand side of the adjoint equation will be non-zero. The adjoint equation is developed in Chapter 3 and is shown here for reference.

\[
\left[ \frac{\partial R}{\partial w} \right]^T \psi = \left[ \frac{\partial I}{\partial w} \right]^T
\]
Once initialized, the adjoint matrix is setup using the current flow solution. Here the forward mode ADjoint method is used as described in Chapter 3 to calculate $\partial R/\partial w$ and $\partial R/\partial x$ or $\partial R/\partial \alpha$. The Jacobian matrix $\partial R/\partial w$ is a very large and very sparse matrix. It is independent of the function of interest and the selected design variable, depending only on the governing equations, their discretization, the boundary conditions and the states of the flow at the converged solution. Because of this, the Jacobian can be calculated once with the converged flow solution and saved as a sparse matrix to be used in the adjoint calculation for each function of interest and design variable.

The derivatives used to compute the discrete adjoint equations are stored in the portable, extensible toolkit for scientific computation, PETSc. PETSc is a suite of data structures and routines for the scalable, parallel solution of scientific applications modelled by partial differential equations\[^6\]. First the PETSc variables are created to store the residual matrices. In the forward mode ADjoint the transpose of the residual matrix of the discrete adjoint problem, $(\partial R/\partial w)^T$ is computed using the automatically differentiated routines generated by Tapenade. These computations are done in a block-based method described in Chapter 3. Each block is set to a single processor with two sets of halo cells around the exterior of the block in which the neighbouring block’s data is stored. The halo cells are not stored in the PETSc matrix but are copied between
processors such that no special treatments are required at the boundaries.

Within each block, using a colouring technique, the boundary conditions are applied. The internal block boundaries are considered as boundary conditions using penalty terms. Next the inviscid fluxes are calculated, then the dissipation fluxes. The viscous computations, discussed in Chapter 4, are done within this differentiated block-based routine for the calculation of the viscous fluxes. For each colour the generated values of \( \partial R/\partial w \) are stored in a PETSc sparse matrix format.

The ordering of the residuals in the ADjoint matrix is based on the global cell numbering such that the five states are saved as a block of data for the global cell. The residual matrix \( \partial R/\partial x \) or \( \partial R/\partial \alpha \) is similarly computed, however the metric transformations are taken into account in the residual calculation. The values are saved in its respective PETSc matrix.

Next, the right hand side of the adjoint equation is computed using the cost function as the function of interest. Both \( \partial I/\partial w \) and \( \partial I/\partial x \) or \( \partial I/\partial \alpha \) are computed and saved as PETSc matrices using the process discussed in Chapter 3. The partial derivative term, \( \partial I/\partial x \), only presents itself on the surface nodes. For simplicity, however, the entire volume mesh is included in the computation.

With all of the partial derivatives saved in PETSc’s sparse data structures the adjoint itself, \( \psi \), can now be computed. This is done using PETSc’s scalable linear equations solver, KSP\textsuperscript{[35]}. It is a combination of a Krylov subspace iterative method and a preconditioner or a sequential direct solver. In this case it is used as an iterative solver of the linear system described by the adjoint equation.

pySUmb uses pyWarp directly to create the surface derivatives necessary for the total sensitivity. This will be discussed in the following section.

Using the adjoint vector, the total aerodynamic derivatives, such as the derivatives with respect to Mach number or angle of attack, can also be computed. This, unlike the surface derivatives, does not require a mesh warping for the partial derivatives.
2.2 pyWarp

pyWarp is an algebraic, multi-block, mesh warping algorithm for structured meshes. pyWarp uses the original distribution of interior points to scale the mesh linearly. An algebraic warping algorithm is applied to each surface grid-line radiating from a displaced surface. An example of mesh warping and a more detailed description of the process can be found in Kenway et al.\cite{30}.

pyWarp uses a top level python driver module, \texttt{MultiBlockMesh.py}, that interacts with a structured multi-block mesh, typically in a 3D CFD program, as is the case here. It is here that the multi-block mesh is initialized. This is done by setting the indices defining the transformation of an external solver grid to the original design grid. This is done for both the external mesh and force indicies.

With the mesh in place, pyWarp may be used with the solver and warping functions. Examining Equation (2.1), the component \( dx_{\text{CFD}} / dx_{\text{CFD}} \) is the mesh warping derivative. This is computed in pyWarp and is used to transform the partial volume derivatives, \( \partial I / \partial x_v \) and \( \partial R / \partial x_v \) into surface derivatives for the total sensitivity.

The final two components of Equation (2.1) are computed in DVGeometry.

2.3 DVGeometry

DVGeometry is a geometry surfacing engine. It performs multiple functions including producing surfaces from cross sections, fitting surfaces and has built-in design variable handling. DVGeometry deals with all the details of taking user supplied design variables and mapping them to the discrete surfaces on the CFD disciplines. Free form deformation (FFD) volumes are currently used to create the surface geometries. The total sensitivity is computed by taking the total derivative of a function of interest, \( I \), with respect to the points controlled by the processor. DVGeometry generates the derivative of the geometry with respect to the design variables, \( dx_{\text{GEO}} / dx_{\text{DV}} \). This completes the
necessary derivatives to form the total sensitivity. The total sensitivity can then be used by the optimizer to find a new design point.

2.4 pySNOPT

SNOPT\textsuperscript{[11]} is a gradient-based optimization algorithm based on the sequential quadratic programming (SQP) method. SNOPT is specifically designed for optimization problems with many thousands of constraints. The SQP method has proven highly effective for solving constrained nonlinear optimization problems. The coupled adjoint method more efficiently computes the gradients of a function of interest than a nonlinear function, thus, the computational effort required to find an optimal solution is significantly reduced by SNOPT\textsuperscript{[4]}. pySNOPT is the Python wrapped version of SNOPT allowing for the integration into MACH\textsuperscript{[33]}.

The aerodynamic and geometric variables are passed to pySNOPT as well as the constraints. The sensitivities and function of interest are then used to run the optimization.

2.5 MDO Framework Summary

Now that the various components of the optimization framework have been established the overall functionality can be outlined. Figure 2.4 shows the process which each optimization cycle follows. First, DVGeometry generates the geometry and passes it to pyWarp. pyWarp uses this geometry to establish a volume mesh. This is then passed to pySUmb, which calculates the flow solution. pySUmb then applies the ADjoint method to determine the total surface derivative. pySUmb uses pyWarp directly to compute $dx_v/dx_s$, which is used to compute the total surface derivative, as described above. DV-Geometry then computes the total sensitivity, $dI/dx_{DV}$, using the partial derivatives calculated in pySUmb and applying them to $dx_{GEO_s}/dx_{D}$. It is this total derivative that is returned to pySNOPT. pySNOPT uses the derivative information and the flow
solution to select a new design point. This new design point is then passed to DVGeometry to generate a new geometry and begin the process again. The computational efficiency of the optimization process depends heavily on the efficiency of the derivative calculations.

In order to test the implementation of the forward mode ADjoint using colouring a sensitivity analysis of a simple test case was done computing all of the pySUmb components of the total aerodynamic derivative and the total surface derivative. The test case results are presented in Chapter 5.
Chapter 3

The ADjoint Method

3.1 Adjoint Equations

The adjoint approach is well known for its capability to efficiently compute the derivatives for design problems with large numbers of design variables and small numbers of functions of interest\textsuperscript{[6]}. If a generic function of interest, \( I \), is considered, which could represent lift coefficient, drag coefficient or cruise Mach number, the design optimization problem can be written as,

\[
\text{minimize} \quad I(w(x), x) \\
\text{w.r.t.} \quad x \\
\text{subject to} \quad R(w(x), x) = 0 \\
C_i(w(x), x) = 0 \\
C_i(x) \geq 0 \\
i = 1, \ldots, m,
\]

where \( R(w(x), x) = 0 \) represents the discrete flow equations and boundary conditions that must be satisfied, \( C_i(w(x), x) = 0 \) are \( m \) additional constraints, \( C_i(x) \geq 0 \) are the inequality constraints, \( x \) is the vector of design variables and \( w \) is the vector of states.
Chapter 3. The ADjoint Method

The design variables, $x$ can be physical variables such as cruise Mach number, angle of attack, sideslip angle, as well as the shape variables.

The total sensitivity of the the function of interest $I$ can be found using the chain rule,

$$\frac{dI}{dx} = \frac{\partial I}{\partial x} + \frac{\partial I}{\partial w} \frac{dw}{dx} \quad (3.1)$$

The function of interest can also be a constraint. The state vector $w$ is implicitly dependent on the design variables $x$ through the solution of the governing equations $R(w(x), x) = 0$. Because of this, and the $R(w(x), x) = 0$ constraint, a total sensitivity for the purpose of this work must include a converged flow solution for every new set of design variables. The partial sensitivities are derivatives evaluated for a constant set of states requiring only a single computation of a converged flow solution. For large numbers of design variables, as is common in aerodynamic shape optimization, the computational cost of Equation (3.1) becomes prohibitive as the flow would need to be reconverged for each design variable in $dw/dx$.

Using the fact that any variation of the design variables must remain feasible with respect to the governing equations, we can write the derivative of the residual as,

$$\frac{dR}{dx} = \frac{\partial R}{\partial x} + \frac{\partial R}{\partial w} \frac{dw}{dx} = 0 \quad (3.2)$$

The total derivative of the states with respect to the design variables, $dw/dx$, may then be obtained using,

$$\frac{dw}{dx} = - \left[ \frac{\partial R}{\partial w} \right]^{-1} \frac{\partial R}{\partial x} \quad (3.3)$$

We can substitute this into the derivative, Equation (3.1), to obtain,

$$\frac{dI}{dx} = \frac{\partial I}{\partial x} - \frac{\partial I}{\partial w} \left[ \frac{\partial R}{\partial w} \right]^{-1} \frac{\partial R}{\partial x} \quad (3.4)$$

This total sensitivity equation can be used in the direct method where the system generated by the last two terms is solved, thus solving for $dw/dx$. However, in doing so, the system returns to its original dependence on the design variables and a linear system
solution is required for each $x$. The adjoint method solves the system generated by the second and third terms on the right hand side of Equation (3.4). This has the form,

$$\psi = \left[ \frac{\partial I}{\partial w} \right]^T \frac{\partial R}{\partial w}$$  \hspace{1cm} (3.5)

where $\psi$ represents the adjoint vector. This is more traditionally written as,

$$\left[ \frac{\partial R}{\partial w} \right]^T \psi = \left[ \frac{\partial I}{\partial w} \right]^T.$$  \hspace{1cm} (3.6)

Thus a linear system is solved for each function of interest instead of each design variable. Note that the right hand side of Equation (3.6) needs to be recalculated for each constraint as well, because gradient-based optimization also requires the derivatives of all constraints with respect to all design variables. Once the adjoint vector has been calculated, the total sensitivity of the function of interest may be solved by,

$$\frac{dI}{dx} = \frac{\partial I}{\partial x} - \psi^T \frac{\partial R}{\partial x}.$$  \hspace{1cm} (3.7)

This total derivative is then used by the optimizer to compute the search direction.

Because the cost of computing the gradient using the adjoint method is independent of the total derivative $dw/dx$, the gradient of the function of interest can be calculated with a very large vector of design variables without having to re-compute a converged flow solution. We now consider the practical implementation of the partial derivatives in Equations (3.6) and Equation (3.7), specifically AD.

### 3.2 Automatic Differentiation

All computer programs, no matter how complicated, can be broken down into a sequence of elementary arithmetic operations and functions. Automatic differentiation (AD) takes advantage of this by applying the chain rule to these operations and computing derivatives of arbitrary order with an accuracy to working precision. AD applies the chain rule to the function rather than the formula. This means that derivatives can be defined for
computer subroutines and programs as well as functions of interest. Symbolic differentiation, on the other hand, requires explicit formulas to determine the derivatives. Symbolic differentiation uses the same derivative definitions as AD, but the former requires more resources and a more detailed knowledge of the formula being differentiated. AD also differs from numerical differentiation in that it is not subject to truncation error caused by increases in step size. While a decrease in step size may help to reduce truncation errors in finite differences, it results in an increase in round-off error. Although AD is exact in theory, in practice it is also subject to round-off error. The round-off error may be kept to a minimum, however, because there is no truncation error with which to trade off. AD is more generally applicable than symbolic differentiation and is more accurate than numerical differentiation.\textsuperscript{[36]}

For the most basic equation, \( f(x) = f(h(x)) \), the chain rule is,

\[
\frac{df}{dx} = \frac{df}{dh} \frac{dh}{dx}
\]  \hspace{1cm} (3.8)

Two distinct modes of AD are available: the forward mode and reverse mode. The forward mode applies the chain rule from right to left. In the case of the formula given above \( \frac{dh}{dx} \) is computed first followed by \( \frac{dg}{dh} \) and the cost of these operations is dependent on the number of inputs. The reverse mode applies the chain rule from left to right and its cost is dependent instead on the number of outputs. These are discussed in more detail below. For any function or program, AD first creates additional code that computes the corresponding derivatives.\textsuperscript{[36]} For illustration purposes, consider the following example,

\[
f(x_1, x_2) = x_1 x_2 + \sin(x_1)
\]  \hspace{1cm} (3.9)

This function can be written as the sequence of elementary operations on the work
variables \( q_i \) giving the following sequence,

\[
\begin{align*}
q_1 &= x_1 \\
q_2 &= x_2 \\
q_3 &= q_1 q_2 \\
q_4 &= \sin(q_1) \\
q_5 &= q_3 + q_4
\end{align*}
\] (3.10)

It is this sequence that is then used to compute the derivative of Equation (3.9).

### 3.2.1 Forward Mode

Forward mode AD is the more basic and more intuitive of the two modes. Assuming in Equation (3.9) that \( x_1 \) and \( x_2 \) are independent inputs, the rules of differentiation are applied to the sequence in Equation (3.10), as follows,

\[
\begin{align*}
\Delta q_1 &= \Delta x_1 \\
\Delta q_2 &= \Delta x_2 \\
\Delta q_3 &= \Delta q_1 q_2 + q_1 \Delta q_2 \\
\Delta q_4 &= \cos(q_1) \Delta q_1 \\
\Delta q_5 &= \Delta q_3 + \Delta q_4
\end{align*}
\] (3.11)

Once the sequence and its corresponding gradients for the function in Equation (3.9) are known, \( x_1 \) and \( x_2 \) can be seeded to determine the gradient of the function. Since \( x_1 \) and \( x_2 \) are assumed to be independent inputs, seeding each independently means to set the variation of one to 1 while the other remains zero such that \( \Delta x_1 = [1, 0] \) and \( \Delta x_2 = [0, 1] \). Forward mode AD sweeps over the computations in Equation (3.11) twice, once for each
input as follows,

\[ \Delta f_{x_1} = \Delta q_{5x_1} \]
\[ = (\Delta q_{3x_1} + \Delta q_{4x_1}) \]
\[ = ((1)q_2 + q_1(0) + \cos(q_1)(1)) \]
\[ = q_2 + \cos(q_1) \]
\[ = x_2 + \cos(x_1) \]

\[ \Delta f_{x_2} = \Delta q_{5x_2} \]
\[ = (\Delta q_{3x_2} + \Delta q_{4x_2}) \]
\[ = ((0)q_2 + q_1(1) + \cos(q_1)(0)) \]
\[ = q_1 \]
\[ = x_1 \]

These are then the expected derivatives for the original function in Equation (3.9). This can be written in a more general format by considering the general sequence \( q = (q_1, ..., q_n) \). Considering \( m \) input variables and \( p \) output variables the sequence becomes \( q = (q_1, ..., q_m, q_{m-p+1}, ..., q_n) \). For \( i > m \), each \( q_i \) must have a dependence on some member of the sequence prior to \( i \). If \( k < i \) then the entry \( q_i \) of the sequence must depend explicitly on \( q_k \). The forward mode can then be written as the chain rule summation given by Rall and Corliss\[^{36}\],

\[ \Delta q_i = \sum \frac{\partial q_i}{\partial q_k} \Delta q_k \] (3.12)

for \( i = m + 1, ..., n \) and \( k < i \). The forward mode AD evaluates the gradients of the intermediate variables first such that \( \Delta q_1, ..., \Delta q_{i-1} \) are known prior to the evaluation of \( \Delta q_i \). It is easy to see then that the forward mode builds up the derivative information as it progresses forward through the algorithm producing the derivative information for all of the output variables with respect to a single seeded input variable.
Chapter 3. The ADjoint Method

25

The computational cost of the forward mode is proportional to the number of inputs. Although the forward mode is intuitive and easy to implement, it is not, on its own, an ideal choice for a gradient-based optimizer with a large number of inputs and only a small number of outputs.

3.2.2 Reverse Mode

The reverse mode, though less intuitive, is dependent only on the number of outputs. When considering the example given, it is somewhat easier to understand reverse mode AD by considering the partial derivatives of \( f \). For the inputs \( x_1 \) and \( x_2 \), the derivatives are the following,

\[
\frac{\partial q_5}{\partial q_1} = \frac{\partial q_5}{\partial q_4} \frac{\partial q_4}{\partial q_1} + \frac{\partial q_5}{\partial q_3} \frac{\partial q_3}{\partial q_1} \frac{\partial q_3}{\partial q_2} = \frac{\partial q_5}{\partial q_4} \frac{\partial q_4}{\partial q_2} + \frac{\partial q_5}{\partial q_3} \frac{\partial q_3}{\partial q_2} \tag{3.13}
\]

Here \( q_5 \) represents the single output \( f \). The reverse mode runs a forward sweep to determine all of the intermediate values in the sequence. Then, starting with a single output variable, in this case \( q_5 \), the AD tool steps backward through the algorithm to compute the derivatives in reverse order. So from the example, and using the sequence from Equation (3.10),

\[
\frac{\partial q_5}{\partial q_5} = 1 \tag{3.14}
\]

\[
\frac{\partial q_5}{\partial q_4} = 1 \tag{3.15}
\]

\[
\frac{\partial q_5}{\partial q_3} = 1 \tag{3.16}
\]

\[
\frac{\partial q_5}{\partial q_2} = \frac{\partial q_5}{\partial q_3} \frac{\partial q_3}{\partial q_2} = (1)(q_1) \tag{3.17}
\]

\[
\frac{\partial q_5}{\partial q_1} = \frac{\partial q_5}{\partial q_4} \frac{\partial q_4}{\partial q_1} + \frac{\partial q_5}{\partial q_3} \frac{\partial q_3}{\partial q_1} \frac{\partial q_3}{\partial q_1} = (1)(q_2) + (1)(\cos(q_1)) \tag{3.18}
\]

giving the following final result,

\[
\frac{\partial q_5}{\partial q_1} = \frac{\partial f}{\partial x_1} = x_2 + \cos(x_1) \tag{3.19}
\]

\[
\frac{\partial q_5}{\partial q_2} = \frac{\partial f}{\partial x_2} = x_1 \tag{3.20}
\]
The advantage here is that only one reverse sweep is required to evaluate the derivatives with respect to both \(x_1\) and \(x_2\). Should there be a much greater number of inputs, such as is typical in an aerodynamic shape optimization problem, a single forward sweep to accumulate the code list as well as a single reverse mode sweep is all that would be necessary to calculate the sensitivities for a single output.

The disadvantage of the reverse mode is that the implementation is much more complicated than the forward mode. The reverse mode was used in the original development of the ADjoint method and so is used as a benchmarking tool in the development of a forward mode ADjoint method. In order to avoid the high computational costs associated with the forward mode of AD, a colouring method was used to accelerate the computation.

### 3.2.3 AD For Sensitivity Analysis

There are a number of automatic differentiation tools available for various programming languages. Tapenade was chosen for this work. The work here is done in Fortran 90 and Tapenade is an AD tool with support for this language. It is a non-commercial, source transformation tool developed by INRIA and is capable of performing differentiation in both forward and reverse mode\[^{37}\].

The ADjoint computes the total aerodynamic sensitivity as well as the partial derivatives from Equations (3.6) and Equation (3.7), \(\partial R/\partial w\), \(\partial R/\partial x\), \(\partial I/\partial w\) and \(\partial I/\partial x\) to form the total volume sensitivity. It is not necessary, however, to AD the entire ADjoint computation. As proposed by Mader\[^{6}\], AD is used only to form the partial derivative matrices. This is shown in Figure 3.1 which outlines the computation of \(\partial R/\partial w\) and where AD is used within this framework, shown by the greyed box. A similar setup is used for the other three partial derivative matrices.
3.3 Colouring Method

In order to understand the colouring method, consider again the total sensitivity given by Equations (3.6) and Equation (3.7). The partial derivatives, $dR/dw$, $dR/dx$, $dI/dw$ and $dI/dx$, in this equation are very sparse matrices. If the forward ADjoint were run using no special techniques, a perturbation would be placed on a single element of the grid. The sparse Jacobian matrix would then be evaluated, calculating the effect of that single perturbation. The perturbation would then be shifted to the next element on the grid and again the sparse matrix would be formed for this new perturbation. This method would require a residual evaluation for every grid point and every dependent variable in the field. However, upon examination of the sparsity of the matrix, it can be seen that the only non-zero terms in the matrix are in a specific stencil around the perturbation. Determining the stencil requires a knowledge of the physical model being solved, as well as the sensitivity calculation method being used. In the case of the Euler ADjoint method...
the stencil depends on the nearest and next-nearest neighbours of the perturbed cell. The stencil is shown in Figure 3.2. A significant improvement in computational efficiency can be made by taking advantage of this property as is shown by Nielsen and Kleb\cite{7} and Goldfarb and Toint\cite{28}. To do this, before applying any perturbation, the residual is preprocessed to establish element colourings. To understand this process consider first a one-dimensional array where only the nearest and next-nearest neighbours are affected by a perturbation.

This gives a 5-point stencil which can be packed onto the grid in such a way that no perturbed nodes lie within a stencil width of another. A colour then, consists of all of the perturbed nodes whose stencils do not overlap. In this fashion all of the nodes in this colour can be perturbed simultaneously and processed by the residual routine. In the simple case of the one-dimensional array it would require five colours in order to perturb every node. This is shown in Figure 3.3. In this example it would take 26 residual evaluations to complete the total sensitivity, and when using the colouring method it would require only 5 residual evaluations to complete the same calculation.

Figure 3.2: Stencil of affected cells in an Euler ADjoint calculation
Chapter 3. The ADjoint Method

Consider the formation of the Jacobian matrix

\[ A \equiv \left[ \frac{\partial R}{\partial Q} \right]^T \]

using complex variables, where the perturbation size in equation 4 on page 5 is taken to be the square root of the Fortran 95 intrinsic \( \text{tiny()} \) applied to a standard double precision real variable. After applying the complex perturbation \( i\Delta Q \) to an element of \( Q \) at grid point \( j \), the entry \( A_{jk} \) can be determined by performing a complex residual evaluation and mining the imaginary parts of the residual at node \( k \). In this manner, the rows of \( A \) can be constructed in a sequential fashion by successively perturbing the elements of \( Q \) at every grid point in the field. As noted above, this would require a complex residual evaluation for every grid point and every dependent variable in the field. However, note that upon applying a perturbation \( i\Delta Q \) and evaluating the complex-valued residual, the imaginary part of \( R \) will be largely zero. The only nonzero terms will lie within the stencil width of the residual operator. For the discretization used in the current work, these terms correspond to the nearest and next-nearest neighbors of the perturbed grid point. A significant speedup can be realized by taking advantage of this property.

Prior to applying any complex perturbations to the field, the grid is preprocessed to establish node colorings. The nodes in each color represent nodes that do not lie within a stencil width of another, and, therefore, may be simultaneously perturbed and processed by the complex residual routine. In this manner, a much larger number of elements in \( A \) may be computed during a single complex residual evaluation across the domain.

As the number of nodes increase, the number of colours remains the same.

The Euler stencil shown in Figure 3.2 is composed of 13 cells so it can be packed such that only 13 colours are needed to populate the residual derivatives matrix, \( dR/dw \). In practice, the three-dimensional stencil does not always fit perfectly with itself but may require a pattern which forces unnecessary colouring. A three-dimensional tessellation is required such that the packing pattern can be repeated in any direction. An optimal colouring pattern is one in which the tessellation resembles the stencil as much as possible and is necessary to minimize the excess iterations and simplify the implementation.

In order to demonstrate the three-dimensional packing the colour convention shown in Figure 3.4 is used to represent the three dimensional blocks. By using the colour convention in Figure 3.4 every cell in the stencil can be seen on the single plane displayed in Figure 3.5(a). The tessellation in two dimensions needs to encompass every cell in the three-dimensional stencil and as few extra cells as possible. In the case of the Euler tessellation no extra cells are needed. The tessellation which is repeated is highlighted in Figure 3.5(a). In order for the pattern to be fully three-dimensional each cell needs to be represented in a single line on the pattern shown in Figure 3.5(a). Starting with the centre cell of the stencil, the cells are numbered on a single line until the centre cell is repeated. Each number then represents a colour on the stencil. This number and pattern scheme is shown in Figure 3.5(a). The stencil can then be numbered according

![Figure 3.3: Perturbation colouring scheme for a one-dimensional grid, where each point indicates a perturbation.](image)

![Grid](image)

![Color 1](image)

![Color 2](image)

![Color 3](image)

![Color 4](image)

![Color 5](image)
to the colour it was assigned, shown in Figure 3.5(b). For the case of the Euler stencil, no excess colouring is required and only 13 colours are needed to populate the $dR/dw$ matrix. Every cell is then assigned a colour according to this stencil. This is done using the modulus function $\mod(m, n)$, which gives the remainder when $m$ is divided by $n$. Here, $m$ is the function determined by the numbered stencil and $n$ is the total number of colours required to populate the matrix. The remainder is the colour for the specified cell. The pattern needs to be fully three-dimensional, representing each cell on a single line of the pattern so that this simple implementation can be used. For the Euler stencil, the equation is,

$$\text{colour}(i, j, k) = \mod(i + 14j + 4k, 13).$$

(3.21)

For this particular stencil, no excess colouring occurs.

Now that the AD and colouring tools are understood, the ADjoint can be discussed in more detail.
3.4 The ADjoint

Adjoint methods are popular for use in aerodynamic shape optimization since their cost is independent of the number of inputs. Very effective adjoint techniques have been developed using acceleration algorithms, such as the method presented by Nemec and Zingg\cite{17} who use a Newton–Krylov algorithm.

The ADjoint, developed by Mader\cite{6} is an efficient combination of automatic differentiation and an adjoint method, which reduced the time required to differentiate complex
systems of equations. The original ADjoint was developed using reverse mode AD in an attempt to further reduce computational cost by getting rid of the dependency on the design variables within the sensitivity calculations. The idea behind the ADjoint is fairly straightforward. Automatic differentiation is used to compute the four partial derivative terms from the total sensitivity equation, given by Equation (3.6) and Equation (3.7), \( \partial I/\partial x \), \( \partial I/\partial w \), \( \partial R/\partial x \) and \( \partial R/\partial w \). These are then used in an adjoint method to calculate the total sensitivities. In this manner, the efficiency of the adjoint method is achieved along with its overall independence of the design variables while taking advantage of the efficient, accurate and simple implementation of automatic differentiation.

This proved to be a successful method for aerodynamic shape optimization\(^{[26,38-40]}\), as well as aerostructural optimization\(^{[4]}\). The ADjoint was developed only for the Euler and dissipation fluxes using reverse mode AD to evaluate the partial derivatives. Because of the nature of the reverse mode AD, the extension of the ADjoint to include viscous fluxes and turbulence models became much more complicated. In order to reduce the implementation time, a method to use forward mode AD with a colouring technique is developed and tested against the original reverse mode ADjoint.

### 3.5 The ADjoint Implementation

The ADjoint was developed for the three dimensional CFD solver, SUmb. The SUmb code is selectively differentiated to produce the partial derivatives, \( \partial R/\partial w \), \( \partial R/\partial x \), \( \partial I/\partial w \) and \( \partial I/\partial x \) to compute the sensitivities using an adjoint method. The discrete adjoint equations are then solved using PETSc. PETSc’s built in solver, GMRES, is used to compute the adjoint solution by solving Equation (3.6) and Equation (3.7) for a specific function of interest.

The benefit of using forward mode AD is that the original SUmb code is used in its development. This means a much less demanding implementation and will in the future
require fewer modifications should Sumb be updated. The reverse mode ADjoint and the complex step methods are used as benchmarking tools. The following discussion outlines the differences between the forward and reverse mode setups.

### 3.5.1 CFD Equations

In the reverse mode ADjoint flow solver, the discrete total sensitivities are developed for the three-dimensional Euler equations. The forward mode ADjoint is set up using the same governing equations for a baseline comparison of the two methods. The governing equations are written as follows,

\[
\frac{\partial w}{\partial t} + \frac{\partial f_i}{\partial x_i} = 0, \quad (3.22)
\]

where \( x_i \) are the coordinates in the \( i^{th} \) direction. The states, \( w \), and the fluxes, \( f \), for each cell are

\[
w_i = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho u_z \\ \rho E \end{bmatrix}, \quad f_i = \begin{bmatrix} \rho u_i \\ \rho u_i u_x + p\delta_i^1 \\ \rho u_i u_y + p\delta_i^2 \\ \rho u_i u_z + p\delta_i^3 \\ \rho u_i H \end{bmatrix}. \quad (3.23)
\]

\( u_{x,y,z} \) are the velocities in the \( x, y \) and \( z \) directions respectively. A coordinate transformation to computational coordinates \((\varepsilon_1, \varepsilon_2, \varepsilon_3)\) is then applied defined by the metrics,

\[
K_{ij} = \left[ \frac{\partial X_i}{\partial \varepsilon_j} \right], \quad J = \det(K), \quad (3.24)
\]

\[
K_{ij}^{-1} = \left[ \frac{\partial X_i}{\partial \varepsilon_j} \right], \quad S = JK^{-1}, \quad (3.25)
\]

where \( S \) represents the areas of the face of each cell projected on to each of the physical coordinate directions. Once the metrics are applied the Euler equations become,

\[
\frac{\partial Jw}{\partial t} + \frac{\partial F_i}{\partial \sigma_i} = 0, \quad (3.26)
\]
where the fluxes in the computational cell faces are given by $F_i = S_{ij} f_j$. In semi-discrete form the Euler equations are,

$$\frac{dw_{ijk}}{dt} + R_{ijk}(w) = 0, \quad (3.27)$$

where $R$ is the residual with all of its components (fluxes, boundary conditions, artificial dissipation, etc.).

The resulting set of five coupled ordinary differential equations, Equation (3.27), are marched in time using a hybrid five-stage Runge–Kutta, or a Newton–Krylov solver.

### 3.5.2 $\partial R/\partial w$ and $\partial R/\partial x$

The computation of the partial derivatives of the residual with respect to the states, $\partial R/\partial w$, is one of the most expensive and time consuming portions of the adjoint computation. This is why it is crucial to reduce computation time as well as memory usage as much as possible in these calculations.

The original SUmb solver computes the residual in a set of nested loops. The fluxes for the entire grid are calculated for each direction and only then is the final value of the residual in each cell computed. The subroutine structure of the original solver is shown in Figure 3.6.

In the original subroutine structure the states and pressures are calculated implicitly. The boundary conditions are reapplied for each perturbed value and the entire partial derivative matrix is recalculated. The highlighted section is used in the ADjoint using the converged states and explicitly calculating the pressure instead of computing them implicitly.

The reverse mode ADjoint took advantage of the sparsity of the $\partial R/\partial w$ matrix by using the same stencil as is used for the colouring, shown in Figure 3.2. Instead of using the stencil to simultaneously calculate the residuals, the reverse mode mimicked the order of the original routines highlighted in Figure 3.6, but did not loop over all of the cells.
in the domain. Instead, a single cell residual was calculated by only looping over the stencil of cells. This required a new code to be written that eliminated the loops over all the cells in the domain and included the boundary condition calls for every cell in the stencil. Reverse mode AD used Tapenade to differentiate this reduced stencil routine. An external set of loops of the entire domain was then used to fully populate the $\partial R/\partial w$ matrix. Because each routine needed to be re-written as a reduced stencil routine, the implementation of new routines was prohibitive.

The computation of $\partial R/\partial w$ in the forward mode ADjoint is set up as a stand-alone routine shown in Figure 3.1. The matrix is first initialized by setting up the colouring. This is done using a routine that systematically runs through the entire matrix marking the cells according to the stencil shown in Figure 3.2.

Once the colours have been assigned an outer loop is used whereby each colour is individually perturbed. Once a colour is selected, a seed is set on every cell in the
colour by setting the derivatives of their state variables, $\Delta w_i$, to 1. The fluxes are then calculated by mimicking the computations done in the original SUmb solver, highlighted in Figure 3.6. The routines being used have been slightly modified to allow for parallel processing of multi-block grids. It is these block-based routines that are differentiated using forward mode AD with respect to the states. Because of the colouring every residual in the selected colour is calculated simultaneously at the end of the nested loop evaluations. Figure 3.7 shows an outline of the subroutines used in this implementation of the block-based residual routines. The use of the original solver routines is where the forward mode ADjoint differs significantly from the reverse mode ADjoint.

In both cases, once computed, the elements of $\partial R/\partial w$ are stored in a sparse data structure for later use in the computation of the adjoint vector, $\psi$.

The partial residual with respect to the spatial variables, $\partial R/\partial x$, is calculated in much the same way. The seeds, however, are set on nodes instead of cells. This causes the calculations to be shifted. The colouring stencil is also different. The stencil is the same three-dimensional cross structure as the Euler stencil in Figure 3.2 but each beam is $2 \times 2 \times 4$ cells. The $\partial R/\partial x$ colouring uses the stencil shown in Figure 3.8 which also
show the colour convention used for the colour pattern.

![Figure 3.8: \( \partial R/\partial x \) colour convention and stencil.]

The actual pattern used for the colouring is shown in Figure 3.9(a) which also highlights the two-dimensional tessellation. Figure 3.9(b) shows the numbering used to colour the matrix.

The \( \text{mod} \) equation used is,

\[
\text{colour}(i, j, k) = \text{mod}(i + 7j + 27k, 38)
\]  

For this particular stencil, six excess colours occur.

The same block-based routines are used for the forward differentiation. Tapenade is again used to differentiate the routines, now with respect to the spatial inputs and aerodynamic design variables such as \( M_\infty, \alpha, \beta \) and twist. In this way \( \partial R/\partial x \) is populated in a sparse data structure.

In order to verify the sensitivity calculations, a complex-step routine is set up that follows the original SUmb solver. This is checked against the reverse mode ADjoint for accuracy. This complex routine can then be used as a step by step check for the new forward ADjoint routines. Because the forward mode ADjoint uses the original routines the benchmarking is much simpler than the benchmarking done for the reverse mode.
ADjoint. The sequential calculation of $\partial R/\partial w$ and $\partial R/\partial x$ allows for a more direct verification and a very structured verification process.

3.5.3 $\partial I/\partial w$ and $\partial I/\partial x$

The right hand side of Equation (3.6), $\partial I/\partial w$, and the first term in Equation (3.7), $\partial I/\partial x$, are the sensitivities of the function of interest with respect to flow variables, design variables and states. For the specific case where $I$ is dependent on the forces and moments, such as for lift, drag, $C_D$, $C_L$ or moments, the forces and moments must
be calculated using the original SUMb routine, \texttt{forcesAndMoments.f90}. In this case the original forces and moments calculation need a slight modification from the original routine in order to account for the colouring stencil. The colouring stencil here is a $2 \times 2 \times 2$ dense cube for $\partial I / \partial w$ and a $3 \times 3 \times 3$ dense cube for $\partial I / \partial x$. In the original routines the forces and moments are summed within the lowest level routine. This method worked because only a single cell was perturbed in each iteration. Because the colouring method perturbs multiple cells the forces and moments must be summed according to their stencil in order to keep the resulting derivatives separate. To accomplish this, a simple loop was added to the original routine. This loop takes the forces and moments calculated for each cell and spreads the value out evenly to the four surrounding nodes (or vice versa if a node is perturbed as in the calculation of $\partial I / \partial x$). Because only the first and second layer of cells can be affected by a force, these calculations are reduced to the first and second row of cells on the surfaces with applicable boundary conditions. Outside of the forces and moments calculation routine the forces in the stencil of affected cells or nodes are summed for each perturbed cell or node, again reduced only to the first and second layers along the boundary.

Then, the dot product of this six point vector (for the three forces and three moments) is taken with the cost function vector. The cost function vector is derived from the manipulations required to obtain the effects of the forces and moments on the function of interest. For example, when the function of interest is drag coefficient, $C_D$, the cost function vector would be,

$$ F_{cost} = \begin{bmatrix} w_{Dx} \\ w_{Dy} \\ w_{Dz} \\ 0 \\ 0 \\ 0 \end{bmatrix} $$

(3.29)
where $F_{\text{cost}}$ is the cost function vector and $w_D$ is the drag direction calculated by the rotation of alpha and beta around the corresponding axes. A single point for the partial sensitivity for $\partial I/\partial w$ becomes,

$$\frac{\partial I}{\partial w} = \partial F_{\text{force}} \cdot F_{\text{cost}} + F_{\text{force}} \cdot \partial F_{\text{cost}}$$

(3.30)

where $F_{\text{force}}$ is the vector of forces and moments for that cell. The populated matrix is saved in a sparse matrix format in PETSc for use in the total sensitivity calculation. Both the reverse mode ADjoint as well as the complex step method are used to verify the forward mode ADjoint with colouring of these partial sensitivity calculations using a simple test case as will be described in Chapter 5.

### 3.5.4 Total Sensitivity

In order to solve the total sensitivity Equation (3.7), first the adjoint vector, $\psi$, needs to be calculated. The four partial derivatives, $\partial R/\partial w$, $\partial R/\partial x$, $\partial I/\partial w$, and $\partial I/\partial x$ are all sparse matrices or vectors. These are saved in PETSc’s sparse data structures.

In order to demonstrate the efficiency of the new forward mode ADjoint, a test case is set up looking at the sensitivities of the lift and drag coefficients, $C_L$ and $C_D$, with respect to the free stream Mach number, $M_\infty$. In the case where $I = C_D$ the adjoint equation becomes,

$$\left[\frac{\partial R}{\partial w}\right]^T \psi = -\frac{\partial C_D}{\partial w}$$

(3.31)

The total sensitivity can then be calculated by solving for the adjoint, $\psi$. For the specific case where $M_\infty$ is the design variable, the total aerodynamic sensitivity equation, Equation (3.7) becomes,

$$\frac{dC_D}{dM_\infty} = \frac{\partial C_D}{\partial M_\infty} + \psi^T \frac{\partial R}{\partial M_\infty}$$

(3.32)

The calculation of the first and last terms on the right hand side were discussed previously. The total sensitivity is then formed using the sparse matrices and in the case of
\( \partial C_D/\partial M_\infty \), it is a single value. This total sensitivity is used as a search direction in the optimization algorithm, which is described in Chapter 2.
Chapter 4

Viscous Implementation

Although the Euler equations can give an approximation of the properties within a flow field such as velocity and pressure, ignoring viscosity can lead to unrealistic results. The Euler equations neglect skin friction, and the boundary layer development, which are crucial when optimizing the planform area and shape. By adding viscous fluxes, viscous dissipation and laminar NS boundary conditions, a more accurate analysis can be performed.

4.1 The Original SUmb Solver Method

The original SUmb routine was setup to use the laminar NS equations as well as multiple turbulence models. The code includes the necessary laminar boundary conditions, no-slip condition, as well as the viscous flux calculations. The forward mode AD method allows for a more direct implementation of these routines into the ADjoint method. The laminar NS equations and viscous effects are shown here to demonstrate the process of extending the ADjoint code.
4.1.1 Laminar NS Physical Model

The schematic shown in Figure 4.1 shows the call graph of the forward mode ADjoint routines. The highlighted cells show the viscous routines implemented to extend the method to the laminar NS equations. As with the Euler forward mode ADjoint, there is a top level routine for each partial derivative, $\partial R/\partial w$, $\partial R/\partial x$, $\partial I/\partial w$ and $\partial I/\partial x$. Each calls a set of block-based routines such as those for $\partial R/\partial w$ shown in Figure 4.1. It is only these block-based routines that are differentiated using Tapenade.

First, the laminar viscosity calculations are included in the sensitivity calculations. This is done in the subroutine `computeLamViscosity`. Sutherland’s law is used to calculate the laminar viscosity as follows,

$$ rlv = \mu_{\text{Suth}} \left( \frac{T_{\text{Suth}} + S_{\text{Suth}}}{T + S_{\text{Suth}}} \right) \left( \frac{T}{T_{\text{Suth}}} \right)^\frac{3}{2} $$

(4.1)

where $rlv$ is the nondimensional laminar viscosity and the subscript $Suth$ denotes the nondimensionalized constants in Sutherland’s law for viscosity, $\mu_{\text{Suth}}$, temperature, $T_{\text{Suth}}$ and air, $S_{\text{Suth}}$. The temperature, $T$, is simply calculated using the perfect gas law,

$$ T = \frac{P}{R_{\text{gas}} \rho}, $$

(4.2)

where $P$ is the pressure at the cell being considered and $\rho$ is the density in the cell. Thus, the laminar viscosity affects the residuals with respect to the states, $\partial R/\partial w$ and
the partial derivatives of the objective function with respect to the states, $\partial I/\partial w$. The laminar viscosity is then used in the calculation of the viscous fluxes in `viscousFlux`.

The subroutine `viscousFlux` computes the heat transfer coefficient, $h$, by taking into account the porosity of the surface, $\phi$, the Prandtl number, $Pr$, and the specific heat ratio, $\gamma$,

$$ h = \frac{\phi(lvl_i + lvl_{i+1})}{\frac{1}{2}Pr(\gamma_i + \gamma_i + 1) - 1} $$

(4.3)

The heat transfer is then used in the calculation of the $\rho E$ component of the viscous fluxes. The $x$, $y$ and $z$ components are calculated from the stress tensor, $\tau$. To calculate $\tau$ first the velocity, $(u, v, w)$, is calculated at each node. The gradients at the faces are calculated by averaging the four nodal values. The stress tensor is then calculated as follows,

$$ \tau_{xx} = \phi(lvl_i + lvl_{i+1}) \left(2u_x - \frac{2}{3}(u_x + v_y + w_z)\right) $$

$$ \tau_{yy} = \phi(lvl_i + lvl_{i+1}) \left(2v_x - \frac{2}{3}(u_x + v_y + w_z)\right) $$

$$ \tau_{zz} = \phi(lvl_i + lvl_{i+1}) \left(2w_x - \frac{2}{3}(u_x + v_y + w_z)\right) $$

$$ \tau_{xy} = \phi(lvl_i + lvl_{i+1})(u_y + v_x) $$

$$ \tau_{xz} = \phi(lvl_i + lvl_{i+1})(u_z + w_x) $$

$$ \tau_{yz} = \phi(lvl_i + lvl_{i+1})(v_z + w_y) $$

(4.4)

The stress tensor is used to compute the remaining viscous fluxes. The skin friction velocity, $u_\tau$, is also calculated using the laminar viscosity. This is done in the subroutine `computeSkinFrictionVelocity`. For each viscous subface the velocity difference between the internal cell and the wall is calculated. The normal velocity of the internal cell and the magnitude of the tangential velocity are calculated. These are used to compute the Reynold’s number. Finally $u_\tau$ can be calculated, and it is used in the calculation of the porosity $\phi$.

The boundary conditions are also updated to include the original viscous boundary conditions, `NSWallAdiabatic` and `NSWallIsothermal`. 
Adjustments are also made to the free stream pressure and density. These are adjusted according to the grid velocity, the Reynolds number per unit length, the viscosity using Sutherland’s law and the free stream velocity relative to the body. The adjusted pressure is then used as a pressure switch that adjusts the entropy used to calculate the dissipation fluxes.

For the case where the objective function $I = C_D$, the partial calculations of $\partial I / \partial x$ and $\partial I / \partial w$ include the calculation of the viscous forces. These are calculated using the stress tensor and the laminar viscosity and are added to the inviscid force and moment calculations for all of the viscous subfaces.

These viscous routines are included in the automatic differentiations.

### 4.2 Viscous Colouring

When a single cell is perturbed with the viscous routines, a different stencil of cells is affected than for the Euler case. The stencil for the viscous flux calculations is a dense $3 \times 3 \times 3$ stencil. The stencil, however, must also include the original Euler stencil as the inviscid fluxes must be calculated using this original stencil. The inviscid fluxes are summed to the viscous fluxes. The resulting stencil is shown in Figure 4.2. The colouring is applied in the same manner as the Euler stencil but since it is more dense it requires more colours to pack the stencil into the grid. The colour convention shown in Figure 4.3 is used to determine the two-dimensional tesselation. The same numbering process is achieved by a more complex tesselation, shown in Figure 4.4(a). In this case black squares highlight the excess colours required. When the colour representing a black square is calculated, this provides extraneous data and although it does not diminish the accuracy of the resulting sensitivity calculation, it does reduce the efficiency. This packing, highlighted by the tessellation shown in Figure 4.4(a) allows for a minimum of 35 colours as opposed to the 33 in the stencil in Figure 4.2. This causes an increase in the
computational cost as was expected; however, because only 2 extra colours were needed, this is a very minor increase compared to the cost of packing the stencils side by side, which would require 125 colours. This computational cost decrease will be shown with a test case in Chapter 5.

\section*{4.3 Complex-Step Validation Method}

The validation of the viscous routines in the forward mode ADjoint method rely on the complex-step method. The laminar NS equations had not yet been implemented in the reverse mode ADjoint method and so could not be used for validation as in the Euler case. The reverse mode ADjoint method is, however, used as a benchmarking tool for the verification tools written in the complex-step method. Two main steps are taken to validate the laminar NS equation implementation. First, the partial derivatives, $\partial R/\partial w$, $\partial R/\partial x$, $\partial I/\partial w$ and $\partial I/\partial x$ are compared individually to the original SUmb code using the complex-step method. The total sensitivities are then compared to the original complex SUmb solver to check the validity of the adjoint calculation and total sensitivity formulation.
4.3.1 Individual Validation Routines

The complex-step verification routines are written as simple call graphs following the original SUmb solver routine. Four routines are setup, one for each of $\partial R/\partial w$, $\partial R/\partial x$, $\partial I/\partial w$ and $\partial I/\partial x$. Each routine is setup according to the call graph shown in Figure 4.5. The perturbations set on the $w$ or $x$ are a purely imaginary $10^{-40}$ step. The verification routines are checked against the solver by solving for single cell perturbations. One difference to be noted is the calculation of pressure. The original SUmb solver calculates pressure implicitly and so the ADjoint computePressuresAdj routine is used instead in the verification routines. This is done to more closely mimic the new forward mode ADjoint routines. This helps to ensure that all of the necessary viscous routines from the original SUmb solver are included in the forward mode block-based routines. Once the verification routines are implemented, a step-by-step procedure is used with the forward ADjoint routines to more efficiently locate the source of any problems. With the differentiation verified, the calculations of the partial derivatives need to be addressed.

The original solver calculates the total sensitivity and not the partial sensitivities as the ADjoint does. In order to take advantage of the commonalities between the two, a method is devised to compare them directly. The complex components of the states and figures.
pressures at the beginning of the final iteration in the complex solver convergence are used at the beginning of the forward ADjoint routines as the derivative components. In doing so the resulting residuals can be compared directly and any missing routines can be identified by where the residuals differ.

A secondary method is also used to check the partial state derivative used to form the adjoint, $\partial I/\partial w$. This is done using a direct solution. Consider again Equation (3.4) shown here,

$$\frac{dI}{dx} = \frac{\partial I}{\partial x} - \frac{\partial I}{\partial w} \left[ \frac{\partial R}{\partial w} \right]^{-1} \frac{\partial R}{\partial x}$$

The third and fourth terms on the right hand side of this equation form the total deriva-
tive of the states with respect to the design variables, $dw/dx$. If the first term of the right hand side is forced to zero by removing the perturbation on the design variable, then a simple vector cross product of $dw/dx$ with the partial $∂I/∂w$ should give the total sensitivity output $dI/dx$ from the solver.

### 4.3.2 Total Sensitivity Verification

The total sensitivity can be checked once the four partial derivatives are verified against the complex solver. Since each of the partials has been checked independently, it can be assumed that any eventual discrepancy between the forward mode ADjoint total sensitivity and the complex method total sensitivity would be due to the adjoint variable calculations or in the convergence of the total sensitivity. There is also the possibility of a discrepancy due to the stiffness of the calculations. The total sensitivity of the forward mode ADjoint method are matched to an accuracy of $O(10^{-5})$. These verifications are done using the same simple test case as is used for the Euler verifications with some slight modifications as will be discussed in Chapter 5.
Loop over all cells

- call setPointers
- x(i,j,k) = xref(i,j,k)
- x(i,j,k) = xref + deltax
- call xHalo
- call metric
- call computePressureAdj

  if viscous
  - call computeLamViscosity
  - call applyAllBC
  - call computeUtau

  if viscous
  - call viscousFlux
  - $\frac{dR}{dw}, \frac{dR}{dx}$ $\frac{dI}{dw}, \frac{dI}{dx}$
  - call residual
  - call forcesAndMoments

  write residual to file

  x(i,j,k) = xref

  repeat call routine to reset residual

Figure 4.5: Verification routine call graph.
Chapter 5

Benchmarking and Verification

In order to verify both the accuracy and the computational efficiency of the forward mode ADjoint with colouring, a single block test case of channel flow over a bump is run with the SUmb flow solver. The complex-step method is used in order to verify the accuracy of the new method since the routines follow the original SUmb solver routines nearly exactly. This is done using a complex-step verification routine and individually testing the partial derivative components of the sensitivity equation. Once these match, the total sensitivity is compared to ensure that the adjoint calculations are being computed correctly as well. The efficiency of the new method is tested by comparing the timing of the total sensitivity calculation against the reverse mode ADjoint method as well as comparing the timing results using various colouring applications.

5.1 Test Case

The computational mesh for the test case used is shown in Figure 5.1. The front and back walls of the channel are set with symmetry boundary conditions. The inflow and outflow faces have non-reflection boundary conditions imposed on them. The upper wall is flat and set to far-field conditions. The bottom wall is deformed with a sinusoidal bump to create a reasonable variation in the flow. Both the top and bottom faces use a
linear pressure extrapolation boundary condition. The free stream Mach number used is 0.8395. The mesh is a 3072 cell grid, \((32 \times 24 \times 4)\). The grid is denser near the bottom surface in order to mimic a viscous grid as this bump case is used for both the Euler and the viscous cases, however a correct physical solution is not expected for a grid this coarse, the objective is to verify the numerical values of the derivatives.

The flow solution is converged to an \(l^2\) tolerance of \(O(10^{-12})\) using a single grid. A steady state equation mode is used for both the Euler and laminar NS cases. The adjoint is also required to converge to an \(l^2\) tolerance of \(O(10^{-12})\). The design variable used for the total aerodynamic derivative is \(M_\infty\), and the two objective functions are \(C_D\) and \(C_L\).
5.2 Forward Mode Euler ADjoint With Colouring

The test case shown in Figure 5.1 was used with an Euler wall boundary condition set on the bump surface.

The surface density distribution is shown in Figure 5.2.

![Density Distribution](image)

**Figure 5.2:** Euler case surface density distribution.

The reverse mode ADjoint was setup for the Euler equations and so can be used as a benchmarking tool against the forward mode Euler ADjoint. The reverse mode method is used to show both the accuracy and efficiency of the new method. The accuracy is shown by comparing the total sensitivities computed by the new method with those from the reverse mode method. The complex-step method is included in this comparison as it is numerically exact\(^{[22]}\), meaning that the precision of the total sensitivity is the same order as the precision of the solution.
Table 5.1: Accuracy validations

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>Objective</th>
<th>Forward</th>
<th>Reverse</th>
<th>Complex Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>$C_D$</td>
<td>-0.34251303215</td>
<td>-0.34251300655</td>
<td>-0.34251303206</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>$C_L$</td>
<td>0.63455230531</td>
<td>0.63455255165</td>
<td>0.63455230589</td>
</tr>
<tr>
<td>(13, 1, 5)</td>
<td>$C_D$</td>
<td>0.10075206621</td>
<td>0.10075206217</td>
<td>0.10075206605</td>
</tr>
<tr>
<td>(13, 1, 5)</td>
<td>$C_L$</td>
<td>-0.03564905233</td>
<td>-0.03564838830</td>
<td>-0.03564905251</td>
</tr>
<tr>
<td>(6, 1, 3)</td>
<td>$C_D$</td>
<td>0.00956356749</td>
<td>0.00956356748</td>
<td>0.00956356748</td>
</tr>
<tr>
<td>(6, 1, 3)</td>
<td>$C_L$</td>
<td>0.04736834080</td>
<td>0.04736834066</td>
<td>0.04736834079</td>
</tr>
<tr>
<td>$M_\infty$</td>
<td>$C_D$</td>
<td>1.2431009592167</td>
<td>1.2431009592167</td>
<td>1.2431009592159</td>
</tr>
<tr>
<td>$M_\infty$</td>
<td>$C_L$</td>
<td>0.03031768319328</td>
<td>0.03031768319328</td>
<td>0.0303176831919</td>
</tr>
</tbody>
</table>

The precision specified in this test case is $O(10^{-12})$. To test the accuracy of the forward mode ADjoint, results for $C_D$ and $C_L$ are compared to the reverse mode ADjoint method as well as the complex step method in Table 5.1. The total aerodynamic derivatives, $dC_L/dM_\infty$ and $dC_D/dM_\infty$ are compared, as well as the total surface derivatives. Three points on the surface were chosen as design variables to show the accuracy on the inflow boundary, the symmetry boundary and on the Euler wall boundary. Because of the symmetry of the test case these three points are representative of the four boundary-surface intersections and the surface boundary itself. These show the accuracy of the pySUmb components of $dC_L/dx$ and $dC_D/dx$. This also makes use of the pyWarp component, $dx_v/dx_s$. An algebraic mesh warping is used.

The complex step derivative was given by\cite{22},

$$
\frac{dC_D}{dM_\infty} = \frac{Im[C_D(M_\infty + ih)]}{h}
$$

where $h$ is the complex step and has a value of $h = 10^{-40}$ for this test case.

The results in Table 5.1 show the accuracy of the new method since the total sensitivities are all accurate to within between 9–12 digits, more accurate in some cases than the
reverse mode. The total aerodynamic sensitivities show the reverse mode and the forward mode to match each other better than either matches the complex-step method. Because the accuracy here is at machine precision the use of an implicit pressure calculation for the complex-step versus the explicit pressure calculation in the AD routines would cause very small discrepancies between the methods. All the derivatives with respect to the design variables $x$ were obtained in two ADjoint solutions, once for each objective.

For a more general view of the accuracy over the entire domain, the flux Jacobian computed using the forward mode method was compared to that computed with the complex-step method. The relative error between the two is shown in Figure 5.3. The quantity shown is an $l^2$ norm of the difference between the derivative given by the forward mode and the one given by the complex-step method. Figure 5.3 shows the $l^2$ norm of the error for the density derivatives with respect to all five states. The velocity comparative errors can be found in Appendix A. The majority of the errors are between $O(10^{-14})$ and $O(10^{-9})$, demonstrating the accuracy of the new method.

In order to check the computational efficiency of the new method, the time required for the computation of the total sensitivity as well as all of its component parts was compared to those of the reverse mode ADjoint method. The partial derivatives for the reverse mode method cannot be timed individually. The total time can be broken down into the time required for the RHS and the time required for the flux Jacobian, so these values are compared as well as the time required to solve the adjoint and the time to form the total sensitivity. These are presented in Table 5.2, which indicates that the forward mode ADjoint method does in fact take half the time compared to the reverse mode ADjoint method. The flux Jacobian assembly in the reverse mode method comprises 86% of the ADjoint computational time. In the forward mode ADjoint this time is reduced to less than half of the time required in the reverse mode. This reduction in computational time alone demonstrates the benefits of the forward mode colouring method. The total cost of the forward mode ADjoint solver, including the computation
of all of the partial derivatives and the solution of the adjoint system is only 57% of the total cost including the cost of the flow solution compared to 72% for the reverse mode method. Only 42% of the forward mode computation cost is spent on the flux Jacobian calculations, compared to 62% in the reverse mode.

The benefits of the forward mode ADjoint method become more prominent with a viscous implementation. The use of the forward mode AD allowed for the original SUmb routines to be used and differentiated. Because SUmb is already set up for laminar NS and multiple turbulence models, the implementation of these routines is relatively straight forward.
### Table 5.2: Timing benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Forward (s)</th>
<th>Reverse (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow Solution</td>
<td>4.056</td>
<td>4.133</td>
</tr>
<tr>
<td>ADjoint</td>
<td>5.339</td>
<td>10.384</td>
</tr>
<tr>
<td>Breakdown:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\partial R/\partial w$</td>
<td>0.711</td>
<td></td>
</tr>
<tr>
<td>$\partial R/\partial x$</td>
<td>3.197</td>
<td></td>
</tr>
<tr>
<td>Total Flux Jacobian</td>
<td>3.940</td>
<td>8.93</td>
</tr>
<tr>
<td>Compute RHS</td>
<td>0.019</td>
<td>0.024</td>
</tr>
<tr>
<td>Adjoint Equation Solution</td>
<td>1.38</td>
<td>1.43</td>
</tr>
</tbody>
</table>

#### 5.3 Forward Mode Viscous ADjoint with Colouring

The single block test case of channel flow over a bump shown in Figure 5.1 was used again for benchmarking and verifying the viscous adjoint. The symmetry plane boundary conditions as well as the far-field boundary condition on the top surface were kept the same. Similarly, the inflow and outflow boundary conditions remained unchanged. The bump wall boundary condition was changed to a laminar viscous surface to include the adiabatic Navier–Stokes wall boundary condition.

The density contour plot of the viscous test case is shown in Figure 5.4. Again using the accuracy of the complex-step method the total sensitivity of the forward mode ADjoint method is checked against the original SUmb solver using an imaginary step size of $10^{-40}$. The flow solutions are all converged to $\mathcal{O}(10^{-12})$, with $C_D$ and $C_L$ matching to $\mathcal{O}(10^{-12})$. Again, the two aerodynamic sensitivities, $dC_D/dM_\infty$ and $dC_L/dM_\infty$ and the same three points of the surface derivative as for the Euler case are compared. The resulting total sensitivities are shown in Table 5.3. In the case of the viscous forward mode ADjoint, the total sensitivities did not match as well as the Euler case. Because each of
the partial derivatives, $\partial R/\partial w$, $\partial R/\partial x$, $\partial I/\partial w$ and $\partial I/\partial x$, were matched to the original SUmb solver using the complex-step method to $O(10^{-9})$, the $O(10^{-4})$ discrepancy must be in the stiffness of the calculations or in the adjoint or total sensitivity computations using PETSc. Both the total aerodynamic derivatives and the total surface derivatives matched to 6 significant figures.

The flux Jacobian computed using the forward mode method was compared to that computed with the complex-step method, as with the Euler case. The relative error between the two is shown in Figure 5.5, which shows the $l^2$ norm for the difference between the density derivatives with respect to all five states given by the forward mode and the ones given by the complex-step method. The velocity comparative errors can be found in Appendix A. The quantities shown were computed in the same manner as those for Figure 5.3. The errors are still within an acceptable range mostly varying between
Chapter 5. Benchmarking and Verification

Table 5.3: Accuracy validations

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>Objective</th>
<th>Forward</th>
<th>Complex Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>$C_D$</td>
<td>-0.4256082</td>
<td>-0.4256052</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>$C_L$</td>
<td>-0.8254237</td>
<td>-0.8254232</td>
</tr>
<tr>
<td>(13, 1, 5)</td>
<td>$C_D$</td>
<td>0.1325634</td>
<td>0.1325619</td>
</tr>
<tr>
<td>(13, 1, 5)</td>
<td>$C_L$</td>
<td>0.3492992</td>
<td>0.3492958</td>
</tr>
<tr>
<td>(6, 1, 3)</td>
<td>$C_D$</td>
<td>0.0064826</td>
<td>0.0064877</td>
</tr>
<tr>
<td>(6, 1, 3)</td>
<td>$C_L$</td>
<td>0.0677012</td>
<td>0.0677025</td>
</tr>
<tr>
<td>$M_\infty$</td>
<td>$C_D$</td>
<td>1.4131662</td>
<td>1.4131678</td>
</tr>
<tr>
<td>$M_\infty$</td>
<td>$C_L$</td>
<td>6.1079985</td>
<td>6.1079921</td>
</tr>
</tbody>
</table>

$O(10^{-8})$ to $O(10^{-5})$.

Unlike the Euler case, there is no reverse mode ADjoint viscous case for comparison. No time validations could be done comparing the forward mode to the reverse mode techniques. However, the efficiency of a good colouring implementation is demonstrated by comparing the timing results of each partial derivative as well as the time required to solve the adjoint and form the total sensitivity. These computations were done for a method with no colouring, one using a dense $5 \times 5 \times 5$ stencil, and the third using the colouring laid out in Figure 4.2. These results are shown in Table 5.4.

Table 5.4 shows the efficiency of using colouring to accelerate the computation of the partial derivatives. The cost of the forward mode ADjoint solver with no colouring is almost 9 times the cost of a flow solution compared to only 112% when using the $5 \times 5 \times 5$ stencil and a further reduction to 81% with the viscous stencil colouring. The flux Jacobian is clearly the most costly portion of the computation showing the importance of reducing this as much as possible. The flux Jacobian computation using no colouring comprises 90% of the total computational cost down to 46% when using the coloring. The flux Jacobian takes a tenth of the time when using the necessary colouring.
technique. When no colouring is used the viscous computation is about 220 times the cost of the Euler computation. In comparison, it is only about 20 times the cost when the viscous colouring is used, a drastic reduction.

5.4 Discussion

The test case presented showed that the forward mode ADjoint approach was able to maintain the accuracy of the reverse mode ADjoint method for the Euler equations while significantly increasing the efficiency of the method. The viscous case demonstrated an acceptable accuracy when compared to the complex-step method and showed the decrease in computational cost associated with using a colouring method. There are some significant points, however, which merit discussion.
Table 5.4: Timing benchmarks

<table>
<thead>
<tr>
<th></th>
<th>No Colouring (s)</th>
<th>$5 \times 5 \times 5$ (s)</th>
<th>ViscousStencil (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow Solution</td>
<td>128.37</td>
<td>126.11</td>
<td>126.35</td>
</tr>
<tr>
<td>ADjoint</td>
<td><strong>1 117.865</strong></td>
<td><strong>141.512</strong></td>
<td><strong>101.812</strong></td>
</tr>
<tr>
<td>Breakdown:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\partial R/\partial w$</td>
<td>779.82</td>
<td>128.889</td>
<td>91.079</td>
</tr>
<tr>
<td>$\partial R/\partial x$</td>
<td>329.935</td>
<td>8.948</td>
<td>7.261</td>
</tr>
<tr>
<td>Total Flux Jacobian</td>
<td>1 109.755</td>
<td>137.837</td>
<td>98.34</td>
</tr>
<tr>
<td>Compute RHS</td>
<td>0.040</td>
<td>0.043</td>
<td>0.032</td>
</tr>
<tr>
<td>Adjoint Equation Solution</td>
<td>4.27</td>
<td>3.60</td>
<td>3.44</td>
</tr>
</tbody>
</table>

The loss in accuracy in the viscous case is significant compared to that in the Euler case. Because the differentiated routines were taken from the original SUmb solver, this significantly reduced the possibility of error within the routines themselves. The order of the routines used also came from the original solver. The error then most likely stems from the method in which the equations were applied to the original solver causing an unexpected loss in accuracy when the adjoint equations were solved. The goal of the test case was simply to show the relative ease of incorporating a new set of governing equations into the ADjoint as well as to demonstrate the capabilities of the colouring method. This was achieved despite the loss of accuracy. The implementation process for the forward mode method was more direct than the reverse mode method requiring fewer alterations to the original code. This new method will be easier to maintain should the original code be updated and requires only a basic understanding of AD to implement.

Computational cost is a significant cause for concern when dealing with the discrete adjoint equations. Often sacrifices need to be made to accuracy in order to reduce this cost as argued by Rumpfkeil and Zingg\cite{41}, who used large time steps and a coarse grid, increasing performance with an acceptable loss of accuracy. Computational efficiency
here is gained without a considerable loss of accuracy. The colouring method allowed the simpler forward mode AD to be used without increasing computational cost. The Euler test case clearly showed that the new method with colouring was more efficient than the reverse mode method. The viscous test case showed the extent to which this was made possible. Without the benefits of the colouring method forward mode AD would not have been a feasible option.

The implementation discussed here applies to both the total aerodynamic derivative and the total surface derivative. The only difference is the $\partial R/\partial x$ matrix versus the $\partial R/\partial \alpha$ vector and $\partial I/\partial x$ versus $\partial I/\partial \alpha$. The real benefits of the method arise when there are large numbers of design variables, as is the case for most aerodynamic shape optimizations. This takes the derivative of the objective function, in this case $C_D$ and $C_L$, with respect to all of the shape variables with a single ADjoint solution, as was shown for the total surface derivatives. The advantage is that the derivative of any point on the surface is then known for an objective function with only one computation of the time-consuming flux Jacobian.
Chapter 6

Conclusions and Future Work

The previous chapters presented a new approach to the ADjoint method proposed by Mader et al.\textsuperscript{[1]} using a forward mode AD approach with a colouring acceleration technique. First, the forward mode Euler ADjoint was tested against the reverse mode method. The colouring technique showed an improvement in computational efficiency while maintaining the same accuracy as the original method. The colouring technique proved to make forward mode AD a feasible option despite the dependence on the number of inputs. The flux Jacobian, which is the most time-consuming process in the sensitivity analysis, was three times faster than the reverse mode ADjoint method. The errors in the total sensitivities when compared to reverse mode and a complex-step method ranged from $O(10^{-12})$ to $O(10^{-9})$ with the flux Jacobian calculation comprising only 6% of the total computational time.

The second verification conducted was on the laminar NS equation implementation into the new forward mode ADjoint. The forward mode AD allowed for a reduced implementation time and a code which makes it easier to maintain. The use of the original SUmb routines reduced the chance of error in implementation since no code had to be rewritten. By comparing the test result timing using a viscous colouring to one with no colouring it was obvious how much the technique reduced the computational cost.
of the method, it was about ten times more efficient. There was some loss in accuracy in the viscous test case when compared to a complex-step method, showing an accuracy of $\mathcal{O}(10^{-5})$ compared to the complex accuracy of $\mathcal{O}(10^{-12})$. Further work could be done looking into the solution methods within PETSc. Although the solution methods proved efficient and accurate for the Euler sensitivity analysis, they may not be suitable for the laminar NS case. The accuracy is still within the range of a finite-difference method and so is acceptable in showing the benefits in reduced implementation time. The forward mode ADjoint was made feasible by the use of the colouring acceleration technique thus providing the reduced implementation time while maintaining a reasonable accuracy.
References


Appendix A

Sensitivity Analysis

The following graphics show the $l^2$ norm of the difference between the forward mode Euler ADjoint method flux Jacobian and the complex-step flux Jacobian for the velocity states.

![Figure A.1: $l^2$ norm of the velocity state in the x-direction of the flux Jacobians.](image)

The following graphics show the $l^2$ norm of the difference between the forward mode viscous ADjoint method flux Jacobian and the complex-step flux Jacobian for the velocity
Figure A.2: $l^2$ norm of the velocity state in the y-direction of the flux Jacobians.
Appendix A. Sensitivity Analysis

Figure A.3: $l^2$ norm of the velocity state in the z-direction of the flux Jacobians.

Figure A.4: $l^2$ norm of the velocity state in the x-direction of the flux Jacobians.
Appendix A. Sensitivity Analysis

Figure A.5: \( l^2 \) norm of the velocity state in the y-direction of the flux Jacobians.

Figure A.6: \( l^2 \) norm of the velocity state in the z-direction of the flux Jacobians.