Viscosity stabilized adjoint method for unsteady compressible Navier-Stokes equations

by

Chaitanya Anil Talnikar
B.Tech., Indian Institute of Technology Bombay (2013)
S.M., Massachusetts Institute of Technology (2015)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computational Science and Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2018

Author . . . .
Department of Aeronautics and Astronautics
August 1, 2018

Certified by . . . .
Qiqi Wang
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by . . . .
David Darmofal
Professor of Aeronautics and Astronautics
Thesis Committee Member

Certified by . . . .
Semyon Dyatlov
Assistant Professor of Mathematics
Thesis Committee Member

Accepted by . . . .
Nicolas Hadjiconstantinou
Professor of Mechanical Engineering
Co-Director, Computational Science and Engineering

Accepted by . . . .
Hamsa Balakrishnan
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Viscosity stabilized adjoint method for unsteady compressible Navier-Stokes equations

by

Chaitanya Anil Talnikar

Submitted to the Department of Aeronautics and Astronautics
on August 1, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computational Science and Engineering

Abstract

Design optimization methods are a popular tool in computational fluid dynamics for designing components or finalizing the flow parameters of a system. The adjoint method accelerates the design process by providing gradients of the design objective with respect to the system parameters. But, typically, adjoint-based design optimization methods have used low fidelity simulations like Reynolds Averaged Navier-Stokes (RANS). To reliably capture the complex flow phenomena like turbulent boundary layers, turbulent wakes and fluid separation involved in high Reynolds number flows, high fidelity simulations like large eddy simulation (LES) are required. Unfortunately, due to the chaotic dynamics of turbulence, the adjoint method for LES diverges and produces incorrect gradients. In this thesis, the adjoint method for unsteady flow equations is modified by adding artificial viscosity to the adjoint equations. The additional viscosity stabilizes the adjoint solution and maintains reasonable accuracy of the gradients obtained from it. The accuracy of the method is assessed on multiple turbulent flow problems, including subsonic flow over a cylinder and transonic flow over a gas turbine vane. The utility of the method is then tested in performing shape optimization of the trailing edge of a transonic turbine vane. The optimal design, found using a modified gradient-based Bayesian optimization algorithm, shows approximately 15% better aero-thermal performance than the baseline design.

Such design optimizations are possible due to the availability of massively parallel supercomputers. Designing high performance fluid flow solvers for the next generation supercomputers is a challenging task. In this thesis, a two-level computational graph method for writing optimized distributed flow solvers on heterogeneous architectures is presented. A checkpoint-based automatic differentiation method is used to derive the corresponding adjoint flow solver in this framework.

Thesis Supervisor: Qiqi Wang
Title: Associate Professor of Aeronautics and Astronautics

Thesis Committe Member: David Darmofal
Title: Professor of Aeronautics and Astronautics

Thesis Committe Member: Semyon Dyatlov
Title: Assistant Professor of Mathematics

Acknowledgments

I thank my advisor, Professor Qiqi Wang, for all his support and guidance throughout my stay at MIT. Through him I have learnt how to be more independent and rigorous and how to tackle seemingly impossible problems.

Additionally, I thank my thesis committee members, Professor David Darmofal and Professor Semyon Dyatlov for providing advice on research directions and literature review. I thank my thesis readers, Dr. Cuong Nguyen and Dr. Eric Nielsen, for comments and suggestions on the thesis.

I also thank my research sponsor and supervisor, Dr. Gregory Laskowski, for funding my research and providing practical advice on many research problems. I thank Argonne National Lab, Oakridge National Lab and Center for Turbulence Research at Stanford University for giving access to their supercomputers, without which acquiring the results in this thesis would not have been possible.

I thank my colleagues in ACDL and friends at MIT for all the great times working together, discussing ideas and sharing stories.

Finally, I thank my parents, Anil and Nirupama, for their love and constant support during my time at MIT. I also thank my sister Neha and my brother-in-law Chinmay Sane for listening to my problems and helping me through some difficult times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Design optimization is used by researchers in many computational fluid dynamics (CFD) applications for improving the performance of various fluid machinery components. The adjoint method for design optimization is a popular technique to obtain gradients of the design objective with respect to design parameters of the flow problem[32]. In the field of aerospace engineering, Jameson [49] performed an adjoint-based shape optimization of an airfoil using the steady state Euler equations and Lyu [24] performed a design optimization of a blended wing body aircraft using the Reynolds Averaged Navier-Stokes (RANS) equations. Majority of the adjoint-based design optimizations carried out in literature use low-fidelity simulations like RANS or Euler [49, 24, 62].

To accurately model the complex fluid flow structures in turbomachinery problems, like flow separation, turbulent boundary layer development and wake mixing, high fidelity simulations like Large Eddy Simulations (LES) are required [70, 36, 66, 8, 51]. Gourdain [36] compared LES to RANS on a transonic turbine vane, and found that LES predicts heat transfer with a much higher accuracy. Coupling the accuracy of LES with the performance of the current generation of supercomputers, adjoint-based design using LES is becoming a more practical option [69].

The adjoint method has wide applicability in gradient-based optimization [49], uncertainty quantification [114] and error estimation [33]. In this method, a single additional solution (known as the adjoint solution), obtained by solving the adjoint equations, is required to calculate the gradient of an objective with respect to multiple parameters. In contrast, the tangent method and the finite difference method for obtaining gradients require multiple linearized flow and nonlinear flow solutions respectively. An adjoint solution is always tied to a particular design objective function.

The application of the adjoint method to LES introduces certain complications. The magnitude of the adjoint solution field diverges to infinity as the LES is performed for a longer time [116, 14, 12]. This divergence introduces a significant error in the gradient computed from the adjoint solution, especially when the design objective function is a long time-averaged quantity. The reason for the divergence is the chaotic nature of turbulent fluid flows. In chaotic systems, nearby initial conditions diverge exponentially fast when the system is solved forwards in time. In mathematical terms, such a system is known to have at least one positive Lyapunov exponent [93]. Even though the instantaneous solutions in a chaotic system are sensitive to parameter perturbations it is widely believed that infinite time-averaged quantities of interest have a linear response [94, 92], assuming ergodicity for the chaotic system. This assumption has been numerically verified in a variety of chaotic systems like the Lorenz system [58, 40] and 2-dimensional chaotic flow over an airfoil [12].

In literature, numerous algorithms have been proposed to overcome the adjoint solution divergence problem and obtain the correct gradients. One of the earliest methods is the ensemble adjoint method [25, 58], in which multiple adjoint solutions are obtained over short interval of time and the gradient is computed by averaging all the adjoint solutions. A disadvantage of this method is that it has a very slow convergence rate and requires a large number of trajectories to achieve a reasonably accurate gradient. Another method is the Fokker-Planck adjoint method [106], which first estimates the physical ergodic measure for the system on a discretization of the phase space of the system (forming a grid) and then computes the gradient by integrating over this measure. The main problem with this method is that for high dimensional systems estimating the measure is very expensive. In addition, the ergodic measure could be fractal, making representation of the measure on a grid difficult. Abramov

[3] proposed the blended response algorithm using the fluctuation-dissipation theorem for computing gradients of long-time averaged quantities of chaotic systems. This method requires the computation of all the Lyapunov exponents and covariant vectors [34] of a chaotic system, which can become prohibitively large for high-dimensional chaotic systems. Finally, a recently discovered method is the Least Squares Shadowing method (LSS) [117, 115]. In this method, a shadowing trajectory of the linearized system is found, from which gradients can be computed. When applying this method to high-dimensional systems, finding the shadowing trajectory becomes an expensive optimization problem.

In order to understand the mechanisms behind the divergence of the adjoint solution for the incompressible Navier-Stokes equations, Wang [116] performed an L_2 norm analysis of the corresponding adjoint equations. The analysis showed that there are two terms which govern the growth or decay dynamics of the L_2 norm or energy of the adjoint solution. Each of the terms is an integral of a quantity that depends on the adjoint solution and the flow solution over the domain of the flow problem. One of the terms is a growth term for the adjoint energy and the other is a dissipation term. The integrand in the growth term is large in regions where the L_2 norm of the gradient of velocity is large. The integrand in the dissipation term is proportional to the viscosity of the fluid. The energy of the adjoint solution diverges to infinity when the growth term dominates the dissipation term. In contrast, the adjoint energy is stable when the dissipation term manages to balance the growth term. Based on these findings, a similar analysis is performed for the adjoint equations of the compressible Navier-Stokes equations in Chapter 2. Chapter 3 describes a new method for solving the problem of diverging adjoint solutions by stabilizing the adjoint equations for compressible Navier-Stokes equations [103] using an injection of artificial viscosity based on the adjoint energy analysis. Chapter 4 analyzes the error due to added viscosity in this new method (known as the viscosity stabilized adjoint method) and applies it to non-intrusive least squares shadowing [74], another recent sensitivity analysis method for chaotic systems.

In order to assess the effectiveness of the viscosity stabilized adjoint method, it is applied to a representative turbomachinery design problem. In a gas turbine engine, the nozzle guide vanes are responsible for directing air onto the turbine blades. Shape design of these vanes is an important problem as it influences the amount of work that can be extracted from the gases exiting the upstream combustion chamber [37]. In particular, the downstream stagnation pressure loss is sensitive to the shape of the trailing edge of the vane. It can alter how the boundary layer develops and where the flow separates on the surface of the vane [102]. Additionally, the trailing edge shape impacts how much heat from the hot gas is absorbed by the vane. Reducing the heat transfer can significantly cool the vane, thereby increasing its life span[42]. The turbine vane flow problem is described in detail in Section 1.4.3.

Design optimization with LES is a challenging task [104] because of the noise in the design objective and gradient evaluations for a particular design parameter and the prohibitive computational cost of the evaluations. Numerous optimizations algorithms have been designed for handling noisy and expensive objective functions [87]. Gradient-based optimization algorithms are preferred as they require fewer iterations to reach locally optimal designs than the typical derivative-free optimization algorithms used in CFD and can scale to a larger number of design parameters[64, 27, 15]. One of the most basic gradient-based methods is the Robbins-Monro algorithm [88], which decides the next design point to evaluate by taking a variable step in the gradient direction, similar to steepest descent. The algorithm has the tendency to get stuck in local optimums and requires extensive tuning of the step parameter [98, 84]. Another optimization algorithm is SNOBFIT (Stable noisy branch and fit) [48], which belongs to the divide and conquer class of optimization algorithms. While the algorithm is robust to noise, it cannot effectively utilize gradient information from all past evaluations as it forms locally approximate quadratic surrogate models. Bayesian optimization algorithms have several properties that make them ideal for design optimization using LES [50, 119]. In this thesis, a standard Bayesian optimization algorithm is modified to handle noisy evaluations more effectively. Chapter 5 discusses the optimization algorithm and its application, combined with the viscosity stabilized adjoint method, to shape design of the trailing edge of a turbine vane using an aero-thermal design objective.

The utilization of the adjoint method in large scale design optimization requires the difficult task of writing a correct and efficient implementation, one that scales to thousands of CPU (Central Processing Unit) cores or utilizes GPU (Graphics Processing Unit) accelerators. There are two common approaches implementing the adjoint method for fluid dynamics simulations, one is the continuous adjoint method and the other is the discrete adjoint method [71]. In the continuous adjoint method, a fluid dynamics simulation (or a flow solver) is treated as a means to provide a numerical approximation to the continuous flow solution, whereas the adjoint flow solver provides an approximation to the continuous adjoint solution which is utilized to compute the design objective gradient. In contrast, gradients obtained from the discrete adjoint method are correct for the discretized flow equations. These gradients can be verified using the finite difference method up to round-off error. The continuous adjoint method provides gradients at truncation error accuracy when the governing equations for the fluid flow are discretized using a dual inconsistent scheme [72]. As this thesis utilizes second-order finite volume schemes which are not dual consistent, the higher error of the continuous adjoint method, in comparison to the discrete adjoint method, reduces its utility.

Manual derivation of the adjoint flow solver using the discrete adjoint method is a laborious process. Employing reverse mode automatic differentiation (AD) to construct it simplifies the implementation process and increases maintainability [11] due to a reduction in the amount of effort required to adapt the adjoint flow solver to different discretizations, design objectives or parameterizations of the flow problem. But, on the other hand, discrete adjoint flow solvers obtained using manual differentiation generally can provide better performance than an adjoint flow solver derived by applying a general purpose reverse mode AD tool to the entire flow solver [46]. This thesis focuses on a special purpose AD tool for flow solvers.

There are two methods for applying AD to construct a discrete adjoint flow solver, operator overloading and source transformation. In operator overloading, floating point operations are overloaded to perform linearizations and record the result and order of operations. This record is used to execute the adjoint flow solver. The performance of the adjoint flow solver derived using operator overloading is heavily degraded in comparison to the flow solver. This is especially true for unsteady flow solvers that use explicit time integration, where the unsteady adjoint flow solver can be a factor of 10 to 50 times slower than the flow solver, for example when using *Adept* [46], *CoDiPack* [95] or *FAD* [6]. The primary reason for the slowdown is that the compiler does not have access to the call graph of the adjoint flow solver as it is generated at runtime [11]. This reduces opportunities for compiler optimizations through static analysis of the code and may result in unnecessary computation and storage of intermediate variables [11].

Source transformation based AD involves parsing the source code of a flow solver, representing the computations in the form of a graph and then deriving the corresponding adjoint flow solver using a similar procedure like the one used in operator overloading [109, 43]. This method gives better performance than operator overloading based AD [9]. But, for many source transformation tools, the source code has to be written in a restrictive manner. For example, OpenAD [109] only supports a subset of FORTRAN and for efficient usage requires the splitting of the flow solver into numerical and non-numerical components (source files). The current source transformers cannot understand many of the advanced programming language features of modern FORTRAN or C/C++ that can facilitate flow solver development and increase maintainability.

This thesis adopts an approach that combines operator overloading method with source transformation. There are existing AD tools/libraries that follow this method. Examples of such libraries are *Theano* [7] and *Tensorflow* [1]. They utilize operator overloading to build a computational graph of the flow solver, from which the "adjoint computational graph" can be derived. Once constructed, both graphs are transformed by these libraries into optimized source code routines, ready for compilation and then execution. This method combines the usability advantage of operator overloading and the performance of source transformation. The aforementioned libraries implement this method by defining an API (Application Programming Interface) in a high-level language, like Python or C++, for specifying the computations involved in a numerical method. They rely on compilers like *gcc* or *clang* to transform the generated C/C++ code into optimized assembly tuned for specific hardware architectures. This approach, known as domain-specific modelling (DSM), has been widely used in many simulation applications [52].

A deficiency with existing DSM libraries is that they miss out on combining different types of computations (like arithmetic expressions and memory accesses) into a single loop over all the elements of the mesh. Fusing computations together into a single loop leads to better cache utilization of processors and fewer memory allocations. An AD method that is adopts this approach and efficiently reuses memory can provide better performance and lower execution time than the current libraries. Bischof [10] proposed a hierarchical AD method for efficiently differentiating a computational graph representing the code of a numerical simulation. Chapter 6 describes a two-level computational graph based AD method, which is inspired by the hierarchical AD method, that can be used to construct a high performance and memory efficient explicit unsteady flow solver for heterogeneous architectures and derive the corresponding discrete adjoint flow solver using automatic differentiation.

1.1 Thesis contributions

The main contributions of this thesis address the various problems associated with design optimization for LES. The methods and tools developed in the thesis are listed below

1. **Viscosity stabilized adjoint method**
   A method for producing high-dimensional gradients for LES. It stabilizes the diverging adjoint solution by adding artificial viscosity to the adjoint equations.

2. **Adjoint-based design optimization for LES**
   A framework for design optimization using LES with gradients. It biases standard Bayesian optimization to do more exploration for noisy objective functions.

3. **Two-level computational graph method**

   A tool for running expensive numerical simulations on current and next generation supercomputers. It achieves high performance and low memory usage by representing computations in a two-level graph structure.

1.2 Physical models

In this thesis, turbulent fluid flow is modeled using the compressible Navier-Stokes equations with the ideal gas law serving as an approximation to the thermodynamic state equation [29]. The gas is assumed to be air.

$$
\begin{aligned}
&\text{In } \mathbf{x} \in V, t \in [0,T],\\
&\quad \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = s_\rho\\
&\quad \frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) + \nabla p = \nabla \cdot \sigma + \mathbf{s}_{\rho \mathbf{u}}\\
&\quad \frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho E \mathbf{u} + p\mathbf{u}) = \nabla \cdot (\mathbf{u} \cdot \sigma + \alpha \rho \gamma \nabla e) + s_{\rho E}\\
&\quad \sigma = \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T) - \frac{2\mu}{3}(\nabla \cdot \mathbf{u})\mathbf{I}\\
&\quad p = (\gamma - 1)\rho e\\
&\quad e = E - \frac{\mathbf{u} \cdot \mathbf{u}}{2}\\
&\quad c = \sqrt{\frac{\gamma p}{\rho}}
\end{aligned}
\tag{1.1}
$$

where V is the domain of the flow problem, T is the terminal time of the flow problem (which can be ∞), ρ is the density, $\mathbf{u}$ is the velocity vector, ρE is the total energy, p is pressure, e is internal energy of the fluid, c is the speed of sound, γ is the isentropic expansion factor (for air $\gamma = 1.4$), μ is the viscosity field modeled using Sutherland's law for air

$$\mu = \frac{C_s T^{3/2}}{T + T_s} \tag{1.2}$$

where $T_s = 110.4\,K$ and $C_s = 1.458 \times 10^{-6} \frac{kg}{m\,s\,\sqrt{K}}$. α is the thermal diffusivity modeled using

$$\alpha = \frac{\mu}{\rho Pr} \tag{1.3}$$

where Pr is the Prandtl number (for air $Pr = 0.71$). $s_\rho, \mathbf{s}_{\rho\mathbf{u}}, s_{\rho E}$ denote the source terms prescribed in a flow problem. In addition to the above equations, depending on the specifics of the flow problem, boundary conditions are prescribed on each of the boundary regions of the domain (S). Lastly, appropriate initial conditions for the flow variables, $\rho, \rho\mathbf{u}, \rho E$, are defined.

1.3 Numerical methods

The numerical approximation to the flow solution is obtained using large eddy simulations on a discretized domain of the flow problem, generally known as a mesh or a grid. In an LES, the large scale eddies of the flow are resolved by the grid while the contribution of the small scale eddies to the filtered Navier-Stokes equations are modeled using a sub-grid scale Reynolds stress model [29]. The choice of the LES model can have a large impact on the accuracy of the relevant statistical quantities of interest of the flow. In this thesis, in all the simulations an implicit LES model is used. In this model, the numerical error of the discretization scheme serves as the LES model. It has been shown that when using a dissipative discretization method, the numerical viscosity from the grid can be of the same order of magnitude as the sub-grid scale viscosity [68]. Hence, using an explicit LES model may be unnecessary.

The numerical solution of the flow problem is obtained on an unstructured hexahedral mesh using a second order finite volume method (FVM) [59]. The central differencing scheme is used to interpolate cell averages of the flow solution onto faces of the mesh [112]. The numerical fluxes for the conservative flow variables are computed using the Roe approximate Riemann solver [89]. An explicit time integration scheme, the strong stability preserving third order Runge-Kutta method [63], is used for time marching the numerical flow solution. The size of the time step is determined using the acoustic Courant-Friedrichs-Lewy (CFL) condition [20]. The flow solver is implemented in Python using a library ($adFVM$) that provides a high-level abstract application programming interface for writing efficient CFD applications. The flow solver is parallelized using the Message Passing Interface (MPI) library. More details about the flow solver and the implementation of the discrete adjoint method for unsteady problems can be found in Chapter 6.

1.4 Test cases

There are a couple of turbulent flow problems and a chaotic system that are used as test cases for verifying the effectiveness of various algorithms devised in this thesis. Their description is given in this section. For the flow problems, the following reference quantities are defined

$$\begin{aligned} \rho_r &= 1\, kg/m^3 \\ u_r &= 100\, m/s \\ p_r &= C_r \rho_r u_r^2 \\ \mu_r &= 1.8 \times 10^{-5}\, m^2/s \end{aligned} \tag{1.4}$$

where $C_r = 10.1325$ is a fixed constant. The source term in the flow equations for all the flow problems is 0, unless otherwise specified.

1.4.1 Lorenz 63 system

The Lorenz system is a three degree of freedom ordinary differential equation (ODE). It is one of the smallest (by degree of freedom) ODEs that forms a chaotic system, that is, the solution of the system is sensitive to initial conditions for certain parameter values of the system [99] and is aperiodic. The Lorenz system was first used by Edward Lorenz to model atmospheric convection[61]. It approximates 2-dimensional thermal convection between two parallel planes. It is controlled by three parameters

ρ_l, β_l and σ_l.

$$\frac{dw_1^l}{dt} = \sigma_l(w_2^l - w_1^l) \tag{1.5}$$

$$\frac{dw_2^l}{dt} = w_1^l(\rho_l - w_3^l) - w_2^l \tag{1.6}$$

$$\frac{dw_3^l}{dt} = w_1^l w_2^l - \beta_l w_3^l \tag{1.7}$$

where the state of the system is given by w_1^l, w_2^l, w_3^l.

For the test case, the standard parameters of the chaotic Lorenz system are used, $\rho_l = 28$ and $\beta_l = \frac{8}{3}$ and $\sigma_l = 10$. The system of equations are integrated using explicit fourth-order Runge-Kutta method with time step size equal to 0.01 time units. A plot of w_3^l as a function of time is show in Figure 1-1. It shows that the state of the system is aperiodic. The design objective is the following infinite time-averaged objective function

$$J = \lim_{T\to\infty} \frac{1}{T}\int_0^T w_3^l\, dt \tag{1.8}$$

Figure 1-2 shows the design objective as a function of ρ_l. The uncertainty in the time-averaged design objective is computed using the time series analysis techniques discussed in Section 5.1.2.

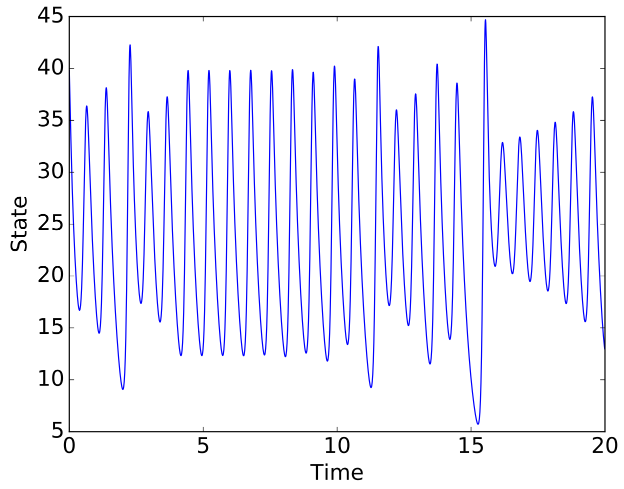


Figure 1-1: Plot of w_3^l as a function of time (represented by time units)

The design objective gradient computed with respect to ρ_l is found to be 1.

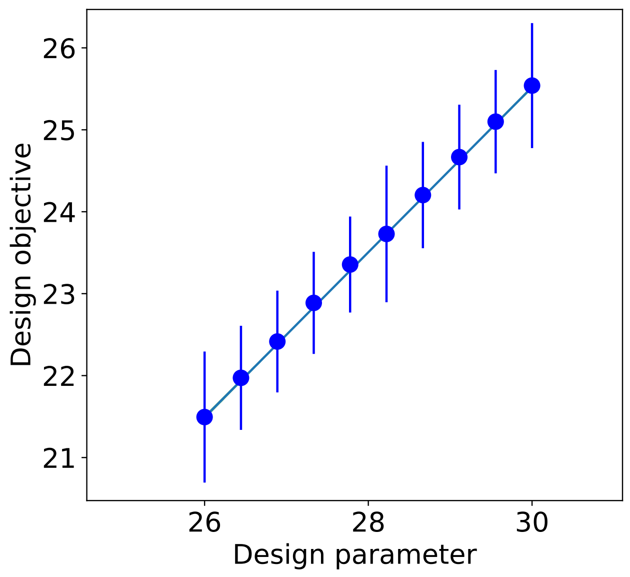


Figure 1-2: Plot of design objective vs ρ_l. Blue dots denote the time-averaged design objective and the blue error bars signify one standard deviation away from the mean. The blue line shows the design objective gradient.

1.4.2 3-dimensional subsonic flow over cylinder

The first flow problem is 3-dimensional subsonic flow over a cylinder at Reynolds number $Re = 1,100$ and inflow Mach number $Ma = 0.1$. The reference velocity of the flow is $u_r = 31.4\,m/s$, which is also the inflow velocity. The axial length of the domain is $60d$, where $d = 0.25\,mm$ is the diameter of the cylinder. The Reynolds number is defined using the diameter of the cylinder and the density, velocity and viscosity of inflow. The span-wise extent, at $z = 2d$, is sufficient to capture most of the important flow features, like a turbulent wake and flow separation. The size of the mesh is approximately 700,000 cells, with 50 cells in the span-wise direction. Periodic boundary condition is used in the span-wise direction. The surface of the cylinder is maintained at a constant temperature, 300 K. Static pressure equal to $1\,atm$ is prescribed on the outlet boundary. Visualizations of the domain of the flow problem and the mesh are shown in Figures 1-3,1-4 and 1-5. Figure 1-6 shows a visualization of the magnitude of an instantaneous velocity field for flow over the cylinder. Tritton [107] performed experiments of flow over a cylinder at low Reynolds numbers, whereas Roshko [90] performed experiments at high Reynolds numbers. Kravchenko [54] conducted numerical simulations of flow over a cylinder at $Re = 3,900$.

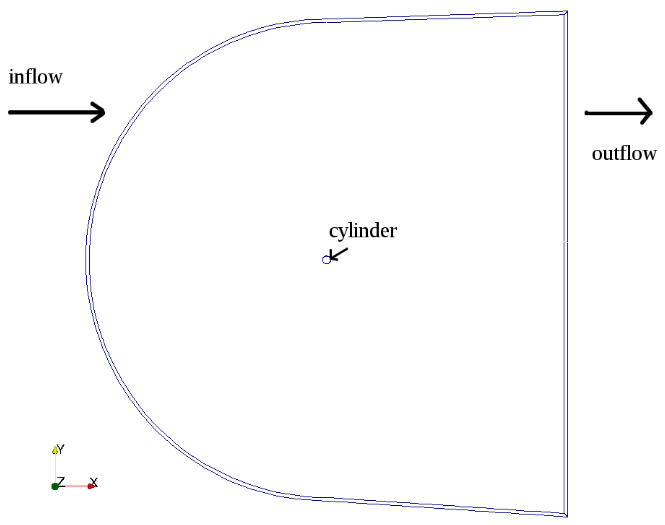


Figure 1-3: Visualization of the domain for simulating subsonic flow over a cylinder

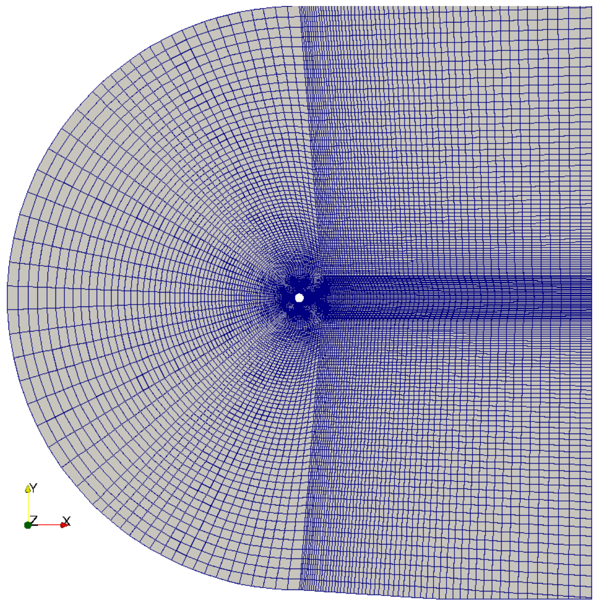


Figure 1-4: Visualization of the mesh for the flow over cylinder problem

The design objective function for the adjoint solution is time-averaged drag coefficient over the cylinder. The instantaneous drag is defined using an incompressible

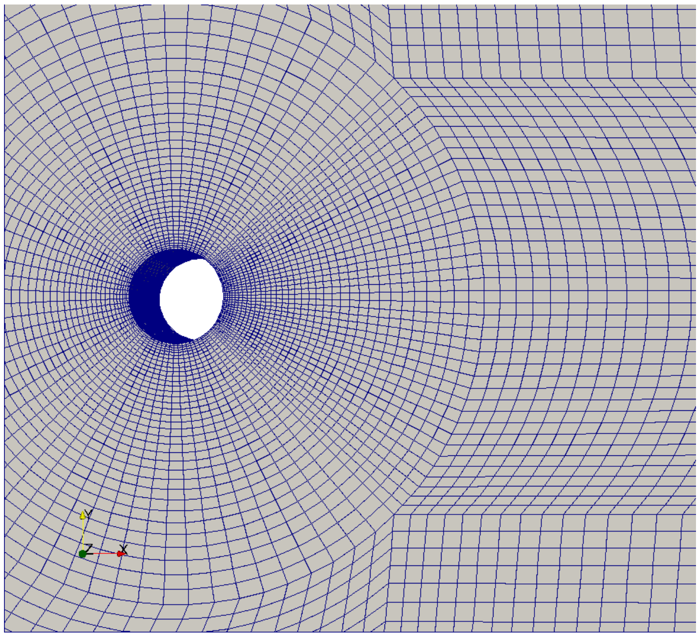


Figure 1-5: Visualization of the mesh near the surface of the cylinder

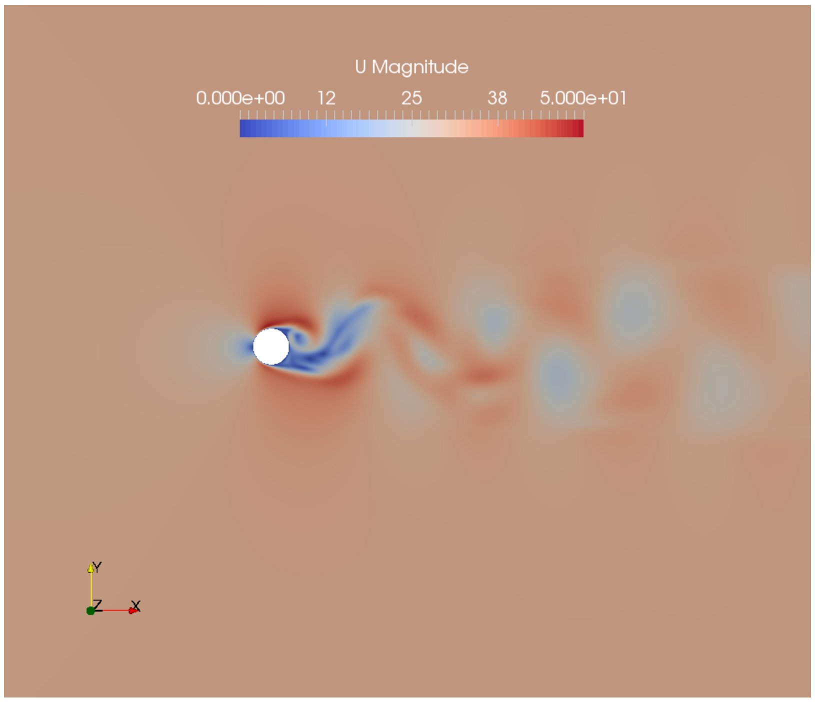


Figure 1-6: Visualization of the magnitude of the velocity field (units: m/s)

approximation.

$$\bar{J}(\rho, \rho\mathbf{u}, \rho E) = \frac{2}{\rho_r u_r^2 z d} \lim_{T\to\infty} \frac{1}{T} \int_0^T \int_{\partial S} \left(p n_x - \mu \boldsymbol{n} \cdot \nabla u_x\right) dS\, dt \tag{1.9}$$

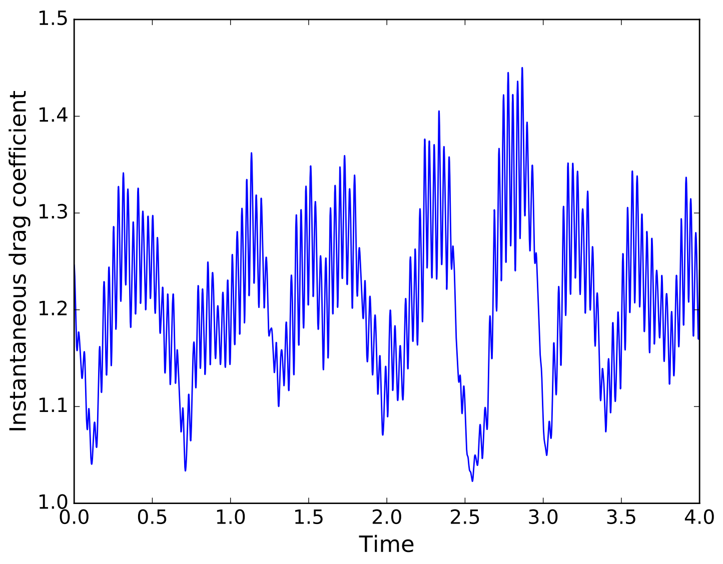


Figure 1-7: Visualization of the instantaneous drag coefficient as a function of time (represented in time units).

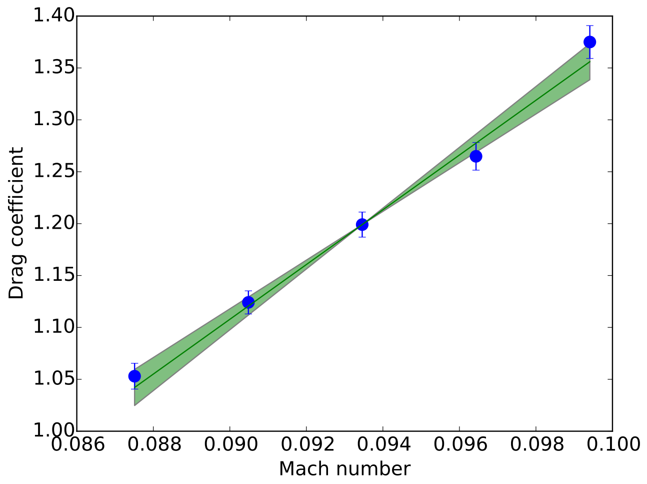


Figure 1-8: Drag coefficient as a function of inlet Mach number. Blue dots denote the mean drag coefficient and the blue error bars signify one standard deviation away from the mean. The green shaded regions denotes the gradient estimated using linear regression and the associated uncertainty given by one standard deviation.

The boundary integral in the above equation is performed over the surface of the cylinder. A visualization of the instantaneous drag coefficient is shown in Figure 1-7.

Using a CFL number of 1.2, the flow simulation runs for $1,000,000$ time steps, which corresponds to approximately 12 time units, where one time unit (t_r) is the amount of time that the flow takes to traverse the maximum axial length of the domain or $60d$.

$$t_r = \frac{60d}{u_r} \tag{1.10}$$

This time interval is sufficient to obtain a statistically converged estimate of the design objective. Figure 1-8 shows a plot of the design objective with respect to the Mach number at the inlet boundary. The uncertainty in the time-averaged design objective is computed using the time series analysis techniques discussed in Section 5.1.2. The design objective gradient is computed with respect to the Mach number of the flow at the inlet boundary. At $Ma = 0.093$, the gradient is approximately 25.0 ± 2.1. The gradient is estimated using ordinary least squares based linear regression and the uncertainty in the gradient is computed from the standard deviation of the relevant estimator [41].

The drag coefficient shown in Figure 1-8 for $Ma = 0.093$ (which corresponds to $Re = 1,100$) is 1.2 ± 0.03. It reasonably matches the drag coefficient reported in literature from experiments [118, 90, 107] for $Re = 1,100$, which is approximately equal to 1.0 ± 0.15.

1.4.3 Transonic flow over a turbine vane

The second flow problem is transonic flow over a nozzle guide vane designed by researchers at the Von Karman Institute (VKI) [5] for gas turbines.

A visualization of the domain of the flow problem is shown in Figure 1-9. In a gas turbine engine, gases exiting from the combustion chamber impinge on a circular cascade of nozzle guide vanes. High temperature and high pressure subsonic flow hits the leading edge of each vane and then accelerates over the suction side, reaching close to Mach one near the trailing edge on the suction side of the vane where the boundary layer transitions from laminar to turbulent. In contrast, on the pressure side, the boundary layer stays laminar. The boundary layers after separation from the surface of the vane, merge into a turbulent wake. The exit pressure in gas turbine engines is much larger than 1 atm.

In the experimental setup of the VKI vane, the circular cascade is approximated as a linear cascade. The Reynolds number of the flow, computed using the chord length of the vane, is approximately 1,000,000. The downstream isentropic Mach number is 0.92. The chord length of the vane is $c_l = 67.6\,mm$. The Reynolds number is defined using the chord length of the vane and the density, velocity and viscosity of the inflow. Gourdain [36] and Morata [70] performed large eddy simulations of this flow problem at various Reynolds numbers.

A 2-dimensional slice of the computational domain used to model the flow for this thesis is shown in Figure 1-9. The x-direction is the direction of the inflow, the y-direction is the direction of the periodic cascade from the bottom to the top surface and the z-direction is perpendicular to the x and y-directions. The tip of the leading edge serves as the origin of the computational domain. Periodic boundary conditions are used in directions transverse to the flow direction. Static pressure equal to $1\,atm$ is prescribed on the outlet boundary. A non-reflecting characteristic boundary condition is used on the inlet boundary of the domain [83]. The stagnation pressure (computed from the downstream isentropic Mach number) and stagnation temperature ($420\,K$) are prescribed on the inlet boundary. Using this information on one side of the boundary and the solution fields from the interior of the domain on the other side, the Roe approximate Riemann solver is used to propagate the fluxes on the inlet boundary. The surface of the vane is maintained at a constant temperature of $300\,K$. The span-wise extent is approximately $0.16\,c_l$, which is sufficient to accurately capture most of the important flow features [36]. In order to resolve the small scale eddies of the flow near the wall, the dimensionless wall normal cell spacing (y_+) needs to be below 1 wall unit [18]. However, this thesis does not adopt this practice due to computational cost considerations. As the Reynolds number of the flow is high, the y_+ restriction along with a maximum CFL number of 1.2 over the domain of the flow problem, significantly lowers the maximum time step size that can be used by an explicit time integration scheme. To obtain results from the flow simulation in a reasonable time frame, the maximum y^+ is kept at 10. This does not appear to lead to significant loss in physical accuracy of the simulation as demonstrated by the comparison of the experimental heat transfer and static pressure data, obtained by Arts [5], with numerical data generated by an LES on an under resolved wall grid in Figures 1-10 and 1-11. The LES on the under resolved wall grid appears to provide better results than the LES on a resolved wall grid. This potentially can be attributed to the difference in sub-grid scale models, the former uses an implicit LES model while the latter uses the Wall-Adapting Local Eddy Viscosity (WALE) model which could be introducing modelling error in the solution. The maximum x^+ is 250 and maximum z^+ is 50. The total number of cells in the mesh of the computational domain is approximately 16 million. A visualization of the magnitude of an instantaneous velocity field of the flow over the turbine vane near the trailing edge is shown in Figure 1-12.

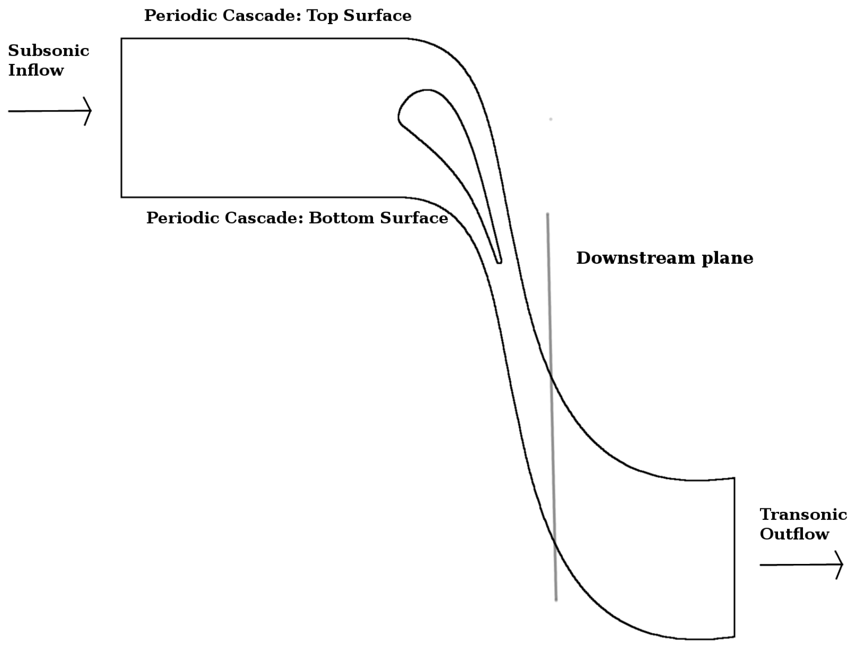

Periodic Cascade: Top Surface
Subsonic Inflow
Periodic Cascade: Bottom Surface
Downstream plane
Transonic Outflow

Figure 1-9: Turbine vane computational domain

The design objective for the flow problem is infinite-time averaged and mass flow averaged pressure loss coefficient on a plane $0.23\,c_l$ downstream of the trailing edge of the vane and perpendicular to the direction of the inflow.

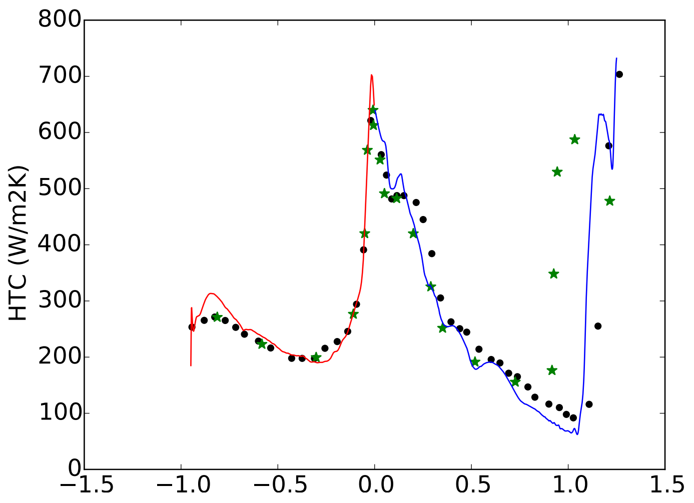


Figure 1-10: Comparison of heat transfer coefficient on the surface of the vane obtained from experimental data produced by Arts [5] (denoted by black dots), numerical data from an LES on a resolved wall grid produced by Gourdain [36] (denoted by green colored stars) and numerical data generated by an LES on an under-resolved wall grid (denoted by blue and red colored lines) at isentropic Mach number 0.9 and Reynolds number 10^6. In the figure, the color blue denotes suction side, the color red denotes pressure side. The x-axis represents the distance from the leading edge along the surface of the vane normalized by the chord length.

$$\begin{aligned}
\bar{J} &= \frac{\bar{p}_{t,l}}{p_{t,in}} \\
\bar{p}_{t,l} &= \lim_{t_e \to \infty} \frac{1}{t_e} \int_0^{t_e} \frac{\int_{S_p} \rho_p u_n (p_{t,in} - p_{t,p}) \, dS_p}{\int_{S_p} \rho_p u_n \, dS_p} \, dt \\
p_{t,p} &= p_p \left(1 + \frac{\gamma - 1}{2} M_p^2\right)^{\frac{\gamma}{\gamma-1}} \\
p_{t,in} &= p_{ex} \left(1 + \frac{\gamma - 1}{2} M_{is}^2\right)^{\frac{\gamma}{\gamma-1}}
\end{aligned} \tag{1.11}$$

where M_p is the Mach number on the plane, $p_p, \rho_p, p_{t,p}$ are the pressure, density and stagnation pressure on the plane respectively, S_p represents the plane surface, u_n is the velocity normal to the plane surface, $p_{t,in}$ is the inlet stagnation pressure, p_{ex} is the exit static pressure and M_{is} is the downstream isentropic Mach number. A plot of the instantaneous pressure loss coefficient as a function of time is shown in Figure 1-13.

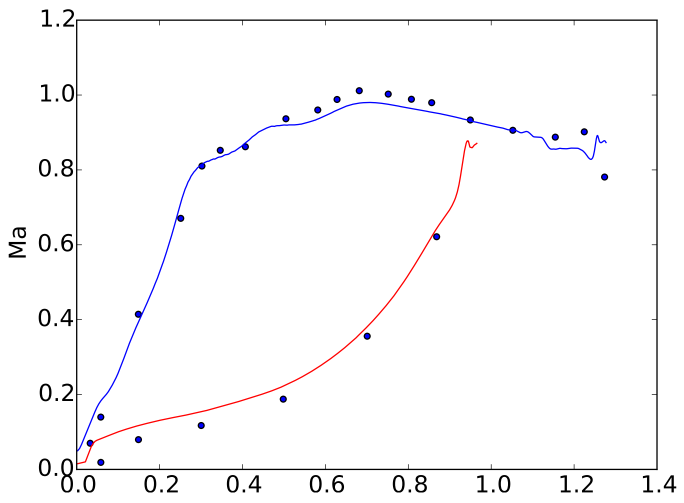


Figure 1-11: Comparison of isentropic Mach number on the surface of the vane obtained from experimental data produced by Arts [5] (denoted by blue dots) and numerical data generated by an LES on an under-resolved wall grid (denoted by blue and red colored lines) at isentropic Mach number 0.875 and Reynolds number 10^6. In the figure, the color blue denotes suction side, the color red denotes pressure side. The x-axis represents the distance from the leading edge along the surface of the vane normalized by the chord length.

1.5 Adjoint method for unsteady flows

In this section, the adjoint method applied to unsteady flow problems governed by the time-dependent compressible Navier-Stokes equations is introduced. It provides a way for computing gradients of a design objective with respect to multiple parameters of a flow problem [16, 78, 79].

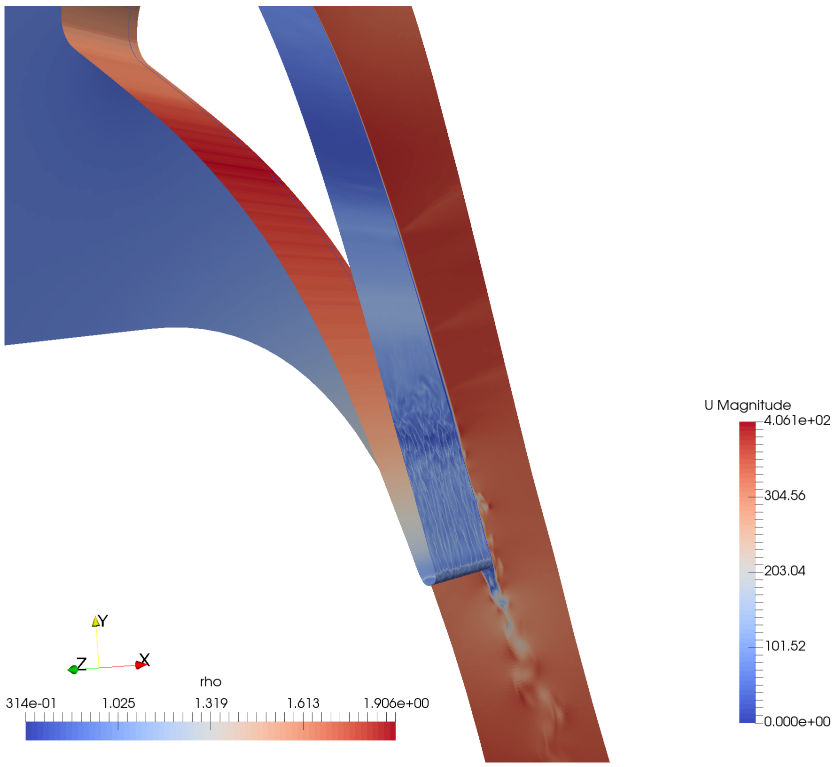


Figure 1-12: An instantaneous snapshot of the flow field near the trailing edge of the vane. The density (units: kg/m^3) is colored over the surface of the vane and the magnitude of the velocity field (units: m/s) is colored on the planar cross section.

Rewriting the governing equations from Equation 1.1 in vector form

$$\begin{aligned}
\frac{\partial \mathbf{w}}{\partial t} + \nabla \cdot \mathbf{F} &= \nabla \cdot \mathbf{F^v} + \mathbf{s} \\
\mathbf{w} &= \begin{pmatrix} \rho \\ \rho \mathbf{u} \\ \rho E \end{pmatrix} \\
\mathbf{F} &= \begin{pmatrix} \rho \mathbf{u} \\ \rho \mathbf{u}\mathbf{u} \\ (\rho E + p)\mathbf{u} \end{pmatrix} \\
\mathbf{F^v} &= \begin{pmatrix} 0 \\ \sigma \\ \mathbf{u} \cdot \sigma + \alpha \rho \gamma \nabla e \end{pmatrix}
\end{aligned} \tag{1.12}$$

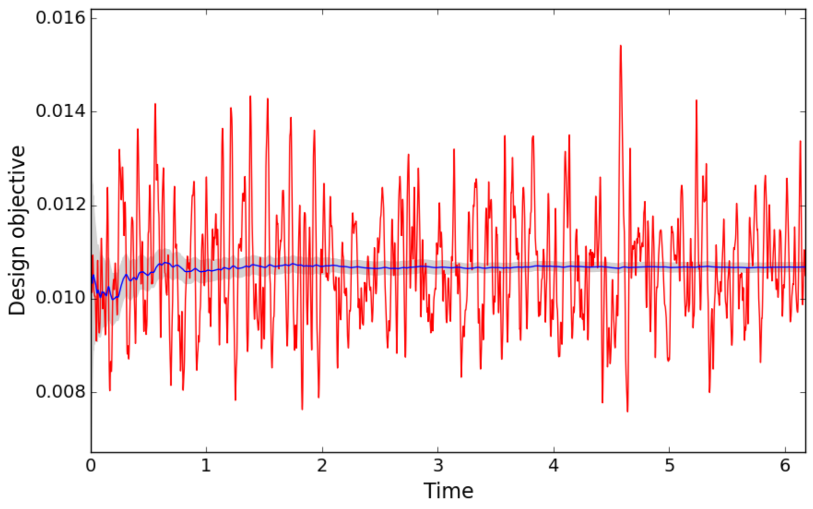


Figure 1-13: Instantaneous pressure loss coefficient ($\bar{p}_l$) plotted as a function of time (represented by time units). The blue time series denotes the cumulative mean and the gray shaded area denotes a single standard deviation of the sample mean. The procedure for time averaging is discussed in Section 5.1.2.

Using the Einstein summation notation in Euclidean space Equation 1.12 can be rewritten as

$$\frac{\partial w_i}{\partial t} + \frac{\partial F_{ij}}{\partial x_j} = \frac{\partial F^v_{ij}}{\partial x_j} + s_i, i = 1...5 \tag{1.13}$$

The source term is parameterized by a finite-dimensional vector. The vector serves as the system parameter of the flow problem. The design objective gradient is computed with respect to this variable. Additional possible system parameters are inflow Mach number or shape design parameters.

In many fluid flow problems, the design objectives of interest are infinite time-averaged quantities. For practical reasons, they are generally approximated by a finite time-average. Without loss in generality, consider a finite time-averaged objective ($\bar{J}$) which is a function of the flow solution on the boundary surface (S).

$$\bar{J} = \frac{1}{T}\int_0^T \int_S J(\mathbf{w})\, dS\, dt \tag{1.14}$$

T is a large enough such that $\bar{J}$ is sufficient to approximate the infinite time-averaged design objective.

The first step in deriving the adjoint equations involves forming a Lagrangian with the introduction of a new vector field known as the adjoint solution ($\hat{\mathbf{w}}$).

$$\bar{J} = \frac{1}{T}\int_0^T \int_S J(w_i)\, dS\, dt \tag{1.15}$$
$$+ \int_0^T \int_V \hat{w}_i(\frac{\partial w_i}{\partial t} + \frac{\partial(F_{ij} - F^v_{ij})}{\partial x_j} - s_i)\, dV\, dt \tag{1.16}$$

Taking the total differential of both sides of the above equations with respect to the system parameters results in the following equations

$$\begin{aligned}\delta\bar{J} &= \frac{1}{T}\int_0^T \int_S (\frac{\partial J}{\partial w_i}\delta w_i)\, dS\, dt \\ &+ \int_0^T \int_V \hat{w}_i(\frac{\partial \delta w_i}{\partial t} + \frac{\partial \delta F_{ij} - \delta F^v_{ij}}{\partial x_j} - \delta s_i)\, dV\, dt\end{aligned} \tag{1.17}$$

Integrating by parts the second integral term in the right hand side of the above equation in time and space

$$\begin{aligned}\delta\bar{J} = \frac{1}{T}\int_0^T \int_S (\frac{\partial J}{\partial w_i}\delta w_i)\, dS\, dt + \int_V (\hat{w}_{i|T}\delta w_{i|T} - \hat{w}_{i|0}\delta w_{i|0})\, dV \\ - \int_0^T \int_V \frac{\partial \hat{w}_i}{\partial t}\delta w_i\, dV\, dt + \int_0^T \int_S \hat{w}_i(\delta F_{ij} - \delta F^v_{ij})n_j\, dS\, dt \\ - \int_0^T \int_V \frac{\partial \hat{w}_i}{\partial x_j}(\delta F_{ij} - \delta F^v_{ij})\, dV\, dt \\ - \int_0^T \int_V \hat{w}_i \delta s_i\, dV\, dt\end{aligned} \tag{1.18}$$

Differentiating F_{ij} with respect to w_k to form the tensor A_{ijk}

$$A_{ijk} = \frac{\partial F_{ij}}{\partial w_k} \tag{1.19}$$

The above tensor can be used to represent δF_{ij} as $A_{ijk}\delta w_k$. The terms of the tensor are described in Appendix A.1.

The tensor F^v_{ij} can be simplified as the product of a term, D_{ijkl} that only depends on the solution, and the gradient of the solution $\frac{\partial w_k}{\partial x_l}$. In other words, F^v_{ij} can be rewritten as

$$F^v_{ij} = D_{ijkl}\frac{\partial w_k}{\partial x_l} \tag{1.20}$$

The terms of the above tensor are described in Appendix A.2.

Substituting the new tensors, A_{ijk} and D_{ijkl}, in the respective terms in Equation 1.18 and rearranging the terms

$$\begin{aligned}\delta\bar{J} =& \frac{1}{T}\int_0^T\int_S [(\frac{\partial J}{\partial w_i} + \hat{w}_k A_{kji} n_j - \hat{w}_k \frac{\partial D_{kjml}}{\partial w_i}\frac{\partial w_m}{\partial x_l} n_j)\delta w_i - \hat{w}_i D_{ijkl}\delta\frac{\partial w_k}{\partial x_l} n_j]\, dS\, dt \\ &+ \int_V (\hat{w}_{i|T}\delta w_{i|T} - \hat{w}_{i|0}\delta w_{i|0})\, dV \\ &- \int_0^T\int_V (\frac{\partial \hat{w}_i}{\partial t} + \frac{\partial \hat{w}_k}{\partial x_j} A_{kji})\delta w_i\, dV\, dt \\ &+ \int_0^T\int_V (\frac{\partial \hat{w}_i}{\partial x_j} D_{ijkl}\frac{\partial \delta w_k}{\partial x_l} + \frac{\partial \hat{w}_i}{\partial x_j}\frac{\partial D_{ijkl}}{\partial w_m}\frac{\partial w_k}{\partial x_l}\delta w_m)\, dV\, dt \\ &- \int_0^T\int_V \hat{w}_i \delta s_i\, dV\, dt\end{aligned} \tag{1.21}$$

Integration by parts can be used to give

$$\begin{aligned}\int_0^T\int_V \frac{\partial \hat{w}_i}{\partial x_j} D_{ijkl}\frac{\partial \delta w_k}{\partial x_l}\, dV\, dt =& \int_0^T\int_S \frac{\partial \hat{w}_i}{\partial x_j} D_{ijkl}\delta w_k n_l\, dS\, dt \\ &- \int_0^T\int_V \frac{\partial}{\partial x_l}(\frac{\partial \hat{w}_i}{\partial x_j} D_{ijkl})\delta w_k\, dV\, dt\end{aligned} \tag{1.22}$$

The second term in the round bracket of the second last term in the right hand side of Equation 1.21 is represented as $A^v_{kji} = \frac{\partial D_{kjml}}{\partial w_i}\frac{\partial w_m}{\partial x_l}$. Additionally, in Equation 1.21, the initial condition is a constant and it does not depend on the system parameters. Hence, $\delta w_{i|0} = 0$. Using this fact, Equation 1.21 simplifies to

$$\delta\bar{J} = -\int_0^T\int_V \hat{w}_i\delta s_i\, dV\, dt \tag{1.23}$$

provided the following adjoint equations are satisfied over the domain of the flow problem

$$-\frac{\partial \hat{w}_i}{\partial t} - (A_{kji} - A^v_{kji})\frac{\partial \hat{w}_k}{\partial x_j} = \frac{\partial}{\partial x_l}(D_{kjil}\frac{\partial \hat{w}_k}{\partial x_j}) \tag{1.24}$$

with the following adjoint boundary conditions on the boundary of the domain

$$(\frac{1}{T}\frac{\partial J}{\partial w_i} + \hat{w}_k(A_{kji} - A^v_{kji})n_j + \frac{\partial \hat{w}_k}{\partial x_j}D_{kjil}n_l)\delta w_i - \hat{w}_i D_{ijkl}\delta\frac{\partial w_k}{\partial x_l}n_j = 0 \tag{1.25}$$

and terminal condition $\hat{w}_{i,T} = 0$. The existence of a terminal condition and the absence of an initial condition implies that the adjoint equations have to be integrated backwards in time. As the adjoint equations require the solutions of the governing (primal) equations, the procedure to compute design objective gradient involves first solving the primal equations from time $t = 0$ to T and then solving the adjoint equations from T to 0.

If the source term is parameterized by $s_i = f_i(\boldsymbol{\theta^s})$, where $\boldsymbol{\theta^s}$ is a finite dimensional vector, then the following equation

$$\frac{\partial \bar{J}}{\partial \theta^s_j} = -\int_0^T \int_V \hat{w}_i \frac{\partial f_i}{\partial \theta^s_j}\, dV\, dt \tag{1.26}$$

provides the design objective gradient with respect to the finite-dimensional parameterization of the source term.

Chapter 2

Adjoint energy analysis

In this chapter, energy analysis of the adjoint equations is performed. Section 2.1 demonstrates the divergence of the adjoint solution on transonic flow over a turbine vane. Section 2.2 discusses the techniques for analyzing the divergence of the adjoint solutions. Section 2.3 describes non-entropy based symmetrization techniques for the Euler equations. Section 2.4 describes entropy-based symmetrization techniques for the compressible Navier-Stokes equations. Section 2.5 details the contribution of the various terms in the adjoint energy analysis. Section 2.6 elaborates on one of the growth terms in the adjoint energy analysis and derives an indicator field for the divergence of the adjoint solution. Finally, Section 2.7 demonstrates the indicator field on a couple of flow problems.

2.1 Divergence of adjoint solution

As discussed in Section 1, the adjoint solution diverges exponentially fast as an LES of a turbulent fluid flow is performed for a longer time [116, 14, 12]. In this section, the adjoint method is applied to the unsteady transonic flow over a turbine vane, described in Section 1.4.3. The design parameter of the system is the scaling parameter of a Gaussian shaped source term upstream of the leading of the vane. The manufactured form of the source term is given below

$$
\begin{aligned}
G &= \frac{\theta}{\Delta T}\exp(-\frac{(\mathbf{x}-\mathbf{x_0})^2}{2l^2}) \\
s_\rho &= \rho_r G \\
\mathbf{s}_{\rho\mathbf{u}} &= \begin{pmatrix} \rho_r u_r \\ 0 \\ 0 \end{pmatrix} G \\
s_{\rho E} &= (\frac{\rho_r u_r^2}{2} + \frac{p_r}{\gamma - 1})G
\end{aligned}
\tag{2.1}
$$

where $\mathbf{x_0} = (-0.296, 0.148, 0.074)\, c_l$, $l = 0.191\, c_l$, $\Delta T = 1$ time unit and θ is the non-dimensional scaling parameter. 1 time unit (t_r) is the time the flow takes to travel from the inlet boundary to the outlet boundary.

$$
t_r = \frac{1.5c_l}{u_r} \tag{2.2}
$$

Figure 2-1 shows a visualization of the source term $s_{\rho u_1}$.

Figure 2-2 shows a plot of the pressure loss coefficient as a function of the scaling parameter of the source term. The design objective gradient computed with respect to the scaling parameter is $(-4.76 \pm 0.22) \times 10^{-3}$. The time-averaging is performed over an interval whose length is 20 time units. Using a CFL equal to 1.2, the number of time steps in the flow simulation is $1,000,000$. The design objective gradient is estimated using ordinary least squares based linear regression.

A total of 2 adjoint solutions are computed. The first adjoint solution is computed over a time interval whose length is 0.2 time units and the second adjoint solution is computed over a time interval whose length is 2 time units. The former is known as the short time adjoint solution, while the latter is known as the long time adjoint solution.

The long time adjoint solution for the turbine vane problem diverges exponentially backwards in time, as shown in Figure 2-3. Figure 2-4 shows a visualization of the density adjoint field of the long time adjoint solution for the turbine vane, at $T = 1$

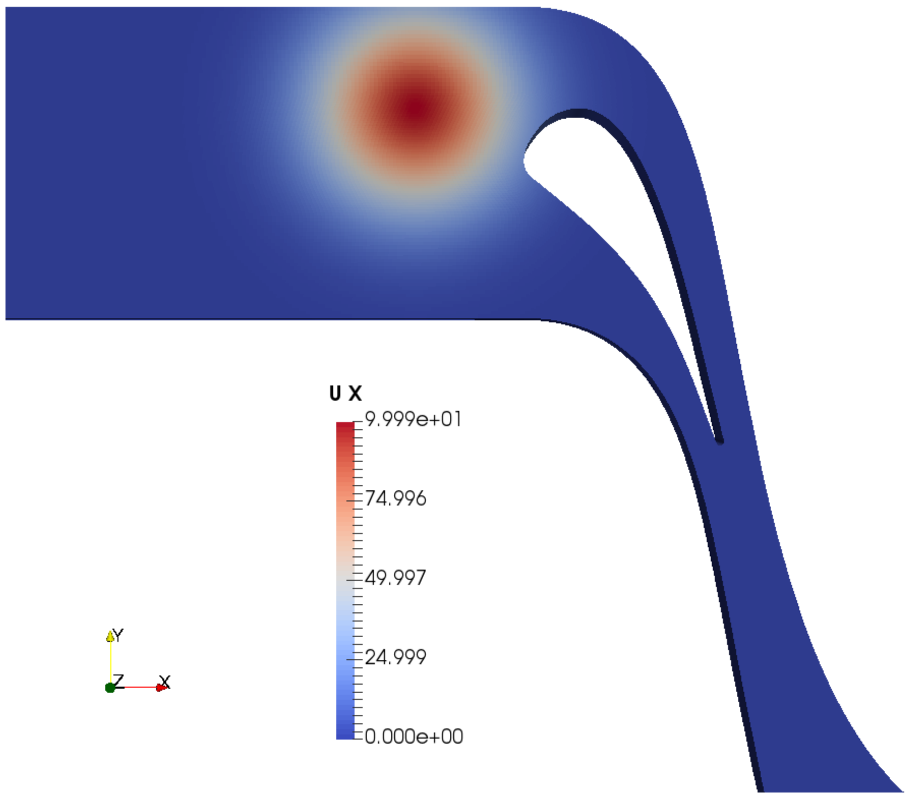


Figure 2-1: Visualization of the source term field, $s_{\rho u_1}$, for flow over a turbine vane using $\theta = 1$.

time unit. The magnitude of the adjoint field is large in the regions near the trailing edge and above the suction side of the turbine vane, which implies that the design objective is sensitive to perturbations in the flow equations in these regions.

Table 2.1: Comparison between gradient obtained from adjoint solution and finite difference gradient for the design objective

Length of time interval	Finite difference gradient	Adjoint gradient
0.2	-4.76×10^{-3}	-6.5×10^{-3}
2	-4.76×10^{-3}	5.3×10^{13}

Table 2.1 shows the gradient of the pressure loss coefficient with respect to θ, the scaling parameter of the Gaussian-shaped source term perturbation, obtained from the adjoint solutions. The short time adjoint solution provides a gradient with the right order of magnitude and sign, but the error with respect to the finite difference

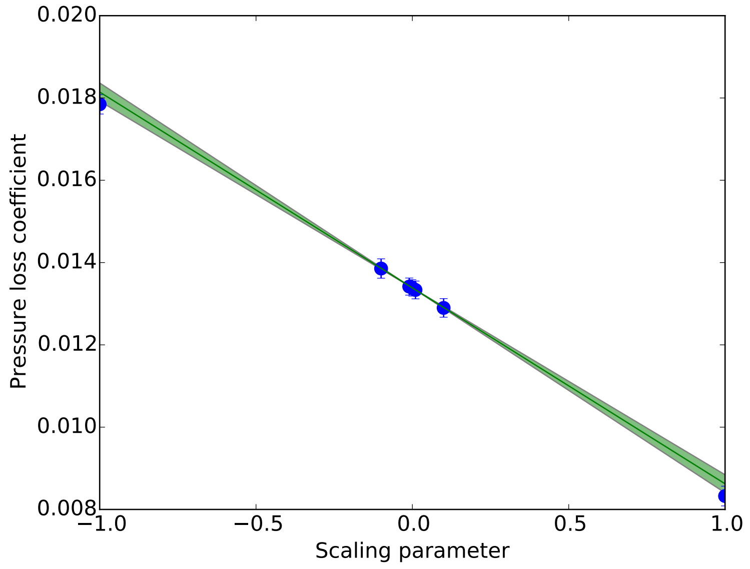


Figure 2-2: Time and mass-flow averaged pressure loss coefficient as a function of scaling parameter of the source term. Blue dots denote the mean pressure loss coefficient and the blue error bars signify one standard deviation away from the mean. The green shaded regions denotes the gradient estimated using linear regression and the associated uncertainty given by one standard deviation.

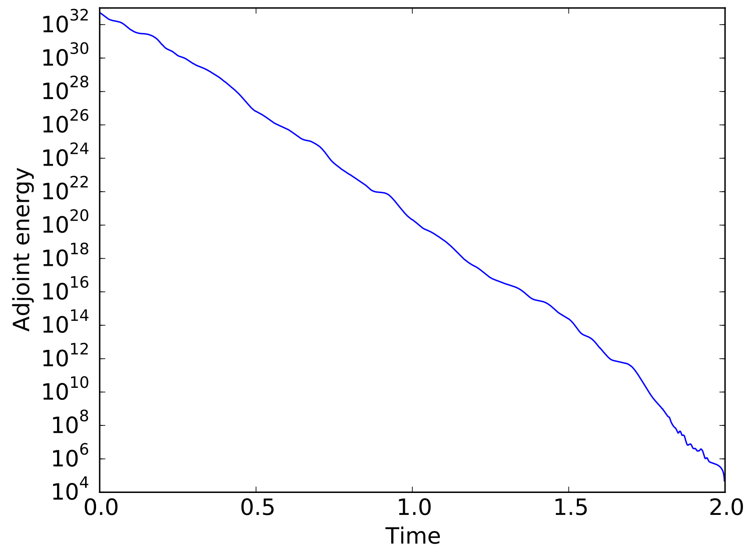


Figure 2-3: Adjoint energy (as defined by Equation 2.41) as a function of time (represented by time units) for the long time adjoint solution

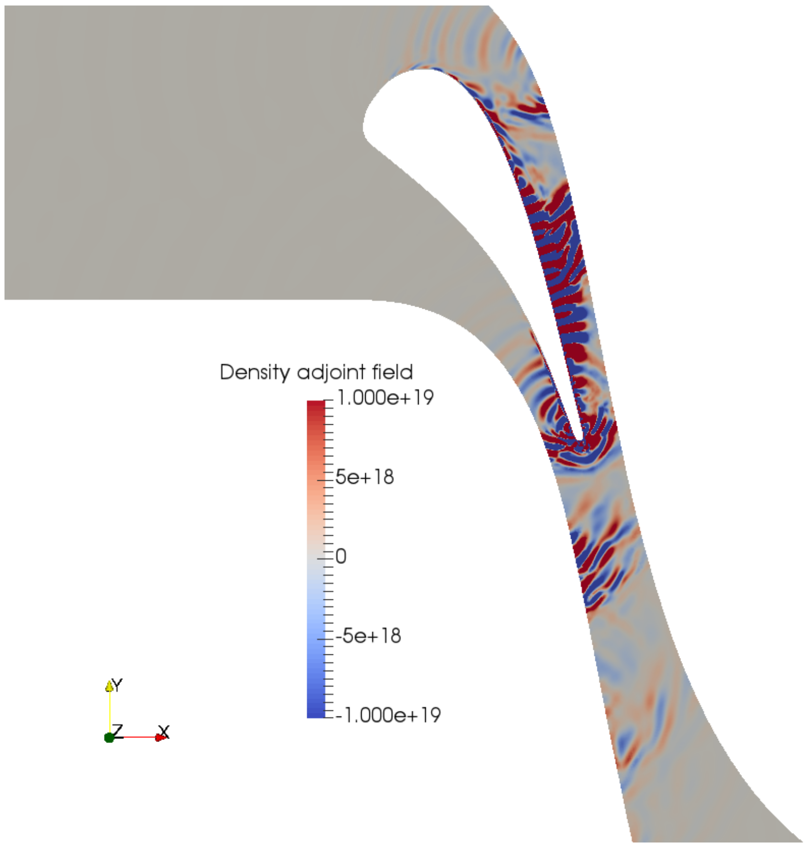


Figure 2-4: A visualization of the density adjoint field at $T = 1$ time unit

gradient is high (36%). In contrast, the large magnitudes of the long time adjoint solution corrupt the value of the gradient obtained from it. Hence, for long time averaged design objectives, the conventional adjoint method for unsteady flow problems fails.

2.2 Understanding mechanisms of divergence

In this thesis, stability analysis methods are used to analyze the adjoint equations for the compressible Navier-Stokes equations. The analysis provides an understanding of the mechanisms of divergence of adjoint solution by finding the regions in the domain of the flow problem that contribute most to the divergence.

2.2.1 Stability analysis methods

As the adjoint equations derived in this thesis are linear with time-dependent coefficients, conventional eigenvalue-based linear stability analysis methods cannot be used. In order to understand the mechanisms of divergence of the adjoint solution in turbulent fluid flows, nonlinear stability analysis methods are more helpful. Examples of such stability analysis methods are

1. Lyapunov analysis

2. Energy analysis

The Lyapunov analysis method for stability analysis involves computing the Lyapunov exponents and Lyapunov covariant vectors of the adjoint flow [56]. It is a global (spatially) stability analysis method and the exponents indicate the stability of the flow, whereas the covariant vectors provide the modes of divergence. A chaotic system can have multiple diverging modes or covariant vectors associated with positive exponents [23]. More information about Lyapunov exponents and covariant vectors can be found in Section 4.3. As analytical solutions for the compressible Navier-Stokes equations are generally not available, the exponents and covariant vectors have to be computed numerically.

In the energy analysis method for stability analysis, the time derivative of the adjoint energy is computed and the terms that increase the norm of the adjoint solution are analyzed [101]. It is a local (spatially) stability analysis method which means that the stability information provided by the analysis can be localized in the domain of the flow problem. Additionally, for the Navier-Stokes equations the analysis can be performed analytically. Because of this, the energy analysis method is substantially cheaper than Lyapunov analysis. Hence, energy analysis is the preferred stability analysis method for the adjoint equations.

2.2.2 Symmetrization of the Navier-Stokes equations

Performing energy analysis of the conservative form of the adjoint equations for the compressible Navier-Stokes equations does not lead to usable information about the regions of divergence of the adjoint solution [103]. It is more practical to perform the analysis on the symmetrized form [2, 108, 47] of the adjoint of the compressible Navier-Stokes equations. This form can be derived by first transforming the Navier-Stokes equations using a symmetrization procedure.

Symmetrization of the Navier-Stokes equations involves transforming the equations in such a way that the tensors in the symmetrized form of the equations corresponding to A_{ijk} and D_{ijkl} from Equation 1.21, are symmetric in the indices i and k. There are two types of symmetrization for the compressible Navier-Stokes in the fluid dynamics literature

1. Non-entropy based symmetrization

2. Entropy based symmetrization

In the non-entropy based symmetrization methods, the conservative form of compressible Navier-Stokes equations are transformed into the equations for the symmetrized variables by pre-multiplying the equations using a symmetrizing transformation. In contrast, in the entropy based symmetrization methods, the conservative form of the compressible Navier-Stokes equations are directly used, but are represented in the symmetrized variables using a symmetrizing transformation.

2.3 Non-entropy symmetrization

Non-entropy based symmetrization methods provide a symmetrizing transformation for the Euler equations [2, 108]. The Euler equations provide the governing equations for a compressible inviscid ideal fluid. The equations are represented in the following form using Equation 1.19

$$\frac{\partial w_i}{\partial t} + A_{ijk}\frac{\partial w_k}{\partial x_j} = s_i \tag{2.3}$$

Denoting the symmetrized variables by v_i, the linearized transformation from the conservative to symmetrized variables is given by, $\delta v_i = T_{ik}\delta w_k$. Rewriting the equations using the symmetrized variables

$$T_{ik}^{-1}\frac{\partial v_k}{\partial t} + A_{ijk}T_{km}^{-1}\frac{\partial v_m}{\partial x_j} = s_i \tag{2.4}$$

Pre-multiplying the equations by T_{ni}

$$\frac{\partial v_n}{\partial t} + T_{ni}A_{ijk}T_{km}^{-1}\frac{\partial v_m}{\partial x_j} = T_{ni}s_i \tag{2.5}$$

Representing the term $T_{ni}A_{ijk}T_{km}^{-1}$ as $\hat{A}_{njm}$ and $T_{ni}s_i$ as $\hat{s}_n$

$$\frac{\partial v_n}{\partial t} + \hat{A}_{njm}\frac{\partial v_m}{\partial x_j} = \hat{s}_n \tag{2.6}$$

The linear transformation, T_{ni} is chosen such that $\hat{A}_{njm}$ is symmetric in the indices n and m. The above equations are the symmetrized Euler equations.

2.3.1 Symmetrizing transformations

There are a number of symmetrizing transformations that produce the symmetrized equations derived in the previous section. A couple of these transformations from literature are described below.

Abarbanel [2] proposed a symmetrizing transformation for optimal time splitting of the Navier-Stokes equations and described a symmetrizing transformation from primitive $(\rho, \boldsymbol{u}, p)$ to symmetrized variables. Denoting the primitive variables by q_k, the linear transformation, $\mathbf{S}$, transforms the derivatives of the primitive (q_k) to derivatives of symmetrized variables (v_i).

$$\delta v_i = S_{ik}\delta q_k \tag{2.7}$$

Using the linear transformation, $\mathbf{V}$, which transforms derivatives of conservative (w_k)

to derivatives of primitive variables (q_i),

$$\delta q_i = V_{ik} \delta w_k \tag{2.8}$$

the corresponding symmetrizing transformation from the derivatives of conservative (w_k) to derivatives of symmetrized variables (v_i) can be constructed.

$$T_{ik} = S_{ij} V_{jk} \tag{2.9}$$

The tensor $\hat{A}_{ijk}$ and the matrices S_{ik} and V_{jk} for the Abarbanel symmetrizing transformation are provided in Appendix A.3. The derivatives of the symmetrized variables are given by

$$\delta \mathbf{v} = (\frac{c\delta\rho}{\sqrt{\gamma}\rho}, \delta u_1, \delta u_2, \delta u_3, \frac{\gamma\delta p - c^2\delta\rho}{\rho c\sqrt{\gamma(\gamma-1)}}) \tag{2.10}$$

Turkel [108] proposed an alternative symmetrizing transformation in which one of the symmetrized variables is the physical entropy variable. The tensor $\hat{A}_{ijk}$ and the matrices S_{ik} and T_{ik} for the Turkel symmetrizing transformation are provided in Appendix A.4. The derivatives of the symmetrized variables are given by

$$\delta \mathbf{v} = (\frac{\delta p}{\rho c}, \delta u_1, \delta u_2, \delta u_3, \delta p - c^2\delta\rho) \tag{2.11}$$

The Turkel and Abarbanel symmetrized variables are similar as can be seen from Equations 2.10 and 2.11. The middle terms of the vectors, from term 2 to 4, are velocity variables and are the same in both the equations. Term 5 of the two vectors look alike, but, it can be shown that only the term of the Abarbanel symmetrizing variables is proportional to δc. Term 1 for the two symmetrizing variables is quite different.

2.3.2 Adjoint of symmetrized equations

Applying the adjoint method to the symmetrized equations results in the symmetrized adjoint equations. The form of these equations is different from the adjoint equations for the conservative variables. The derivation process begins by forming the Lagrangian of the adjoint equations for the symmetrized variables

$$\begin{aligned}\delta\bar{J} = \frac{1}{T}\int_0^T \int_S \frac{\partial J}{\partial v_i}\delta v_i \, dS \, dt \\ + \int_0^T \int_V \hat{v}_i\left(\frac{\partial \delta v_i}{\partial t} + \hat{A}_{ijk}\frac{\partial \delta v_k}{\partial x_j} + \delta\hat{A}_{ijk}\frac{\partial v_k}{\partial x_j} - \delta\hat{s}_i\right) dV \, dt\end{aligned} \tag{2.12}$$

Representing the term $\frac{\partial \hat{A}_{ijk}}{\partial v_l}$ as B_{ijkl} and integrating by parts in time and space

$$\begin{aligned}\delta\bar{J} =& \frac{1}{T}\int_0^T \int_S \frac{\partial J}{\partial v_i}\delta v_i \, dS \, dt \\ &+ \int_V (\hat{v}_{i|T}\delta v_{i|T} - \hat{v}_{i|0}\delta v_{i|0}) \, dV - \int_0^T \int_V \frac{\partial \hat{v}_i}{\partial t}\delta v_i \, dV \, dt \\ &+ \int_0^T \int_S \hat{v}_i\hat{A}_{ijk}\delta v_k n_j \, dS - \int_0^T \int_V \frac{\partial(\hat{v}_i\hat{A}_{ijk})}{\partial x_j}\delta v_k \, dV \, dt \\ &+ \int_0^T \int_V \hat{v}_i B_{ijkl}\delta v_l \frac{\partial v_k}{\partial x_j} \, dV \, dt \\ &- \int_0^T \int_V \hat{v}_i \delta\hat{s}_i \, dV \, dt\end{aligned} \tag{2.13}$$

After simplification, the above equation can be rewritten as

$$\delta\bar{J} = -\int_0^T \int_V \hat{v}_i \delta\hat{s}_i \, dV \, dt \tag{2.14}$$

provided the following symmetrized adjoint equations are satisfied

$$-\frac{\partial \hat{v}_i}{\partial t} - \hat{A}_{kji}\frac{\partial \hat{v}_k}{\partial x_j} - (B_{kjil} - B_{kjli})\frac{\partial v_l}{\partial x_j}\hat{v}_k = 0 \tag{2.15}$$

with corresponding boundary conditions for the adjoint solution. The primary difference of the above adjoint equations for symmetrized variables from the adjoint equations for conservative variables is the existence of a source term linear in $\hat{v}_k$, which is the last term in the left hand side of the above equation. For non-entropy based symmetrization methods, the adjoint solution for symmetrized variables is given by the transpose of the symmetrizing transformation applied to the adjoint solution for conservative variables, as derived in Section 2.3.4.

2.3.3 Adjoint energy analysis

Energy analysis of the symmetrized adjoint equations is performed by analyzing the time derivative of the L_2 norm of the symmetrized adjoint solution. The L_2 norm is defined as,

$$E_{\hat{\mathbf{v}}} = \|\hat{\mathbf{v}}\|_2 = (\int_V \hat{v}_p N_{pq} \hat{v}_q \, dV)^{\frac{1}{2}} \tag{2.16}$$

where $\mathbf{N}$ is a constant diagonal matrix for $\hat{\mathbf{v}}$ that ensures dimensional consistency of the above expression. The expression for $\mathbf{N}$ depends on the symmetrizing transformation. In order to obtain the rate of growth of the adjoint solution, the square of the L_2 norm divided by 2 is differentiated with respect to negative time.

$$\begin{aligned} -\frac{1}{2}\frac{d(E_{\hat{\mathbf{v}}}^2)}{dt} = -E_{\hat{\mathbf{v}}}\frac{dE_{\hat{\mathbf{v}}}}{dt} &= -\frac{1}{2}\frac{d}{dt}(\int_V \hat{v}_p N_{pq} \hat{v}_q) \, dV \\ &= -\frac{1}{2}(\int_V N_{pq}(\frac{\partial \hat{v}_p}{\partial t} v_q + \frac{\partial \hat{v}_q}{\partial t} v_p) \, dV) \end{aligned} \tag{2.17}$$

The minus sign exists because of the fact that the magnitude of the adjoint energy grows exponentially in negative time as the adjoint equations are solved backwards in time. If the term N_{pq} is symmetric in the indices p and q, then $N_{pq}\frac{\partial \hat{v}_p}{\partial t} v_q = N_{pq}\frac{\partial \hat{v}_q}{\partial t} v_p$. Hence,

$$-E_{\hat{\mathbf{v}}}\frac{dE_{\hat{\mathbf{v}}}}{dt} = -\int_V N_{pq}\frac{\partial \hat{v}_p}{\partial t}\hat{v}_q \, dV \tag{2.18}$$

Substituting Equation 2.15 into the above equation

$$-E_{\hat{\mathbf{v}}}\frac{dE_{\hat{\mathbf{v}}}}{dt} = \int_V N_{pq}\hat{v}_q[\hat{A}_{kjp}\frac{\partial \hat{v}_k}{\partial x_j} + (B_{kjpl} - B_{kjlp})\frac{\partial v_l}{\partial x_j}\hat{v}_k] \, dV \tag{2.19}$$

Expanding the term $B_{kjpl} = \frac{\partial \hat{A}_{kjp}}{\partial v_l}$ and using the Gauss divergence theorem, the integral over the first term in the square bracket in the above equation can be rewritten as

$$\begin{aligned}\int_V N_{pq}\hat{v}_q\hat{A}_{kjp}\frac{\partial \hat{v}_k}{\partial x_j}\,dV = \int_S & N_{pq}\hat{v}_q\hat{A}_{kjp}\hat{v}_k n_j\,dS \\ & - \int_V N_{pq}\frac{\partial \hat{v}_q}{\partial x_j}\hat{A}_{kjp}\hat{v}_k\,dV - \int_V N_{pq}\hat{v}_q B_{kjpl}\frac{\partial v_l}{\partial x_j}\hat{v}_k\,dV\end{aligned} \tag{2.20}$$

where $S = \partial V$. Using symmetry of $\hat{A}_{kjp}$ in p and k, the above equation can be rewritten as

$$\int_V N_{pq}\hat{v}_q\hat{A}_{kjp}\frac{\partial \hat{v}_k}{\partial x_j}\,dV = \frac{1}{2}(\int_S N_{pq}\hat{v}_q\hat{A}_{kjp}\hat{v}_k n_j\,dS - \int_V N_{pq}\hat{v}_q B_{kjpl}\frac{\partial v_l}{\partial x_j}\hat{v}_k\,dV) \tag{2.21}$$

Symmetrization of the Euler equations ensures that the contribution of the convective term in the adjoint energy analysis can be written in the form of a spatial integral over a quadratic product. Analysis of the quadratic product provides information about the regions of divergence of the adjoint solution, the details of which are discussed in Section 2.6. Combining Equation 2.21 with 2.19, the rate of growth of adjoint energy equation is

$$-E_{\hat{\mathbf{v}}}\frac{dE_{\hat{\mathbf{v}}}}{dt} = \int_V \hat{v}_q[N_{pq}(\frac{B_{kjpl}}{2} - B_{kjlp})\frac{\partial v_l}{\partial x_j}]\hat{v}_k\,dV + \frac{1}{2}\int_S N_{pq}\hat{v}_q\hat{A}_{kjp}\hat{v}_k n_j\,dS \tag{2.22}$$

The first term in the right hand side in the above equation is the contribution due to the convective term and it can be written in the following quadratic form

$$\begin{gathered}\int_V \hat{v}_q M_{qk}\hat{v}_k\,dV \\ M_{qk} = N_{pq}(\frac{B_{kjpl}}{2} - B_{kjlp})\frac{\partial v_l}{\partial x_j}\end{gathered} \tag{2.23}$$

Denoting the terms in the round bracket in the above equation as $\mathbf{M_1}$ and $\mathbf{M_2}$, such that $\mathbf{M} = \mathbf{M_1} - \mathbf{M_2}$,

$$\begin{aligned}M_{1,qk} &= N_{pq}\frac{B_{kjpl}}{2}\frac{\partial v_l}{\partial x_j} = \frac{1}{2}\frac{\partial \hat{A}_{kjp}}{\partial q_l}\frac{\partial q_l}{\partial x_j}N_{pq}, \\ M_{2,qk} &= N_{pq}B_{kjlp}\frac{\partial v_l}{\partial x_j} = N_{pq}\frac{\partial \hat{A}_{kjl}}{\partial q_m}\frac{\partial q_m}{\partial v_p}\frac{\partial v_l}{\partial x_j} = \frac{\partial \hat{A}_{kjl}}{\partial q_m}S^{-1}_{mp}\frac{\partial v_l}{\partial x_j}N_{pq}\end{aligned} \tag{2.24}$$

For the Abarbanel symmetrizing transformation, the matrices $\mathbf{M_1}$, $\mathbf{M_2}$ and $\mathbf{N}$ are given below. Using $b = \frac{c}{\sqrt{\gamma}}, a = \sqrt{\frac{\gamma-1}{\gamma}}c$

$$\mathbf{M} = \mathbf{M_1} - \mathbf{M_2}$$

$$\mathbf{M_1} = \frac{1}{2}\begin{pmatrix} \nabla\cdot u & \nabla_1 b & \nabla_2 b & \nabla_3 b & 0 \\ \nabla_1 b & \nabla\cdot u & 0 & 0 & \nabla_1 a \\ \nabla_2 b & 0 & \nabla\cdot u & 0 & \nabla_2 a \\ \nabla_3 b & 0 & 0 & \nabla\cdot u & \nabla_3 a \\ 0 & \nabla_1 a & \nabla_2 a & \nabla_3 a & \nabla\cdot u \end{pmatrix}$$

$$\mathbf{M_2} = \begin{pmatrix} 0 & \frac{b}{\rho}\nabla_1\rho & \frac{b}{\rho}\nabla_2\rho & \frac{b}{\rho}\nabla_3\rho & \frac{\sqrt{\gamma-1}}{2}\nabla\cdot u \\ 0 & \nabla_1 u_1 & \nabla_2 u_1 & \nabla_3 u_1 & \frac{a}{2p}\nabla_1 p \\ 0 & \nabla_1 u_2 & \nabla_2 u_2 & \nabla_3 u_2 & \frac{a}{2p}\nabla_2 p \\ 0 & \nabla_1 u_3 & \nabla_2 u_3 & \nabla_3 u_3 & \frac{a}{2p}\nabla_3 p \\ 0 & \frac{2}{\gamma-1}\nabla_1 a & \frac{2}{\gamma-1}\nabla_2 a & \frac{2}{\gamma-1}\nabla_3 a & \frac{\gamma-1}{2}\nabla\cdot u \end{pmatrix} \tag{2.25}$$

$$\mathbf{N} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

For the Turkel symmetrizing transformation, the matrices $\mathbf{M_1}, \mathbf{M_2}$ and $\mathbf{N}$ are given below.

$$\mathbf{M} = \mathbf{M_1} - \mathbf{M_2}$$

$$\mathbf{M_1} = \frac{1}{2}\begin{pmatrix} \nabla\cdot u & \nabla_1 c & \nabla_2 c & \nabla_3 c & 0 \\ \nabla_1 c & \nabla\cdot u & 0 & 0 & 0 \\ \nabla_2 c & 0 & \nabla\cdot u & 0 & 0 \\ \nabla_3 c & 0 & 0 & \nabla\cdot u & 0 \\ 0 & 0 & 0 & 0 & T_r^2\nabla\cdot u \end{pmatrix}$$

$$\mathbf{M_2} = \begin{pmatrix} \frac{\gamma-1}{2}\nabla\cdot u & \frac{1}{\rho c}\nabla_1 p & \frac{1}{\rho c}\nabla_2 p & \frac{1}{\rho c}\nabla_3 p & \frac{T_r^2}{2\rho c}\nabla\cdot u \\ \frac{\gamma-1}{2\rho c}\nabla_1 p & \nabla_1 u_1 & \nabla_2 u_1 & \nabla_3 u_1 & \frac{T_r^2}{2\gamma p\rho}\nabla_1 p \\ \frac{\gamma-1}{2\rho c}\nabla_2 p & \nabla_1 u_2 & \nabla_2 u_2 & \nabla_3 u_2 & \frac{T_r^2}{2\gamma p\rho}\nabla_2 p \\ \frac{\gamma-1}{2\rho c}\nabla_3 p & \nabla_1 u_3 & \nabla_2 u_3 & \nabla_3 u_3 & \frac{T_r^2}{2\gamma p\rho}\nabla_3 p \\ 0 & (\nabla_1 p - c^2\nabla_1\rho) & (\nabla_2 p - c^2\nabla_2\rho) & (\nabla_3 p - c^2\nabla_3\rho) & 0 \end{pmatrix}$$

$$\mathbf{N} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & T_r^2 \end{pmatrix} \tag{2.26}$$

where T_r is a dimensional constant quantity which can be non-dimensionalized using

$$T_r = C_t\frac{p_r}{u_r} \tag{2.27}$$

where C_t is a non-dimensional tunable factor that is in the range $(0, \infty)$ and u_r, p_r are non-dimensionalizing reference quantities of the flow, whose values are defined in Section 1.4. The Turkel symmetrizing transformation matrix $\mathbf{N}$ has one degree of freedom as the value for C_t needs to be set. By default, $C_t = 1$, unless otherwise specified. If required, the value of C_t can be determined by visually evaluating the divergence indicator fields (derived from the adjoint energy analysis in Section 2.6) for different values of C_t.

2.3.4 Relationship between symmetrized and conservative adjoint solutions

The symmetrized and conservative adjoint solutions are related by a linear transformation. This can be shown by knowing that the sensitivity due to a perturbation can be computed from either the conservative adjoint solution or symmetrized adjoint solution, from Equations 1.23 and 2.14

$$\begin{aligned}\delta\bar{J} &= -\int_0^T \int_V \hat{w}_i \delta s_i \, dV \, dt \\ &= -\int_0^T \int_V \hat{v}_i \delta \hat{s}_i \, dV \, dt = -\int_0^T \int_V \hat{v}_i T_{ik} \delta s_k \, dV \, dt\end{aligned} \tag{2.28}$$

Hence, equating the terms inside the integrals,

$$\hat{w}_i = T_{ki} \hat{v}_k \tag{2.29}$$

Applying the Cauchy-Schwarz inequality,

$$\|\hat{\mathbf{w}}\|_2 \leq \|\mathbf{T}^T\|_2 \|\hat{\mathbf{v}}\|_2 \tag{2.30}$$

As the components of the transformation matrix ($\mathbf{T}$) consist of bounded scalar fields, its matrix L_2 norm is also bounded. This implies that if the spatial L_2 norm of the symmetrized adjoint solution is bounded, then the spatial L_2 norm of the conservative adjoint field solution is also bounded. The vice versa can be shown to be true.

2.4 Entropy symmetrization

The Entropy-based symmetrization methods provide a symmetrizing transformation for the compressible Navier-Stokes equations and not just the Euler equations. In addition, the symmetrized equations formed from the entropy-based symmetrizing transformations have a physical meaning. The entropy (or symmetrized) variables satisfy the Clausius-Duhem inequality [47]. The linearized relation between the derivatives of entropy variables and the derivatives of conservative variables is $\delta w_i = A^0_{ij}\delta v_j$, where A^0_{ij} forms a symmetric positive definite matrix. Rewriting the compressible Navier-Stokes equations in terms of the symmetrized variables

$$A^0_{ij}\frac{\partial v_j}{\partial t} + A_{ijk}A^0_{km}\frac{\partial v_m}{\partial x_j} = \frac{\partial}{\partial x_j}(D_{ijkl}A^0_{km}\frac{\partial v_m}{\partial x_l}) + s_i \tag{2.31}$$

Denoting the term $A_{ijk}A^0_{km}$ as $\hat{A}_{ijm}$ and $D_{ijkl}A^0_{km}$ as $\hat{D}_{ijml}$

$$A^0_{ij}\frac{\partial v_j}{\partial t} + \hat{A}_{ijm}\frac{\partial v_m}{\partial x_j} = \frac{\partial}{\partial x_j}(\hat{D}_{ijml}\frac{\partial v_m}{\partial x_l}) + s_i \tag{2.32}$$

The tensor $\hat{A}_{ijm}$ is symmetric in the indices i and m and $\hat{D}_{ijml}$ is symmetric in the following sense, $\hat{D}_{ijml} = \hat{D}_{mlij}$. Additionally, the matrix $\hat{\mathbf{K}}$ formed by arranging the matrices $\hat{D}_{:j:l}$ (for fixed indices j and l) in the following manner,

$$\hat{\mathbf{K}} = \begin{pmatrix} \hat{D}_{:1:1} & \hat{D}_{:1:2} & \hat{D}_{:1:3} \\ \hat{D}_{:2:1} & \hat{D}_{:2:2} & \hat{D}_{:2:3} \\ \hat{D}_{:3:1} & \hat{D}_{:3:2} & \hat{D}_{:3:3} \end{pmatrix} \tag{2.33}$$

is symmetric positive semi-definite. The matrix $\hat{\mathbf{K}}$ can also be written as,

$$\hat{K}_{[3(j-1)+i][3(l-1)+m]} = \hat{D}_{ijml} \tag{2.34}$$

where [] is used to denote separation between indices.

2.4.1 Symmetrizing transformations

Hughes [47] proposed the following symmetrizing transformation, $A^0_{ij} = \frac{\partial w_i}{\partial v_j}$

$$\mathbf{A^0} = \frac{1}{\gamma - 1} \begin{pmatrix} \rho & \rho u_1 & \rho u_2 & \rho u_3 & \rho E \\ \rho u_1 & \rho u_1 u_1 + p & \rho u_1 u_2 & \rho u_1 u_3 & \rho H u_1 \\ \rho u_2 & \rho u_2 u_1 & \rho u_2 u_2 + p & \rho u_2 u_3 & \rho H u_2 \\ \rho u_3 & \rho u_3 u_1 & \rho u_3 u_2 & \rho u_3 u_3 + p & \rho H u_3 \\ \rho E & \rho H u_1 & \rho H u_2 & \rho H u_3 & \rho H^2 - \frac{\gamma p^2}{\rho(\gamma-1)} \end{pmatrix} \tag{2.35}$$

where the symmetrized variables can be written in terms of the conservative variables in the following form

$$\mathbf{v} = \frac{1}{\rho e} \begin{pmatrix} -\rho E + \rho e(\gamma + 1 - s) \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ -\rho \end{pmatrix} \tag{2.36}$$

where the entropy $s = ln[\frac{\beta(\gamma-1)\rho e}{\rho^\gamma}]$, β is a non-dimensionalizing constant given by $\frac{\rho_r^\gamma}{p_r}$. The symmetrized tensors $\hat{A}_{ijm}$ and $\hat{D}_{ijml}$ are given in Appendix A.5.

2.4.2 Nonlinear stability

The entropy variables satisfy the following entropy production inequality

$$\int_V \left(\frac{\partial \rho s}{\partial t} + \frac{\partial \rho s u_i}{\partial x_i} + \frac{\partial}{\partial x_i}\left(\frac{\alpha \rho \gamma}{e}\frac{\partial e}{\partial x_i}\right)\right) dV = \int_V \frac{\partial v_i}{\partial x_j} \hat{D}_{ijkl} \frac{\partial v_k}{\partial x_l}\, dV \geq 0 \tag{2.37}$$

If the right hand side of the above equation (the energy dissipation rate) is bounded from above, then the entropy solution of the Navier-Stokes equations is nonlinearly stable. Such a stability result cannot be obtained for the linearized tangent and adjoint equations of the symmetrized form of compressible Navier-Stokes equations. In fact, the energy analysis of the adjoint equations shows that there are terms that grow fast and contribute to the exponential divergence of the adjoint solution.

2.4.3 Adjoint of symmetrized equations

As the symmetrized equations are the same equations as the compressible Navier-Stokes equations, but written in a different form, the adjoint method applied to the symmetrized equations also results in the same equations as the adjoint equations for the conservative variables, but written in a different form. This implies that $\hat{\mathbf{v}} = \hat{\mathbf{w}}$, which means that the symmetrized adjoint solution is the same as the conservative adjoint solution. Starting from Equation 1.18, rewriting the viscous term in terms of the symmetrized variables and replacing δw_i with $A^0_{in}\delta v_n$ in the first volume and time integral term

$$
\begin{aligned}
\delta\bar{J} =& \frac{1}{T}\int_0^T \int_S [(\frac{\partial J}{\partial w_i} + \hat{v}_k A_{kji} n_j - \hat{v}_k \frac{\partial D_{kjml}}{\partial w_i}\frac{\partial w_m}{\partial x_l} n_j)\delta w_i - \hat{v}_i D_{ijkl}\delta\frac{\partial w_k}{\partial x_l} n_j]\, dS\, dt \\
&+ \int_V (\hat{v}_{i|T}\delta w_{i|T} - \hat{v}_{i|0}\delta w_{i|0}) dV \\
&- \int_0^T \int_V (\frac{\partial \hat{w}_i}{\partial t} + \frac{\partial \hat{v}_k}{\partial x_j} A_{kji}) A^0_{in}\delta v_n \, dV\, dt \\
&+ \int_0^T \int_V (\frac{\partial \hat{v}_i}{\partial x_j}\hat{D}_{ijkl}\frac{\partial \delta v_k}{\partial x_l} + \frac{\partial \hat{v}_i}{\partial x_j}\frac{\partial \hat{D}_{ijkl}}{\partial v_m}\frac{\partial v_k}{\partial x_l}\delta v_m)\, dV\, dt \\
&- \int_0^T \int_V \hat{v}_i \delta s_i \, dV\, dt
\end{aligned}
\tag{2.38}
$$

After applying integration by parts to the first term the viscous term integral and grouping all the δv_n terms, the adjoint equations are

$$
-A^0_{ij}\frac{\partial \hat{v}_j}{\partial t} - (\hat{A}_{kji} - \hat{A}^v_{kji})\frac{\partial \hat{v}_k}{\partial x_j} = \frac{\partial}{\partial x_l}(\hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j}) \tag{2.39}
$$

with appropriate boundary and terminal conditions, where $\hat{A}^v_{kji} = \frac{\partial \hat{D}_{kjml}}{\partial v_i}\frac{\partial v_m}{\partial x_l}$. For the Hughes symmetrizing transformation, $\hat{A}^v_{kji}$ is not symmetric in the indices i and k. This can be seen by computing the terms of the matrix for $i = 1$ and observing that

$$
\hat{A}^v_{312} = \frac{\mu}{v_5^2}\frac{\partial v_5}{\partial x_2}, \quad \hat{A}^v_{213} = -\frac{2}{3}\frac{\mu}{v_5^2}\frac{\partial v_5}{\partial x_2} \tag{2.40}
$$

It can be clearly seen that these two terms are not equal. Consequently, the matrix formed by $\hat{A}^v_{:1:}$ is not symmetric. Similarly, the matrices formed by $\hat{A}^v_{:2:}$ and $\hat{A}^v_{:3:}$ are not symmetric.

2.4.4 Adjoint energy analysis

Energy analysis of the symmetrized adjoint equations is performed by analyzing the time derivative of a weighted L_2 norm of the adjoint solution. The L_2 norm is defined as,

$$E_{\hat{\mathbf{v}}} = \|\hat{\mathbf{v}}\|_2 = (\int_V \hat{v}_p N_{pq} \hat{v}_q \, dV)^{\frac{1}{2}} \tag{2.41}$$

where $\mathbf{N} = \mathbf{A^0}$. The norm is valid as $\mathbf{A^0}$ is symmetric positive definite and the quadratic product is dimensionally consistent.

In order to obtain the rate of growth of the adjoint solution, the square of the L_2 norm divided by 2 is differentiated with respect to negative time. Using the fact that A^0_{ij} is symmetric

$$-\frac{1}{2}\frac{d(E^2_{\hat{\mathbf{v}}})}{dt} = -E_{\hat{\mathbf{v}}}\frac{dE_{\hat{\mathbf{v}}}}{dt} = -\frac{1}{2}\int_V (2\hat{v}_i A^0_{ij}\frac{\partial \hat{v}_j}{\partial t} + \hat{v}_i \frac{\partial A^0_{ij}}{\partial t}\hat{v}_j) \, dV \tag{2.42}$$

From Equation 2.39

$$-A^0_{ij}\frac{\partial \hat{v}_j}{\partial t} = (\hat{A}_{kji} - \hat{A}^v_{kji})\frac{\partial \hat{v}_k}{\partial x_j} + \frac{\partial}{\partial x_l}(\hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j}) \tag{2.43}$$

Using symmetry of $\hat{A}_{kji}$ in the indices k and i and integrating by parts in space it can be shown that

$$\int_V \hat{v}_i \hat{A}_{kji}\frac{\partial \hat{v}_k}{\partial x_j} \, dV = \frac{1}{2}(\int_S \hat{v}_i \hat{A}_{kji}\hat{v}_k \hat{n}_j \, dS - \int_V \hat{v}_i \frac{\partial \hat{A}_{kji}}{\partial x_j}\hat{v}_k \, dV) \tag{2.44}$$

The spatial integral of the viscous term from Equation 2.43 can be rewritten as

$$\int_V \hat{v}_i \frac{\partial}{\partial x_l}(\hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j}) \, dV = \int_S \hat{v}_i \hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j}\hat{n}_l \, dS - \int_V \frac{\partial \hat{v}_i}{\partial x_l}\hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j} \, dV \tag{2.45}$$

Expanding the time derivative, $\frac{\partial A^0_{ij}}{\partial t}$, from Equation 2.42 in terms of the time derivative of the conservative solution $\mathbf{w}$

$$-\frac{\partial A^0_{ij}}{\partial t} = -\frac{\partial A^0_{ij}}{\partial w_k}\frac{\partial w_k}{\partial t} = \frac{\partial A^0_{ij}}{\partial w_k}[\frac{\partial}{\partial x_m}(F_{km} - F^v_{km}) - s_k] \tag{2.46}$$

where F_{km} and F^v_{km} are defined in Equation 1.12. Substituting Equations 2.43, 2.44, 2.45 and 2.46 into Equation 2.42 and rearranging terms

$$\begin{aligned}
-E_{\hat{\mathbf{v}}}\frac{dE_{\hat{\mathbf{v}}}}{dt} = & \frac{1}{2}\int_V \hat{v}_i[-\frac{\partial \hat{A}_{kji}}{\partial x_j} + \frac{\partial A^0_{ij}}{\partial w_n}\frac{\partial F_{nm}}{\partial x_m}]\hat{v}_k \, dV \\
& - \int_V \hat{v}_i \hat{A}^v_{kji}\frac{\partial \hat{v}_k}{\partial x_j}\, dV - \frac{1}{2}\int_V \hat{v}_i \frac{\partial A^0_{ij}}{\partial w_n}(\frac{\partial F^v_{nm}}{\partial x_m} + s_n)\hat{v}_k \, dV \\
& - \int_V \frac{\partial \hat{v}_i}{\partial x_l}\hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j}\, dV \\
& + \frac{1}{2}\int_S \hat{v}_i \hat{A}_{kji}\hat{v}_k \hat{n}_j \, dS \\
& + \int_S \hat{v}_i \hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j}\hat{n}_l \, dS
\end{aligned} \tag{2.47}$$

The first spatial integral term in the right hand side in the above equation is a quadratic term in $\hat{\mathbf{v}}$. The terms inside the square bracket can be rewritten as $\mathbf{M} = -\mathbf{M_1} + \mathbf{M_2}$, where

$$M_{1,ik} = \frac{\partial \hat{A}_{kji}}{\partial x_j} \tag{2.48}$$

$$M_{2,ik} = \frac{\partial A^0_{ik}}{\partial w_n}\frac{\partial F_{nm}}{\partial x_m} \tag{2.49}$$

Unlike in the case of non-entropy symmetrizing transformations, the expressions for $\mathbf{M_1}$ and $\mathbf{M_2}$ for entropy symmetrizing transformations cannot be written in a compact analytical form of the flow variables. They can be computed by differentiating the tensors $\hat{\boldsymbol{A}}$, $\boldsymbol{A^0}$ and $\boldsymbol{F}$ with respect to either $\boldsymbol{x}$ or $\boldsymbol{w}$ using automatic differentiation and then numerically computing the result, in accordance with Equations 2.48 and 2.49. The equations relevant to the Hughes symmetrizing transformation are summarized below

$$\mathbf{M} = -\mathbf{M_1} + \mathbf{M_2}$$

$$\mathbf{N} = \frac{1}{\gamma - 1}\begin{pmatrix} \rho & \rho u_1 & \rho u_2 & \rho u_3 & \rho E \\ \rho u_1 & \rho u_1 u_1 + p & \rho u_1 u_2 & \rho u_1 u_3 & \rho H u_1 \\ \rho u_2 & \rho u_2 u_1 & \rho u_2 u_2 + p & \rho u_2 u_3 & \rho H u_2 \\ \rho u_3 & \rho u_3 u_1 & \rho u_3 u_2 & \rho u_3 u_3 + p & \rho H u_3 \\ \rho E & \rho H u_1 & \rho H u_2 & \rho H u_3 & \rho H^2 - \frac{\gamma p^2}{\rho(\gamma-1)} \end{pmatrix} \tag{2.50}$$

2.5 Contribution of terms in adjoint energy growth

In order to find the regions of the flow that cause divergence of the adjoint solution, the contribution of various terms in Equation 2.47 in increasing the adjoint energy needs to be analyzed. The energy analysis of the symmetrized adjoint solutions from non-entropy and entropy symmetrizations share a few important characteristics. Compared to the energy analysis for the non-entropy symmetrizations, there are a few additional spatial integral viscous terms in the analysis for entropy symmetrization that can contribute to the growth or decay of adjoint energy. Each term of Equation 2.47 along with corresponding terms of Equation 2.22 are analyzed separately below.

1. $\displaystyle\int_V \hat{v}_i M_{ik} \hat{v}_k \, dV$

   The first term in the right hand side of each of the two equations, Equation 2.22 and 2.47, are spatial integrals of quadratic products of $\hat{\mathbf{v}}$ scaled by the appropriate matrix $\mathbf{M}$. These terms provide the contribution of the convective terms of the Euler (for non-entropy symmetrization) or Navier-Stokes equations (for entropy symmetrization) to the respective formulas for the adjoint energy. In addition, owing to their quadratic product form, they are among the primary growth terms responsible for the divergence of the adjoint solution in turbulent fluid flows.

2. $\displaystyle\int_S \hat{v}_i \hat{A}_{kji} \hat{v}_k \hat{n}_j \, dS$

The second term in the right hand side of Equation 2.22 and the second last term in the right hand side of Equation 2.47 are quadratic product terms on the boundary domain of the flow problem. As these terms are quadratic, they can potentially contribute to the growth of the adjoint energy. But, in many compressible fluid problems, the inlet and outlet boundaries of the domain utilize characteristic boundary conditions [83, 105, 28] based on the Euler equations. Denoting the characteristic variables by z_i and the corresponding characteristic adjoint variables by $\hat{z}_i$, the boundary condition for the symmetrized adjoint equations on the inlet and outlet can be written as

$$\hat{v}_k \hat{A}_{kji} n_j \delta v_i = \hat{z}_k \Lambda_{li} \delta z_i = 0 \tag{2.51}$$

on ignoring the contribution of viscous and objective source terms and using the eigendecomposition, $\hat{A}_{kji} n_j = Q_{kl} \Lambda_{lm} Q_{im}$ with the identities, $\delta v_i = Q_{ki} \delta z_i$ and $\hat{z}_i = Q_{ki} \hat{v}_k$. On the inlet of the flow problem, the characteristic variables coming into the domain are prescribed as boundary conditions. This implies that $\delta z_i = 0$, for all i for which the i^{th} characteristic (or eigenvalues in the eigendecomposition) is negative. Consequently, $\hat{z}_j = 0$, for all j for which the j^{th} characteristic is positive, in order to ensure that the adjoint boundary condition is satisfied. Hence, $\hat{\mathbf{v}}$ belongs to the negative eigenspace of the matrix $\hat{A}_{kji} n_j$. This ensures that the convective quadratic product term on the inlet boundary in the adjoint energy analysis is always negative and thus, cannot contribute to the growth of adjoint energy. Similarly, on the outlet boundary of the flow problem, the outgoing characteristics are prescribed as boundary conditions and hence, it can be shown that the outlet boundary too does not contribute to the growth of adjoint energy.

3. $-\int_V \frac{\partial \hat{v}_i}{\partial x_l} \hat{D}_{kjil} \frac{\partial \hat{v}_k}{\partial x_j} \, dV$

   The fourth term in the right hand side of Equation 2.47 is a spatial integral over a quadratic product of the gradient of the symmetrized adjoint solution, scaled by the tensor $\hat{\mathbf{D}}$. The quadratic product can be written as $\nabla \mathbf{v}^{\mathbf{T}} \hat{\mathbf{K}} \nabla \mathbf{v}$, where $\nabla \mathbf{v}$ is a flattened vector that can be described in the following form

$$(\nabla v)_{[3(j-1)+i]} = \frac{\partial v_i}{\partial x_j} \tag{2.52}$$

The product is always positive or greater than zero as the matrix $\hat{\mathbf{K}}$ is positive semi-definite. Hence, as the sign in front of the term is negative, this spatial integral term never increases the adjoint energy.

4. $-\int_V \hat{v}_i \hat{A}^v_{kji} \frac{\partial \hat{v}_k}{\partial x_j}\, dV$ and $-\int_V \hat{v}_i \frac{\partial A^0_{ij}}{\partial w_n}(\frac{\partial F^v_{nm}}{\partial x_m} + s_n)\hat{v}_k\, dV$

   The second term and third terms in the right hand side of Equation 2.47 are spatial integrals that can contribute to the growth of adjoint energy. The first term in the above two terms cannot be converted into a quadratic form as the tensor $\hat{A}^v_{kji}$ is not symmetric in the indices k and i. This complicates the analysis of this term. Additionally, in the absence of any viscous effects in the fluid ($\mu = 0, \alpha = 0$) or source terms, the two terms do not provide any information about the divergence of adjoint energy. Hence, for simplicity, the analysis of these terms is skipped.

5. $\int_S \hat{v}_i \hat{D}_{kjil} \frac{\partial \hat{v}_k}{\partial x_j} \hat{n}_l\, dS$

   The last term in the right hand side of Equation 2.47 provides the contribution of the viscous term to the adjoint energy on the boundary of the domain. On the inlet and outlet boundaries the convective terms dominate the viscous terms. Hence, the contribution of the viscous terms on these boundaries can be ignored. However, wall boundaries can contribute significantly to the growth of adjoint energy due to the presence of large gradients at the boundary. Performing analysis of the contribution of these terms to adjoint energy is complex. In this thesis, the focus is primarily on finding local regions of divergence of the adjoint solution in the domain of the problem. Hence, the analysis of wall boundaries is skipped.

2.6 Analysis of convective growth term

The regions of the domain of the flow problem that significantly contribute to the divergence of the adjoint solution can be found by analyzing the integrand in one of the growth terms, the convective term, from the adjoint energy analysis. The integrand in this growth term is a quadratic product and can be bounded by performing an eigenvalue analysis. The growth term can be rewritten as

$$\hat{\mathbf{v}}^T\mathbf{M}\hat{\mathbf{v}} = \hat{\mathbf{v}}^T\frac{1}{2}(\mathbf{M}+\mathbf{M}^T)\hat{\mathbf{v}} \tag{2.53}$$

The above equation can be derived using the fact that $\hat{\mathbf{v}}^T\mathbf{M}\hat{\mathbf{v}} = \hat{\mathbf{v}}^T\mathbf{M}^T\hat{\mathbf{v}}$. Representing $\frac{1}{2}(\mathbf{M}+\mathbf{M^T})$ as the symmetric matrix $\mathbf{M_s}$ and formulating a generalized eigenvalue problem

$$\mathbf{M_s}\mathbf{v}_i = \lambda_i\mathbf{N}\hat{\mathbf{v}}_i \tag{2.54}$$

where λ_i is the generalized eigenvalue and $\hat{\mathbf{v}}_i$ is the generalized eigenvector. $\mathbf{N}$ is the symmetric positive definite matrix and $\mathbf{M}$ is the growth term matrix defined for various symmetrizing transformations in Equations 2.25, 2.26 and 2.50. As $\mathbf{M_s}$ is symmetric, the eigenvalues λ_i are guaranteed to be real. Using the Rayleigh quotient theorem for symmetric matrices

$$\hat{\mathbf{v}}^T\mathbf{M_s}\hat{\mathbf{v}} \leq \lambda_1\hat{\mathbf{v}}^{\mathbf{T}}\mathbf{N}\hat{\mathbf{v}} \tag{2.55}$$

where λ_1 is the maximum generalized eigenvalue. Consequently, the growth term in the adjoint energy analysis can be bounded using λ_1

$$\hat{\mathbf{v}}^T\mathbf{M}\hat{\mathbf{v}} \leq \lambda_1\hat{\mathbf{v}}^{\mathbf{T}}\mathbf{N}\hat{\mathbf{v}} \tag{2.56}$$

The scalar field λ_1 provides an indicator of the regions in the domain of the flow problem where the growth rate of adjoint energy is high. It has the dimensions $\frac{1}{T}$ and hence has the same meaning as that of a rate of growth term. As there is no compact formula for the maximum generalized eigenvalue, it is computed numerically by forming the matrix $\mathbf{M_s}$ at each grid point for various symmetrizing transformations and solving a 5×5 eigenvalue problem.

2.7 Results

The scalar field λ_1, which serves as an indicator field for the regions of divergence of the adjoint solution, is computed using various symmetrizing transformations for a couple of test cases introduced in Section 1.4. Figure 2-5 shows a visualization of the scalar field, λ_1, for subsonic flow over a cylinder using the non-entropy based Turkel symmetrizing transformation. The scalar field is normalized by a numerical approximation to $\|\lambda_1\|_2$, where $\|\cdot\|_2$ denotes the spatial L_2 norm given by the following expression

$$\|\lambda_1\|_2 = \Big(\int_V \lambda_1^2 \, dV \Big)^{\frac{1}{2}} \tag{2.57}$$

The scalar field is positive in every location of the mesh. As indicated by the magnitude of λ_1 in different regions of the flow in the figure, the region in the boundary layer near the surface of the cylinder and the near wake region are primarily responsible for the diverging adjoint solution. This matches a physical understanding of the flow problem, that perturbations to the flow solution in the boundary layer and near wake region have a major impact on the dynamics of the flow downstream of the cylinder.

Figures 2-6, 2-7 and 2-8 show visualizations of the scalar field, λ_1 for transonic flow over a turbine vane using different symmetrizing transformations. The scalar field is normalized by its spatial L_2 norm. The scalar field is positive in every location of the mesh. A visual inspection of the figures shows that even the magnitude of the indicator fields is different for different symmetrizing transformations. Even so, across all figures, it is larger in the same region, turbulent wake downstream of the trailing edge of the vane, compared to other regions of the flow. This indicates that the indicator field is largely independent of the symmetrizing transformation. In addition to the turbulent wake, all the indicator fields have a large magnitude in the trailing edge region of the suction side of the turbine vane. This is expected from a physical understanding of the problem. This is the region where the turbulent boundary layer

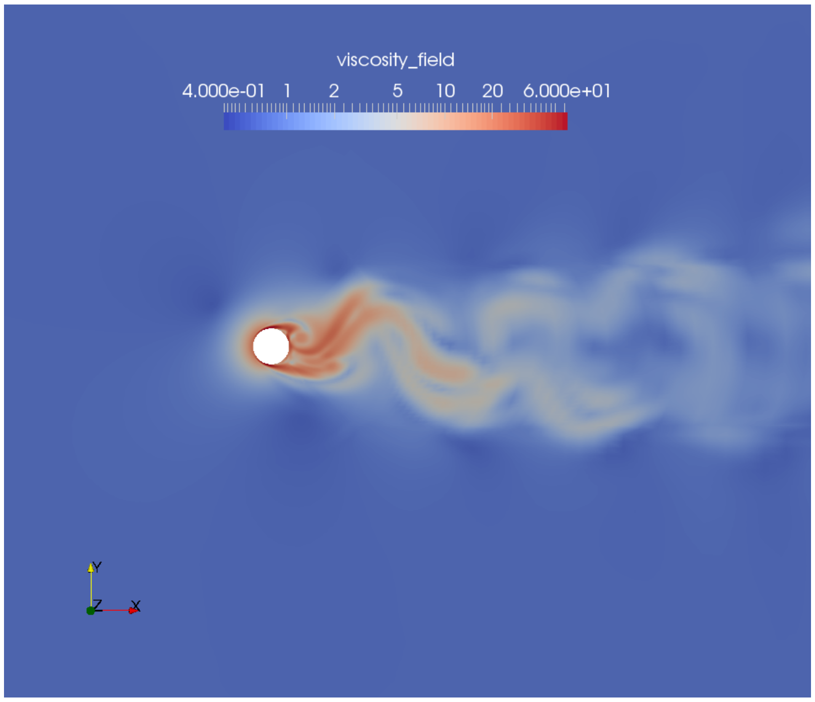


Figure 2-5: Visualization of the scalar field, λ_1, computed using Turkel symmetrizing transformation and normalized using its spatial L_2 norm. The color scale is logarithmic.

is formed, which leads to flow separation and formation of a turbulent wake.

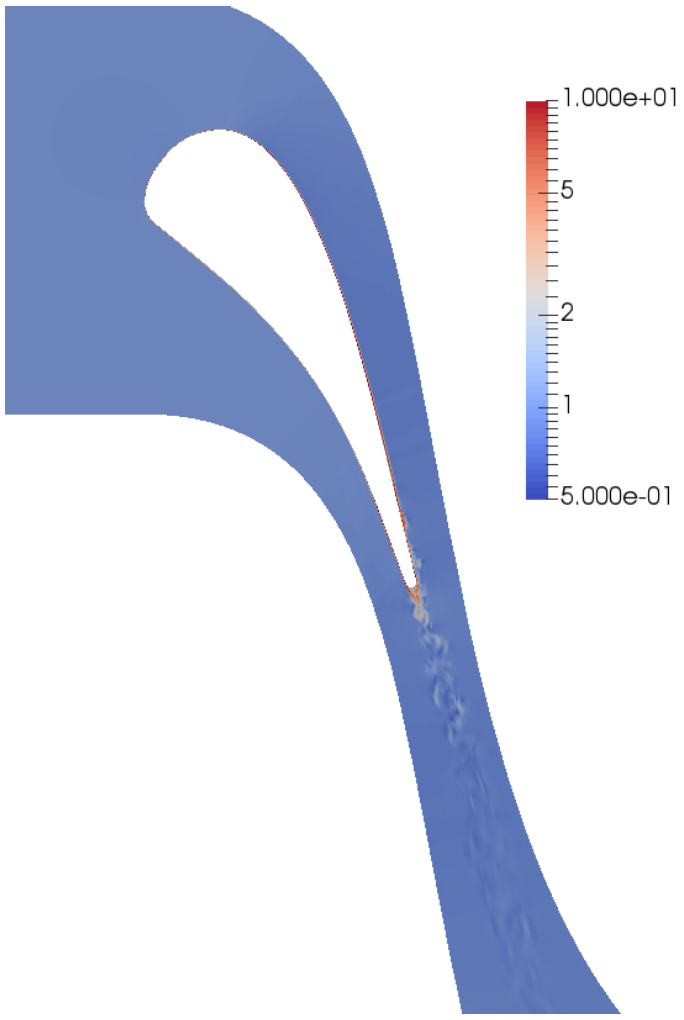


Figure 2-6: Visualization of the scalar field, λ_1, constructed using the Abarbanel symmetrizing transformation.

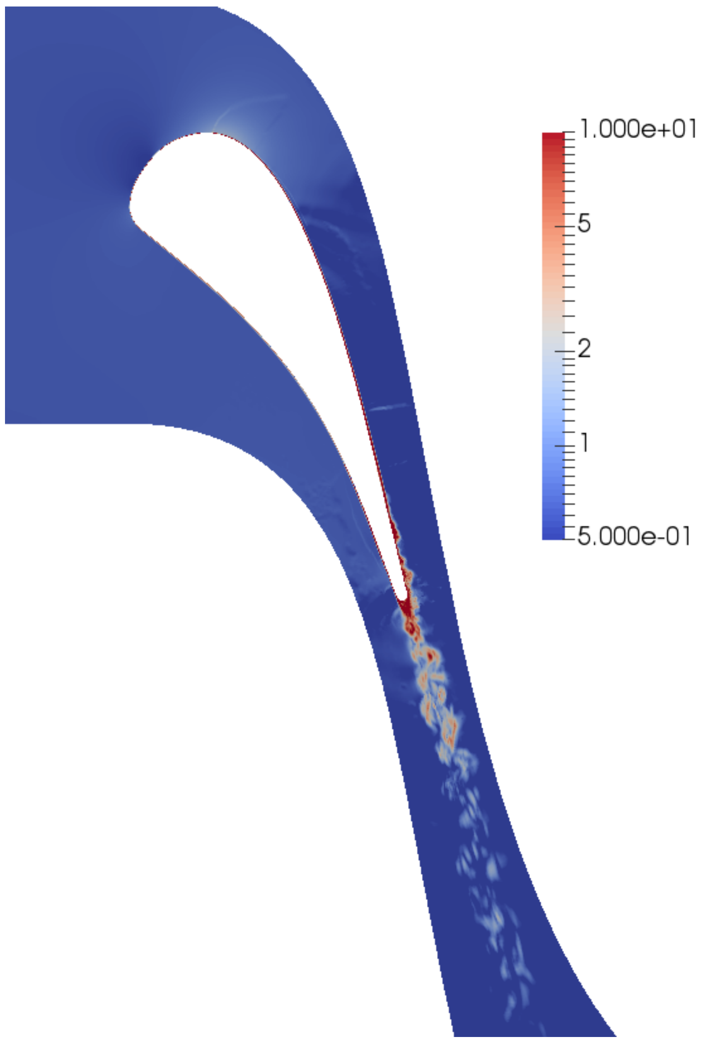


Figure 2-7: Visualization of the scalar field, λ_1, constructed using the Turkel symmetrizing transformation.

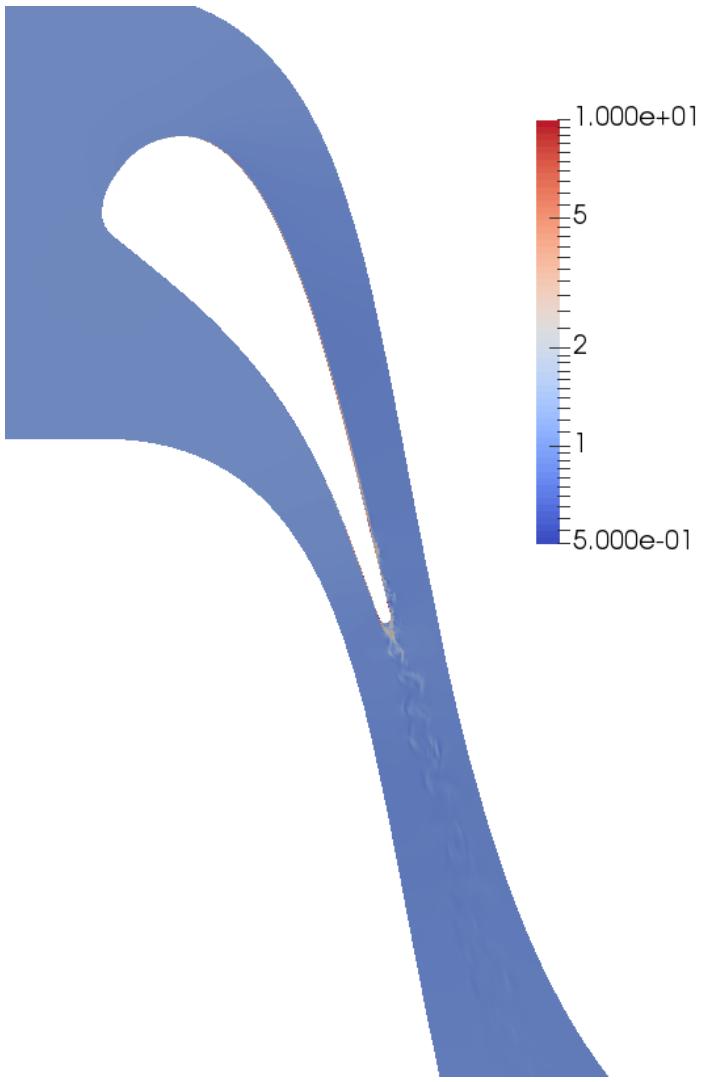


Figure 2-8: Visualization of the scalar field, λ_1, constructed using the Hughes symmetrizing transformation

Chapter 3

Viscosity stabilized adjoint method

In this chapter, the viscosity stabilized adjoint method is derived. Section 3.1 discusses the various ways in which the symmetrized form of the adjoint equations can be modified in order to control their divergence. Section 3.2 describes how artificial viscosity is added to the adjoint equations. Finally, Section 3.3 demonstrates results from applying the viscosity stabilized adjoint method to a couple of flow problems.

3.1 Modification of adjoint equations

The symmetrized adjoint equations, Equation 2.39, formed using the entropy symmetrization method, are modified in order to reduce the growth of adjoint energy and stabilize the adjoint solution. The modification should be designed in such a way so as to preserve the linearity of the adjoint equations. If a design objective is scaled by a constant, a, the design objective gradient should also be scaled by the same constant a. But, if the modified adjoint equations are nonlinear in the adjoint solution ($\hat{\mathbf{v}}$), the design objective gradient may not satisfy this property. Hence, the modification term cannot be a nonlinear function of $\hat{\mathbf{v}}$, but, can be a nonlinear function of the flow solution ($\mathbf{w}$). Additionally, the linearity property restricts the modification to the class of additive transformations to the various linear operators in the adjoint equations. There are multiple terms in the symmetrized adjoint equations (Equation 2.39) which can be modified. They are:

1. Source term (s_i)
2. Convective term ($\hat{A}_{kji}\dfrac{\partial \hat{v}_k}{\partial x_j}$)
3. Right hand side viscous term ($\hat{D}_{kjil}\dfrac{\partial \hat{v}_k}{\partial x_j}$)

The first modification is adding a linear source term ($\eta^s \Lambda^s_{ij} \hat{v}_j$) to the right hand side of Equation 2.39. If the matrix formed by Λ^s_{ij} is negative definite and the constant term η^s is greater than 0, this source term will always decrease the adjoint energy. This modification does not preserve the convective property of the adjoint equations. The convective property refers to the fact that the adjoint equations form a set of linear convection-diffusion partial differential equations with time dependent coefficients and design objective based source terms. The importance of this property is discussed in Section 3.1.1.

The convective term can be modified in the following manner: $\hat{A}_{kji} \rightarrow (\hat{A}_{kji} + \eta^c \Lambda^c_{kji})$, where Λ^c_{kji} is a symmetric matrix for a given index j and η^c is a constant term greater than 0. In order to ensure reduction of adjoint energy, the matrix formed by $\frac{\partial \Lambda^c_{kji}}{\partial x_j}$ has to be positive semi-definite everywhere in the domain of the flow problem. Ensuring this property is not trivial, if not impossible, while constructing a Λ^c_{kji} that is purely a function of the flow solution.

The right hand side viscous term can be modified in the following manner: $\hat{D}_{kjil} \rightarrow (\hat{D}_{kjil} + \eta^v \Lambda^v_{kjil})$, where the matrix formed by $\Lambda^V_{[3(j-1)+k][3(l-1)+i]} = \Lambda^v_{kjil}$ is symmetric positive semi-definite and η^v is a constant term greater than 0. From the symmetrized adjoint analysis in Equation 2.47, it can be seen that this modification will always lead to a decrease in adjoint energy. Additionally, this modification preserves the convective property of the adjoint equations as the modified equations still form a set of linear convection-diffusion partial differential equations.

3.1.1 Importance of convective property

The importance of preserving the convective property can be seen from the following example. Consider a subsonic flow in a box of dimensions $1\,m \times 1\,m \times 0.1\,m$ with inlet and outlet boundaries on opposite faces of the box. The objective function for the adjoint equation is the mass flow on the outlet.

$$\bar{J} = \lim_{T\to\infty} \frac{1}{T} \int_0^T \int_S \rho \mathbf{u} \cdot \mathbf{n} \, dS \tag{3.1}$$

The boundary conditions of the flow problem are the following: velocity at inlet boundary $(u_{in}) = 100\, m/s$, pressure at outlet boundary $101325\, Pa$, with periodic boundary conditions in directions transverse to the flow. The finite time interval over which the adjoint solution is computed is 5 time units, where 1 time unit is the amount of time the flow takes to travel from the inlet to the outlet boundary.

Using the finite difference method for estimating gradients, the gradient $\frac{\partial \bar{J}}{\partial u_{in}}$ is approximately $0.1189\, kg/m$. The gradient provided by the adjoint solution matches the gradient estimated using finite difference. The form of the viscous term modification is $\Lambda^v_{kjil} = \frac{\lambda_1}{\|\lambda_1\|_2} \delta_{ki}\delta_{jl}$ and the form of the source term is $\Lambda^s_{ij} = -\frac{\lambda_1}{\|\lambda_1\|_2}\delta_{ij}$, where λ_1 is the maximum generalized eigenvalue of $\mathbf{M_s}$ as derived in Section 2.6. Figure 3-1 shows the absolute error in the gradient computed from the adjoint solutions obtained by modifying the viscous term or adding a source term to the conservative form of the adjoint equations.

Even though the dimensions of the viscous term and source term scaling factors are different, the figure still shows a fair comparison of the absolute error for the two types of modifications. This is because the ranges of both the scaling factor in the figure correspond to the range of minimum scaling factors for which the adjoint energy is stable for different flow problems, like subsonic flow over a cylinder or transonic flow over a turbine vane. The figure shows that when the source term of the adjoint equations is modified using a scaling factor that is roughly sufficient to stabilize the adjoint solution, the error in the gradient is much higher than the case when the viscous term of the adjoint equations is modified using a scaling factor that is sufficient to stabilize the adjoint solution. Hence, preserving the convective property of the adjoint equations is important as it better preserves the accuracy of the gradient provided by the adjoint solution.

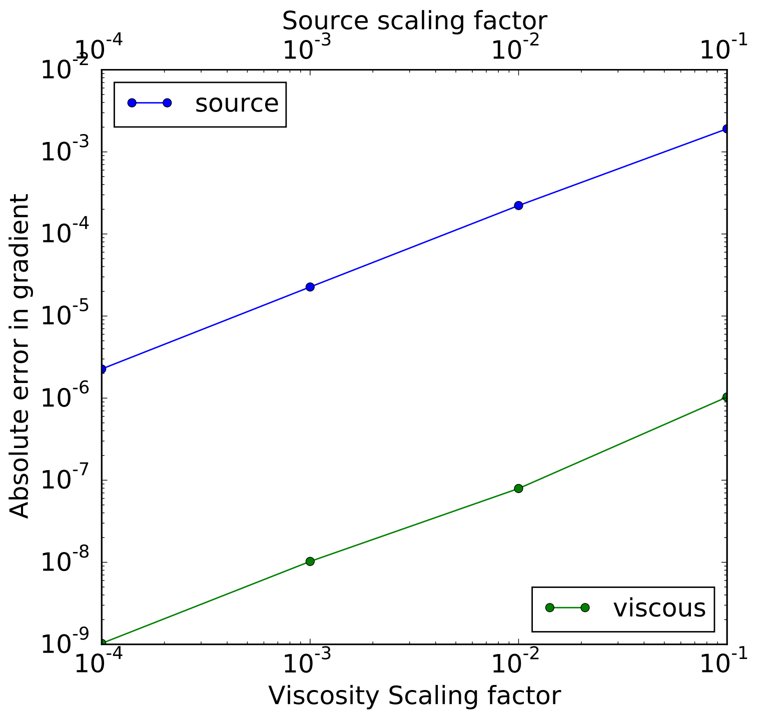


Figure 3-1: Absolute error in gradient computed from adjoint solutions using viscous term modification (viscosity scaling factor η_v, units: $1/s$) and source term modification (with source scaling factor η_s, units: m^2/s)

Consequently, in this thesis, modifying the viscous term of the adjoint equations is the preferred method for stabilizing the adjoint solution. Modifying the convective term has the disadvantage that forming the positive semi-definite matrix is difficult, whereas adding a source term has the disadvantage of not preserving the convective property of the adjoint equations.

3.2 Addition of artificial viscosity to adjoint equations

As discussed in the previous section, the adjoint equations are modified by perturbing the viscous term ($\hat{D}_{kjil}$). It is important to note that only the adjoint equations are modified by adding artificial viscosity, whereas the primal equations remain the same with no additional viscosity. The error due to the addition of artificial viscosity is estimated in Chapter 4.

The tensor Λ^v_{kjil} in the modification is given by

$$\Lambda^v_{kjil} = \lambda_1, \;\; \forall k = i, j = l \tag{3.2}$$

$$\Lambda^v_{kjil} = 0, \;\; \forall k \neq i \,||\, j \neq l \tag{3.3}$$

where λ_1 is the maximum generalized eigenvalue of $\mathbf{M_s}$, derived in Section 2.6 for the non-entropy based symmetrizing transformations (Abarbanel and Turkel) and the entropy based symmetrizing transformations (Hughes). The scalar field, λ_1, is an indicator of the regions of divergence of the adjoint solution and the tensor $\mathbf{\Lambda^v}$ serves as an artificial viscosity tensor field for the modified adjoint equations. As it is directly proportional to λ_1, it adds large amounts of viscosity in the regions where the adjoint solution is diverging. From the above equations it can be seen that the matrix $\mathbf{\Lambda^V}$ is symmetric positive semi-definite. Hence, ensuring that this particular form of the viscous term modification always reduces the adjoint energy. The matrix is positive semi-definite as the scalar field, λ_1, is empirically observed to be positive everywhere in the domain of multiple flow problems. There are many other ways to construct an appropriate tensor $\mathbf{\Lambda^v}$ such that the adjoint energy is always reduced. A couple of them are listed below

$$\Lambda^v_{:1:1} = \Lambda^v_{:2:2} = \Lambda^v_{:3:3} = \lambda_1 \mathbf{A^0} \tag{3.4}$$

$$\Lambda^v_{kjil} = 0, \;\; j \neq l \tag{3.5}$$

and finally

$$\Lambda^v_{kjil} = \lambda_1 \hat{D}_{kjil} \tag{3.6}$$

For simplicity, Equation 3.3 is chosen as the artificial viscosity tensor field, $\mathbf{\Lambda^v}$, for this thesis. It can also be written in the following form

$$\Lambda^v_{kjil} = \lambda_1 \delta_{ki} \delta_{jl} \tag{3.7}$$

Hence, the resulting modified adjoint equations in the symmetrized form are

$$-A^0_{ij}\frac{\partial \hat{v}_j}{\partial t} - (\hat{A}_{kji} - \hat{A}^v_{kji})\frac{\partial \hat{v}_k}{\partial x_j} = \frac{\partial}{\partial x_l}((\hat{D}_{kjil} + \eta^v \lambda_1 \delta_{ki}\delta_{jl})\frac{\partial \hat{v}_k}{\partial x_j}) \tag{3.8}$$

where the constant term η^v serves as a scaling factor with dimensional units length squared. The above equation can be simplified to

$$-A^0_{ij}\frac{\partial \hat{v}_j}{\partial t} - (\hat{A}_{kji} - \hat{A}^v_{kji})\frac{\partial \hat{v}_k}{\partial x_j} = \frac{\partial}{\partial x_l}(\hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j}) + \eta^v \frac{\partial}{\partial x_j}(\lambda_1 \frac{\partial \hat{v}_i}{\partial x_j}) \tag{3.9}$$

A non-dimensional form of the scaling factor can be obtained using the following formula

$$\eta^v = \eta \frac{\mu_r}{\rho_r}\frac{1}{\|\lambda_1\|_\infty} \tag{3.10}$$

where μ_r and ρ_r are reference quantities defined in Section 1.4, resulting in the following modified adjoint equations in the symmetrized form

$$-A^0_{ij}\frac{\partial \hat{v}_j}{\partial t} - (\hat{A}_{kji} - \hat{A}^v_{kji})\frac{\partial \hat{v}_k}{\partial x_j} = \frac{\partial}{\partial x_l}(\hat{D}_{kjil}\frac{\partial \hat{v}_k}{\partial x_j}) + \eta \frac{\mu_r}{\rho_r}\frac{1}{\|\lambda_1\|_\infty}\frac{\partial}{\partial x_j}(\lambda_1 \frac{\partial \hat{v}_i}{\partial x_j}) \tag{3.11}$$

where η is the non-dimensional tunable scaling factor and $\|\lambda_1\|_\infty$ is the maximum value of the scalar field λ_1 in the domain of the flow problem. η is a positive value which can be tuned to a value which is sufficient to stabilize the adjoint solution. Hence, the modification term for the symmetrized adjoint equations is in the form of a Laplacian operator.

The symmetrized form of the adjoint equations can be indirectly modified by altering the viscous term on the right side of the conservative form of the adjoint equations (Equation 1.24).

$$-\frac{\partial \hat{w}_i}{\partial t} - (A_{kji} - A^v_{kji})\frac{\partial \hat{w}_k}{\partial x_j} = \frac{\partial}{\partial x_l}(D_{kjil}\frac{\partial \hat{w}_k}{\partial x_j}) + \eta \frac{\mu_r}{\rho_r}\frac{1}{\|\lambda_1\|_\infty}\frac{\partial}{\partial x_j}(\lambda_1 \frac{\partial \hat{v}_i}{\partial x_j}) \tag{3.12}$$

While it is not possible to show that the above modification will always result in a reduction in the adjoint energy, practically, it has been observed to exhibit that effect and stabilize the adjoint solution on multiple flow problems as seen in Section 3.3.

Figure 3-2 shows a comparison between adding artificial viscosity to the conservative form and symmetrized form of the adjoint equations using $\eta = 10$ for both cases. The figure shows that the difference between the two cases in negligible when comparing the effectiveness in reducing the exponential growth of adjoint energy. But, the computational cost of adding artificial viscosity to the conservative form of the adjoint equations is lower than adding to the symmetrized form. This is due to the fact that when using the implicit Euler time integration scheme for adding the viscosity, the coupled linear system formed for the symmetrized adjoint equations is much larger. Hence, in this thesis, adding artificial viscosity to the conservative form of the adjoint equations is the preferred method.

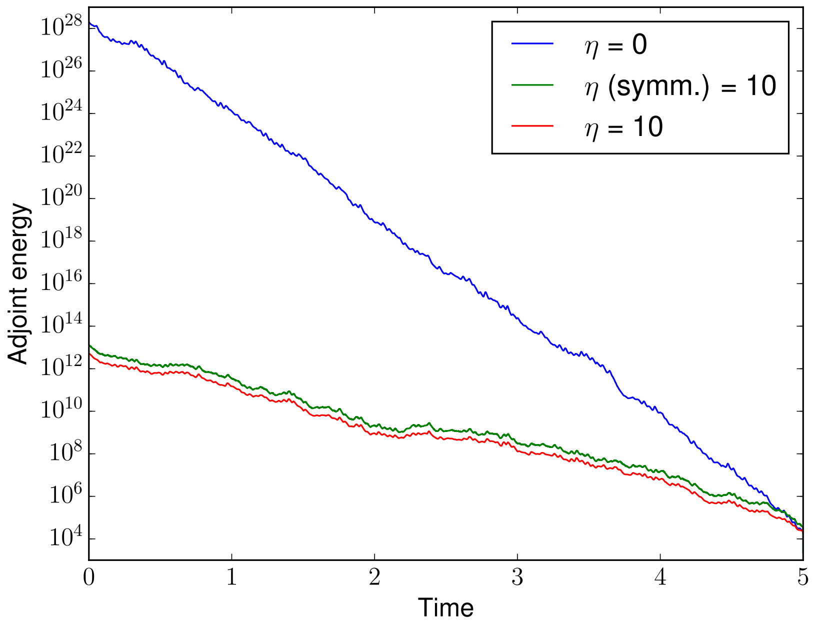


Figure 3-2: Comparison of plot of adjoint energy as a function of time (represented by time units) between adding artificial viscosity to the conservative form of adjoint equations ($\eta = 10$ in figure legend) and adding artificial viscosity to the symmetrized form of adjoint equations (η(symm.) $= 10$ in figure legend).

3.2.1 Choice of scaling factor

The choice of η is important as it can influence the value of the gradient obtained from the adjoint solution of the modified equations. Large values of η will result in fast stabilization of the adjoint solutions, but might adversely affect the accuracy of the gradients obtained from the adjoint solution. In contrast, small values of η might not be sufficient to stabilize the adjoint solution. The optimal value of η is the minimum value of η for which the adjoint solution of the modified equations is stable. Stability of the adjoint solution can be determined either through a visual inspection of the adjoint energy as a function of time. If the adjoint energy does not appear to grow exponentially backwards in time, then it is considered stable. A quantitative method to determine the stability of an adjoint solution is to compute the largest Lyapunov exponent. If it is less than or equal to 0, then the adjoint solution is stable. The theory and computation of Lyapunov exponents is described in Section 4.1.

One strategy for finding the optimal value of η is to find multiple adjoint solutions with different values of η and choose the minimum η which stabilizes the adjoint solution over the entire finite time interval. For example, one possible set of values of η can be obtained by dividing the range $[10^{-1}, 10^{4}]$ into 10 equal intervals in log scale. This method requires obtaining 11 different adjoint solutions using 11 values of the scaling factor. This strategy, also known as the shooting strategy, has the advantage that the number of adjoint solutions that need to be found is generally fixed.

Another strategy for finding optimal value of η is the following: first start with an initial estimate. For example, η can be set to 1 which doubles the amount of viscosity in the adjoint equations in certain regions of the domain of the flow problem. If the adjoint solution is stable, then η is halved. Otherwise, η is doubled. The process is repeated until consecutive values of η show different stability properties, meaning that for one value η the adjoint solution is stable and for the next value of η it is unstable, or vice versa. The final value η for which the adjoint solution is stable is chosen as the approximately optimal value of η. If a more refined value of η is needed, the step factor between consecutive η can be set to a value smaller than 2. This strategy, also known as the bisection strategy, has the advantage that the optimal value of η can generally be found in high precision with a lower cost than the previous strategy.

3.2.2 Algorithm

The final algorithm for computing the design objective gradient using the viscosity stabilized adjoint method is described below

1. Solve the flow equations to obtain the flow solution from $t = 0$ to $t = T$.
2. Solve the viscosity stabilized adjoint equations from $t = T$ to $t = 0$ for multiple scaling factors ($\eta = \eta_1, ..., \eta_P$) using the shooting strategy.
3. Find minimum $\eta = \eta_m$ that stabilizes the adjoint solution.
4. Return the gradient computed from the adjoint solution for η_m as the approximation to the design objective gradient.

The cost of this algorithm is $PC_a + C_p$, where C_a is the computational cost of obtaining a single adjoint solution and C_p is the cost of obtaining a single flow solution. The details of the numerical implementation of this method are described in Section 3.2.4. Roughly, through numerical experiments, it has been observed that $C_a \sim 4C_p$. Hence the cost of the viscosity stabilized adjoint method is $(4P + 1)C_p$. In practice, many of the adjoint simulations for different artificial viscosity scaling factors can be stopped early if it is observed that they are diverging. Hence, potentially, the cost of the algorithm can be much lower than $(4P + 1)C_p$.

3.2.3 Regularity of artificial viscosity field

The regularity of the artificial viscosity field affects the existence and uniqueness properties of the adjoint solution. Assuming that the flow solution given by the fields $\rho, \boldsymbol{u}$ and p are regular and reside in the Hilbert space $H^1(D)$, where D is the domain of the flow problem. Then, the gradients of the flow solution given by $\nabla\rho, \nabla\boldsymbol{u}$ and

∇p) reside in the Hilbert space $H^0(D)$. This implies that the following term

$$\|\nabla\rho\|^2_{H^0(D)} = \int_D (\nabla\rho)_i^2 \, dV \tag{3.13}$$

is bounded. Consequently, each component of the tensor field $\mathbf{M}$ is regular, $\boldsymbol{M}_{ij} \in H^0(D)$, as it is a function of the flow solution and its gradients. Unfortunately, the maximum eigenvalue operation can produce a scalar field that does not have any regularity. This can be observed from the fact that the maximum root of smoothly parameterized polynomial equations can have jumps as a function of the parameters of the polynomial. Hence, the artificial viscosity field is irregular.

An irregular or discontinuous viscosity field could potentially cause issues when trying to obtain the solution to the modified adjoint equations. But, as demonstrated in Section 2.7, in practice, the artificial viscosity fields derived from the divergence indicator fields are smooth or continuous to the naked eye for various flow problems. Furthermore, the linear solver used to obtain numerical solutions to the modified adjoint equations converges in $10-20$ iterations. Hence, empirically, the mathematical irregularity of the artificial viscosity field is not a concern, as shall be seen in Section 3.3.

3.2.4 Numerical implementation of artificial viscosity

The numerical solution of the compressible Navier-Stokes equations is obtained by discretizing the conservative formulation of these equations using FVM with explicit time integration. This is the preferred approach as it ensures a strict discrete conservation of mass, momentum and energy in the numerical flow solution. While entropy formulations are necessary for stability and conservation in higher-order methods for compressible Navier-Stokes [21], discretely conservative schemes are generally stable for low-order discretization schemes [57].

The numerical solution of the modified adjoint equations is obtained by time marching using an Implicit-Explicit (IMEX) Euler-RK scheme [17]. Each time step of this scheme can be divided into two steps, an explicit Runge-Kutta (RK) time integration step followed by an implicit Euler time integration step. In the first step, the numerical solution to the adjoint equations without artificial viscosity is obtained using the discrete adjoint method applied to the computations involved in numerically approximating the flow solution of the conservative form of the compressible Navier-Stokes. The details of the discrete adjoint method and its implementation are described in Section 6.1. This method provides an approximate solution of the following equations

$$-\frac{\partial \hat{w}_i}{\partial t} - (A_{kji} - A^v_{kji})\frac{\partial \hat{w}_k}{\partial x_j} = \frac{\partial}{\partial x_l}(D_{kjil}\frac{\partial \hat{w}_k}{\partial x_j}) \tag{3.14}$$

In the second step of the time marching scheme, artificial viscosity is added to the numerical solution of the adjoint equations using the following equations

$$-A^0_{ij}\frac{\partial \hat{v}_j}{\partial t} = \frac{\partial}{\partial x_l}(\eta\frac{\mu}{\rho}\frac{\lambda_1}{\|\lambda_1\|_\infty}\delta_{ki}\delta_{jl}\frac{\partial \hat{v}_k}{\partial x_j}) \tag{3.15}$$

This is done by performing an implicit Euler time integration step. An implicit integration step is necessary due to potential stiffness of the artificial viscosity term. A linear solver, GMRES with ILU preconditioner [111], is used to solve the linear system formed by the implicit time integration scheme. The above equation is solved using Neumann boundary conditions for the adjoint solution. In the case where artificial viscosity is added to the conservative form of the adjoint equations, the following equations are solved

$$-\frac{\partial \hat{w}_i}{\partial t} = \frac{\partial}{\partial x_l}(\eta\frac{\mu}{\rho}\frac{\lambda_1}{\|\lambda_1\|_\infty}\delta_{ki}\delta_{jl}\frac{\partial \hat{w}_k}{\partial x_j}) \tag{3.16}$$

The implicit time integration step requires solving a linear system of equations. When running on a single CPU, it increases the wall clock time per time step by approximately $10 - 50\%$ depending on the amount of artificial viscosity added to the adjoint equations. When running on a large number of cores, the parallel scaling efficiency of the linear solver reduces, leading to a performance penalty for obtaining the numerical adjoint solution. Hence, in order to reduce the computational cost for solving the modified adjoint equations, instead of adding artificial viscosity at every time step, it can be added every 10 or 100 time steps. As the size of each time step is small, due to the use of explicit time integration for the primal equations and the existence of thin boundary layers in the flow problems, not adding viscosity at every time step allows computing the adjoint solution over longer time intervals. The scaling factor (η) can be appropriately increased to maintain the stability of the adjoint solution. By default, artificial viscosity is added every 10 time steps for each of the simulations of the modified adjoint equations in this thesis, unless otherwise specified. This is roughly equivalent to increasing the scaling factor by a factor of 10 than what is reported. The additional error introduced in the adjoint solution due to adding viscosity every 10 time steps instead of every time step is analyzed in Appendix B.

3.3 Results

The effectiveness of the viscosity stabilized adjoint method in stabilizing the adjoint solution and providing an approximation to the design objective gradient is tested on a couple of test cases introduced in Section 1.4.

The first test case is subsonic flow over a cylinder. The adjoint solution is computed for the design objective: time-averaged drag over the surface of the cylinder. The length of the time interval over which the solution is obtained is 5 time units. The gradient of the design objective is computed with respect to the inlet Mach number. The Turkel symmetrizing transformation is used for constructing the artificial viscosity field. Multiple adjoint solutions are obtained using the scaling factors, $\eta = 1, 10, 30, 100, 300, 1000$ and $3,000$. Artificial viscosity is added to conservative form of the adjoint equations, using Equation 3.12. Figure 3-3 shows the behavior of the adjoint energy (as defined for entropy symmetrizing transformations in Equation 2.41) for different scaling factors. When the scaling factor is low ($\eta = 1$), the additional viscosity does not significantly affect the adjoint solution and hence, the energy of the adjoint solution remains high. On increasing the scaling factor to $\eta = 10$, the energy of the adjoint solution reduces, but still shows exponential growth. Further increasing the scaling factor to $\eta = 30$ halts the exponential growth of adjoint energy and stabilizes the adjoint solution. At this point, the magnitude of the adjoint energy maintains an approximate steady value over the entire finite time interval over which the adjoint solution is computed. The minimum value of η for which the adjoint solution is stabilized is $\eta = 30$.

The accuracy of the gradients obtained using the viscosity stabilized adjoint method (g_η) is tested by comparing them with gradients obtained using the finite difference method (g_δ). Figures 3-4 and 3-5 show that the gradients computed from the viscosity stabilized adjoint solutions approximately match finite difference gradient when η is in the range 30 to $1,000$. The uncertainty in the adjoint gradients is computed using the time series analysis techniques discussed in Section 5.1.2. The relative error is defined as the following,

$$e_\eta = \frac{|g_\eta - g_\delta|}{|g_\delta|} \tag{3.17}$$

For the gradients obtained from the adjoint solution, it varies from 1% to 25% in the aforementioned range of η. The optimal adjoint gradient with the lowest relative error is obtained for $\eta = 300$, $g_\eta = 25.1 \pm 2.0$. This value of η is close (in log scale) to the minimum value of η for which the adjoint solution is stabilized.

The gradient of the design objective when computed using the ensemble adjoint method [25] is 15.57 ± 3.1. The ensemble adjoint method is applied to the same finite time interval used for the viscosity stabilized adjoint method. The ensemble is formed by splitting this interval into 20 segments. The ensemble adjoint gradient has a higher relative error 40% and a higher variance compared to the viscosity stabilized adjoint gradient.

Figures 3-6 and 3-7 show a comparison between a visualization of the density adjoint solution at $t = 2.5$ time units (mid point of the finite time interval over which the adjoint solution is computed) obtained from solving the adjoint equations with and without artificial viscosity. Large magnitudes of the density adjoint solution indicate

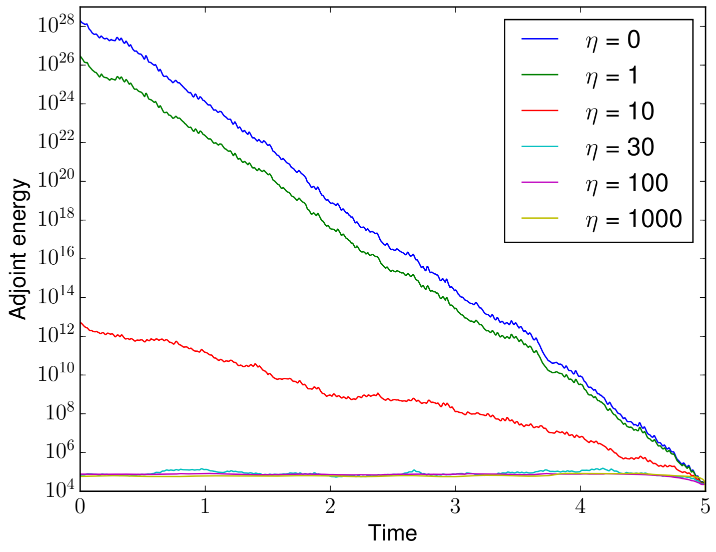


Figure 3-3: Plot showing the growth of adjoint energy (units: kg s/m^3) for various artificial viscosity scaling factors (η) as a function of time (represented by time units) for the flow over cylinder problem.

the regions of the domain where perturbations in the flow can lead to large changes in the objective [65, 113]. The figures show that without any artificial viscosity, the density adjoint solution has large magnitudes near the boundary layer region of the cylinder and the diverging solution is convected upstream towards the inlet boundary. In contrast, with artificial viscosity, the density adjoint solution does not diverge and stays bounded when convected upstream.

The second test problem for the viscosity stabilized adjoint method is transonic flow over a turbine vane. The adjoint solution is computed for the design objective: time-averaged pressure loss coefficient downstream of the trailing edge of the vane. The gradient of the design objective is computed with respect to a source term perturbation upstream of the leading edge of the vane. The length of the time interval over which the solution is obtained is 2 time units. Similar to the cylinder problem, the Turkel symmetrizing transformation is used and artificial viscosity is added to the conservative form of the adjoint equations. Figure 3-8 shows the growth of adjoint energy for various artificial viscosity scaling factors ranging from $\eta = 0$ to $\eta = 13,333$.

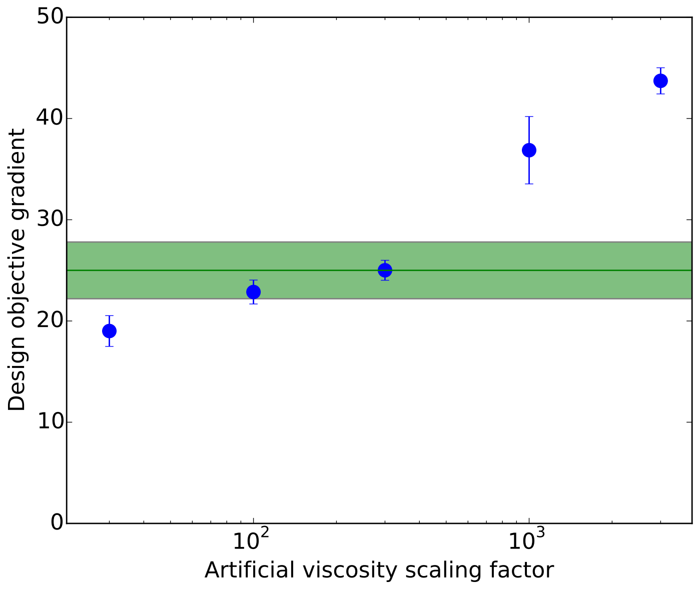


Figure 3-4: Plot showing the design objective gradient computed using adjoint solutions with different artificial viscosity scaling factors (η) vs finite difference gradient for the flow over cylinder problem. The blue dots denote the adjoint gradients and the error bars denote the uncertainty in the gradients. The center line in the green shaded region denotes the finite difference gradient and the shaded region itself denotes the uncertainty in the gradient.

The minimum value of η for which the adjoint solution is stabilized is $\eta = 4,444$. Figure 3-9 shows the design objective gradient obtained using adjoint solutions computed for different scaling factors. Figure 3-10 shows the relative error in the design objective gradient as a function of the artificial viscosity scaling factor. Similar to the cylinder problem, for η in the range $4,444$ to $133,333$, the design objective gradient computed from the adjoint solution has less than 25% relative error with respect to the finite difference gradient. The optimal value of η is $44,444$, which is close (in log scale) to the minimum value of $\eta = 4,444$ for which the adjoint solution is stabilized.

Figures 3-11 and 3-12 show a comparison between a visualization of the density adjoint solution at $t = 1$ time unit (mid point of the finite time interval over which the adjoint solution is computed) obtained from solving the adjoint equations with and without artificial viscosity. The figures show that without any artificial viscosity, the density adjoint solution has large magnitudes near the trailing edge and suction

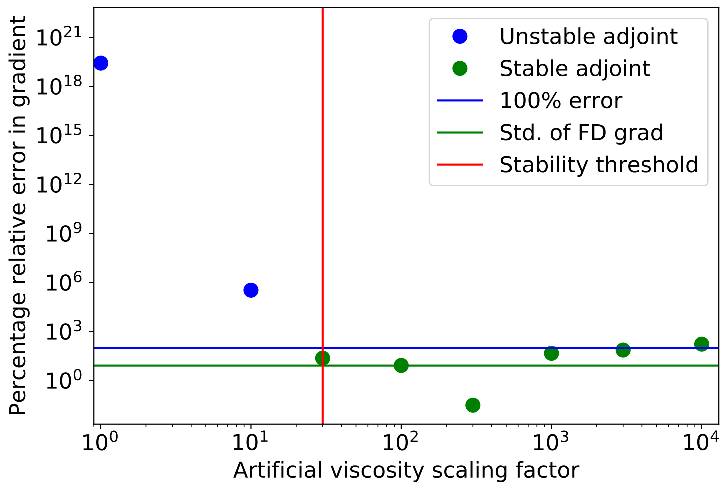


Figure 3-5: Plot showing the relative error in the design objective gradient computed using adjoint solutions with different artificial viscosity scaling factors (η) for the flow over cylinder problem. The blue dots denote unstable adjoint solutions and green dots denote stable adjoint solutions. The blue line denotes 100% relative error ,the red line denotes the minimum η for which the adjoint solution is stable and the green line denotes the standard deviation of the finite difference gradient estimate.

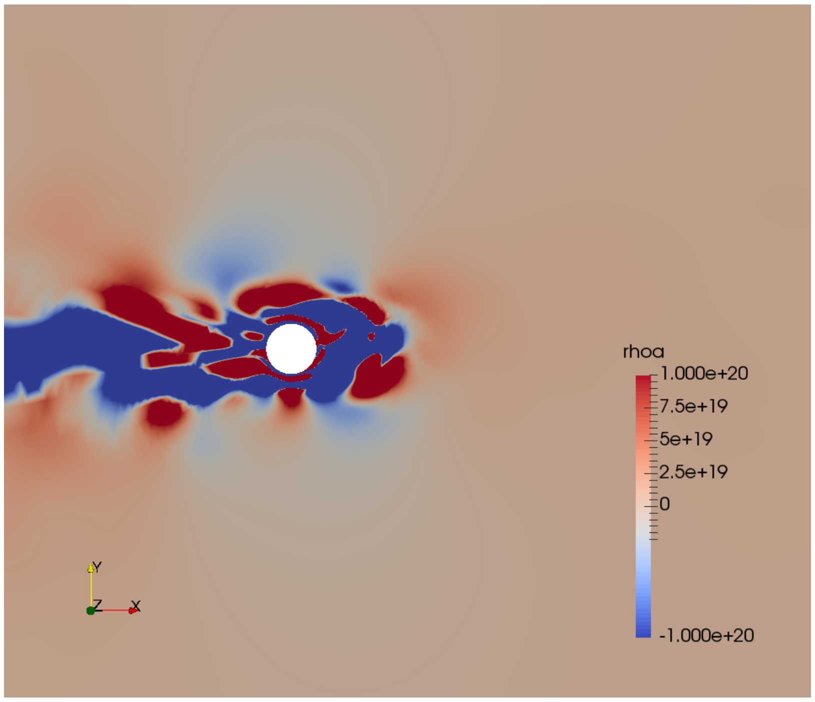


Figure 3-6: Density adjoint solution (units: m^3 s/kg) at t = 2.5 time units without any artificial viscosity.

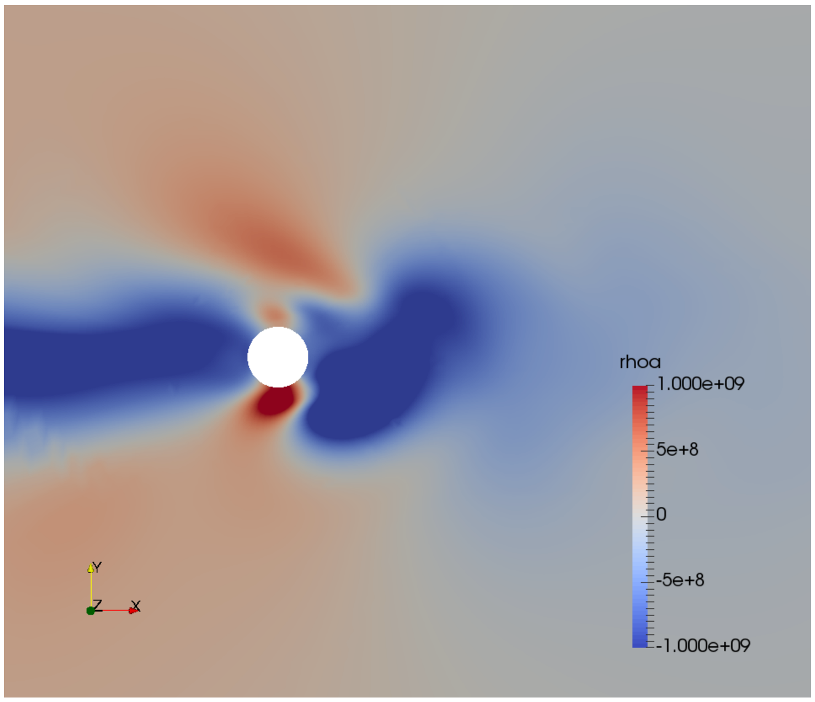


Figure 3-7: Density adjoint solution (units: m^3 s/kg) at $t = 2.5$ time units with artificial viscosity using $\eta = 30$.

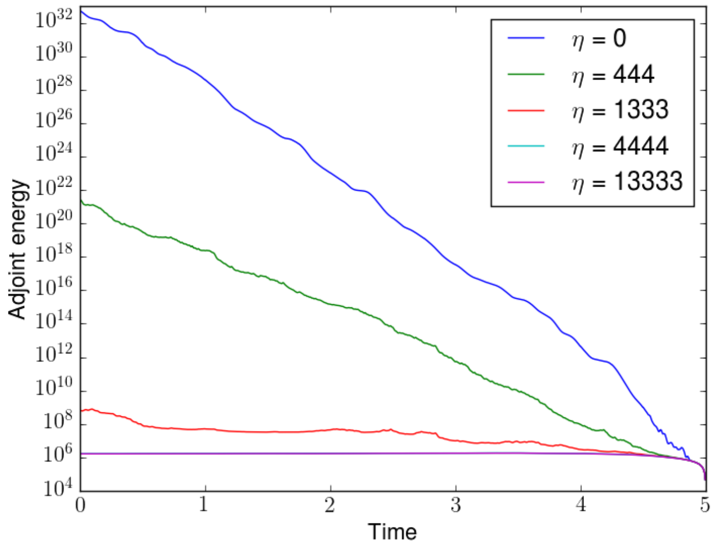


Figure 3-8: Plot showing the growth of adjoint energy (units: kg s/m^3) for various artificial viscosity scaling factors (η) as a function of time (represented by time units) for the turbine vane problem.

side of the surface of the vane. In contrast, with artificial viscosity, the density adjoint

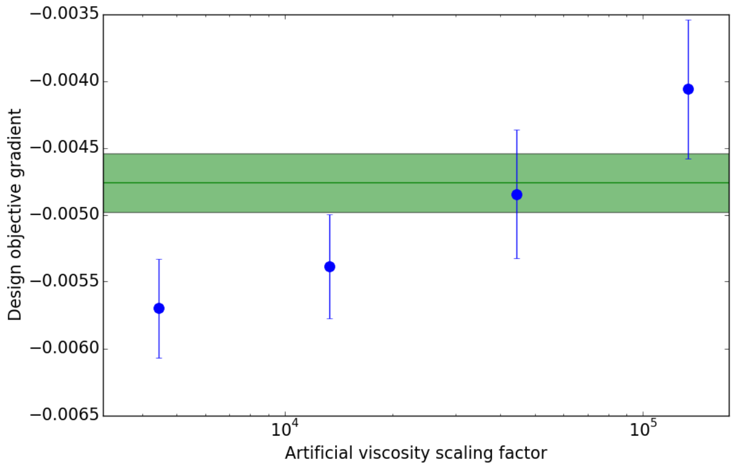


Figure 3-9: Plot showing the design objective gradient computed using adjoint solutions with different artificial viscosity scaling factors (η) vs finite difference gradient for the turbine vane problem. The blue dots denote the adjoint gradients and the error bars denote the uncertainty in the gradients. The center line in the green shaded region denotes the finite difference gradient and the shaded region itself denotes the uncertainty in the gradient.

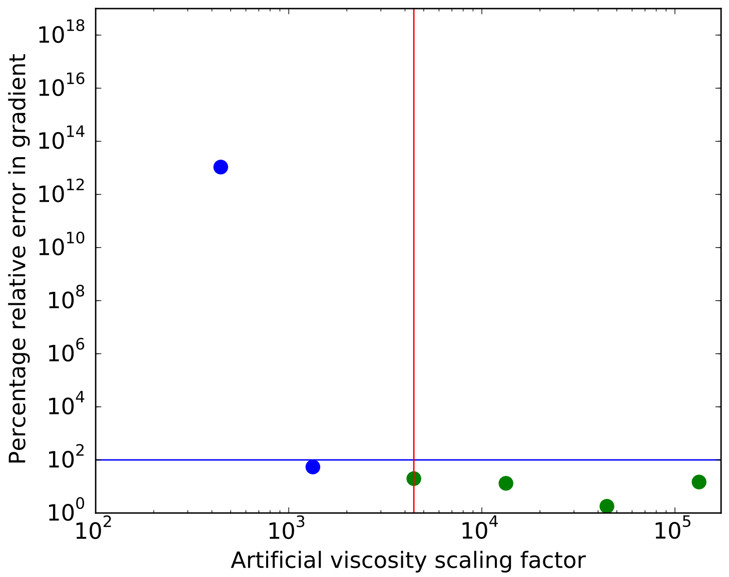


Figure 3-10: Plot showing the relative error in the design objective gradient computed using adjoint solutions with different artificial viscosity scaling factors (η) for the turbine vane problem. The blue dots denote unstable adjoint solutions and green dots denote stable adjoint solutions. The blue line denotes 100% relative error and the red line denotes the minimum η for which the adjoint solution is stable.

solution has a small magnitude in the trailing edge region and remains bounded in the domain of the flow problem. Additionally, there are bump-like structures in the density adjoint solution on the suction side of the vane surface.

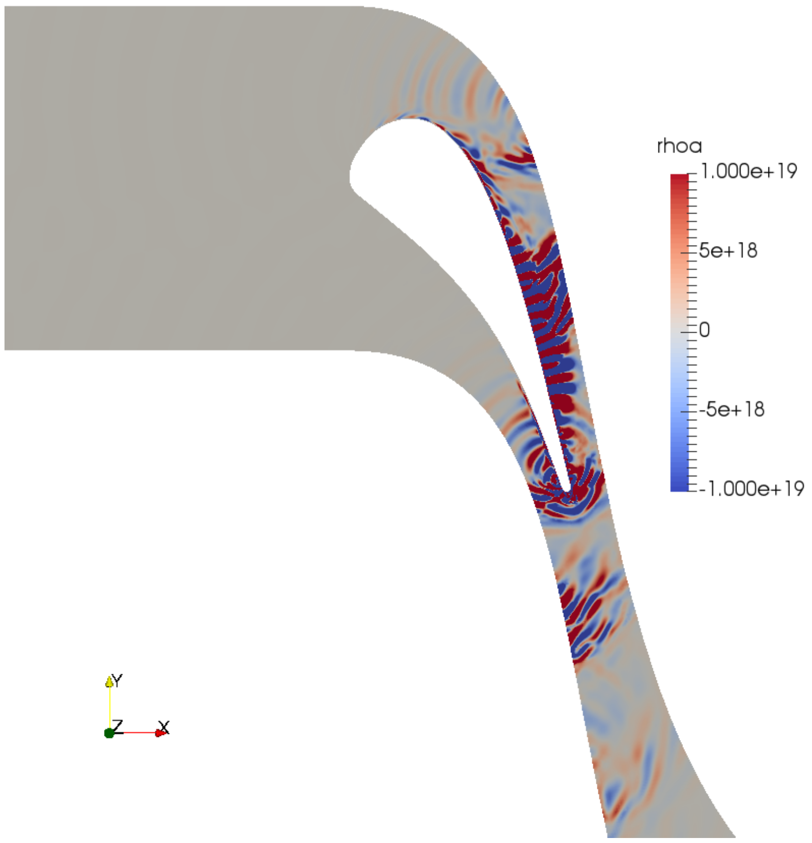


Figure 3-11: Density adjoint solution (units: m^3 s/kg) at $t = 1$ time unit without any artificial viscosity.

From these two cases it can be seen that the viscosity stabilized adjoint method provides a reasonably accurate design objective gradient with relative error as low as 1%. The optimal value of the scaling factor η for which the design objective gradient has the smallest relative error is generally the same or close to the minimum value of η for which the adjoint solution is stabilized. Furthermore, there is a large range of values of the scaling factor for which the design objective gradient has the right order of magnitude and sign. While there is no theoretical guarantee that the gradient obtained using the viscosity stabilized adjoint method is accurate, an estimate of the error in the gradient due to the artificial viscosity can be obtained. The details of the error estimate are discussed in Chapter 4.

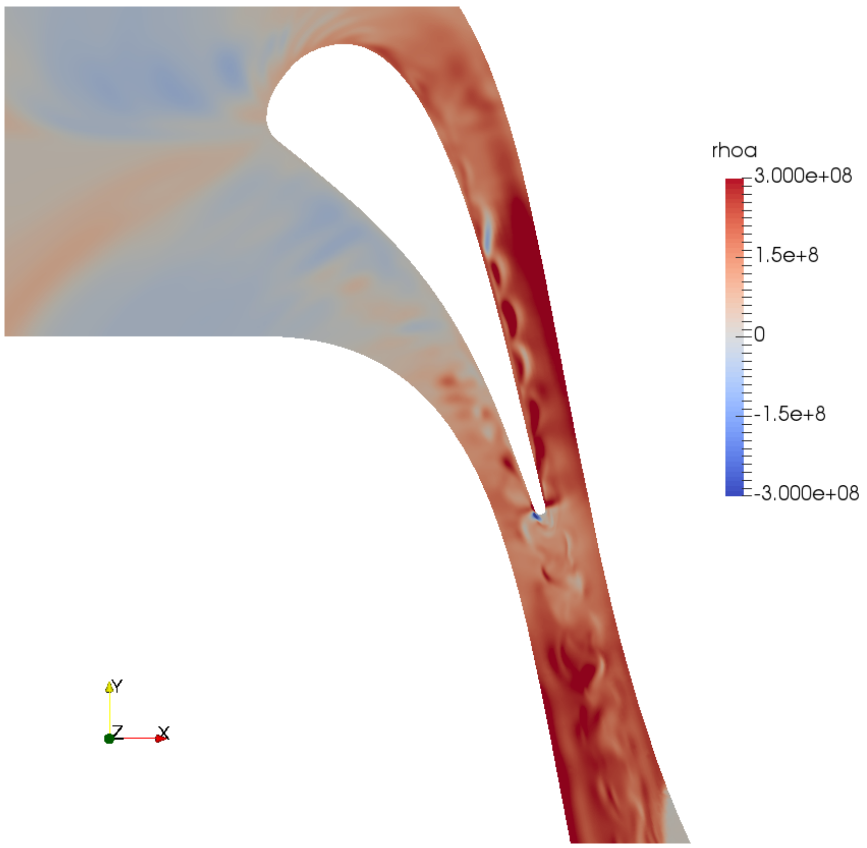


Figure 3-12: Density adjoint solution (units: $\mathrm{m}^3\ \mathrm{s/kg}$) at $t = 1$ time unit with artificial viscosity using $\eta = 4,444$.

The optimal η is different for the two test cases. It is much higher for the transonic flow over a turbine vane in comparison to the subsonic flow over a cylinder. This is primarily due to the higher Reynolds number of the turbine vane case, which causes faster divergence of the adjoint solution. This suggests that there might exist an alternative scaling for the artificial viscosity that depends on the convective scales of the flow problem.

3.3.1 Comparison of different symmetrizing transformations

The relative effectiveness of the various artificial viscosity fields, constructed using the Turkel, Abarbanel and Hughes symmetrizing transformations, in stabilizing the adjoint solution is tested on the transonic flow over a turbine vane flow problem described in Sections 1.4.3 and 2.1. These artificial viscosity fields are additionally compared with a uniform artificial viscosity field, which is an artificial viscosity field that has the same value over the entire domain of the flow problem.

The performance metric for the comparison is the growth of the adjoint energy (as defined in Equation 2.41) as a function of time. In order to ensure a fair comparison between the various artificial viscosity fields, the scaling factor for each of the fields is defined as

$$\eta = \frac{C\rho_r \|\lambda_1\|_\infty}{\mu_r \|\lambda_1\|_2} \tag{3.18}$$

where $C = 10^{-3}\text{m}^2/\text{s}$. The above equation implies that the spatial L_2 norm of each of the artificial viscosity fields is the same constant.

Figure 3-13 shows the adjoint energy as a function of time for the different artificial viscosity fields. The figure shows that the artificial viscosity field derived using any of the symmetrizing transformations is better in dissipating the adjoint energy than the uniform artificial viscosity field. Among the different symmetrizing transformations, the Turkel artificial viscosity field is the most effective in stabilizing the adjoint solution. This is primarily because of the higher contrast in the magnitude of the Turkel viscosity field, as can be seen when comparing Figure 2-7 with Figures 2-6 and 2-8. The Turkel viscosity field has a much lower minimum and higher maximum. Consequently, using it results in the addition of a higher amount of viscosity in the turbulent boundary layer and wake, which are the primary regions of divergence of the adjoint solution for the turbine vane problem. On the other hand, even though the Hughes viscosity field has a lower effectiveness in reducing the exponential growth of adjoint energy, it is more theoretically sound as it is derived using an entropy based symmetrizing transformation for the compressible Navier-Stokes equations.

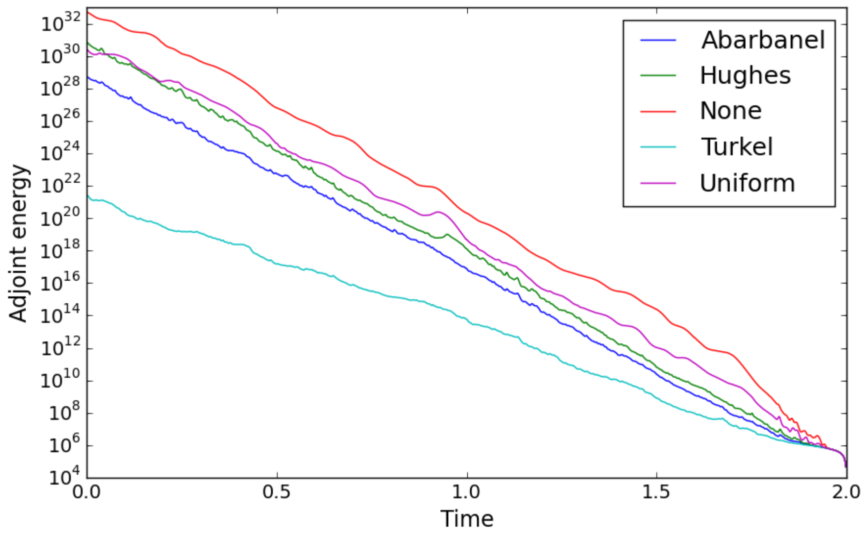


Figure 3-13: Plot showing the growth of adjoint energy (units: kg s/m^3) using different artificial viscosity fields: Abarbanel, Turkel, Hughes, Uniform and No viscosity (None) as a function of time (represented by time units).

Chapter 4

Error analysis of adjoint solution with artificial viscosity and application to non-intrusive least squares shadowing

In this chapter, error analysis of the viscosity stabilized adjoint method is performed. Additionally, the method is combined with another adjoint method for chaotic systems, the non-intrusive least squares shadowing method, which utilizes the shadowing theorem for dynamical systems. Section 4.1 discusses basics of chaotic dynamical systems and the shadowing theorem. Section 4.2 describes error analysis of the viscosity stabilized adjoint method when the modified adjoint equations form either a stable or unstable system. Finally, Section 4.3 demonstrates the application of the viscosity stabilization method to non-intrusive least squares shadowing.

4.1 Shadowing theorem

The discretized flow equations with a design parameter θ can be represented as a parameterized invertible nonlinear map $\mathbf{f} : \mathbb{R}^M \times \mathbb{R} \rightarrow \mathbb{R}^M$, where M is the dimension of the finite-dimensional discretized flow solution. The map is assumed to be ergodic, which means that the long-time behavior of the system is independent of the initial condition. The evolution from one time step, n, to the next, $n+1$, can be represented as

$$\boldsymbol{p_{n+1}} = \mathbf{f}(\boldsymbol{p_n}, \theta), \forall n \in (-\infty, \infty) \tag{4.1}$$

with an appropriate fixed initial condition $\boldsymbol{p_0} \in \mathbb{R}^M$, where $\boldsymbol{p_n} \in \mathbb{R}^M$ is the solution and $\theta \in \mathbb{R}$ is a parameter of the system. The above equations are also known as the primal system. While in Section 1.2, the flow solution is defined for time $t \in [0, \infty)$, the discrete system is defined for both positive and negative time in this section, that is, $n \in (-\infty, \infty)$. The discretized design objective function, averaged from $n = -\infty$ to ∞, is

$$\bar{J} = \lim_{N\to\infty} \frac{1}{2N} \sum_{n=-N}^{N} J(\boldsymbol{p_n}, \theta) \tag{4.2}$$

The adjoint equations can be derived by forming the following Lagrangian

$$\frac{\partial \bar{J}}{\partial \theta} = \lim_{N\to\infty} \frac{1}{2N} \sum_{n=-N}^{N} (\frac{\partial J(\boldsymbol{p_n}, \theta)}{\partial \theta} + \frac{\partial J(\boldsymbol{p_n}, \theta)}{\partial \boldsymbol{p_n}} \frac{\partial \boldsymbol{p_n}}{\partial \theta} - \boldsymbol{\hat{p}_n}^T (\frac{\partial \boldsymbol{p_{n+1}}}{\partial \theta} - \frac{\partial \mathbf{f}(\boldsymbol{p_n}, \theta)}{\partial \theta} - \frac{\partial \mathbf{f}(\boldsymbol{p_n}, \theta)}{\partial \boldsymbol{p_n}} \frac{\partial \boldsymbol{p_n}}{\partial \theta})) \tag{4.3}$$

Consequently, the design objective gradient is given by

$$\frac{\partial \bar{J}}{\partial \theta} = \lim_{N\to\infty} \frac{1}{2N} \sum_{n=-N}^{N} (\frac{\partial J(\boldsymbol{p_n}, \theta)}{\partial \theta} + \boldsymbol{\hat{p}_n}^T \frac{\partial \mathbf{f}(\boldsymbol{p_n}, \theta)}{\partial \theta}) \tag{4.4}$$

if the following adjoint equations are satisfied

$$\boldsymbol{\hat{p}_{n-1}} = \frac{\partial \mathbf{f}(\boldsymbol{p_n}, \theta)}{\partial \boldsymbol{p_n}}^T \boldsymbol{\hat{p}_n} + \frac{\partial J(\boldsymbol{p_n}, \theta)}{\partial \boldsymbol{p_n}}, \forall n \in (-\infty, \infty) \tag{4.5}$$

where $\boldsymbol{\hat{p}_n}$ is the adjoint solution. The above equations form the adjoint system. The adjoint operator or map can be written as $\boldsymbol{T}(\boldsymbol{p_n}) = \frac{\partial \mathbf{f}(\boldsymbol{p_n},\theta)}{\partial \boldsymbol{p_n}}^T$. An initial condition, $\boldsymbol{\hat{p}_0}$, can be prescribed for the system, but as the system is ergodic, any vector with a bounded norm can be chosen.

An M-dimensional ergodic system has at most M Lyapunov exponents, $\lambda_1^e, \lambda_2^e, ..., \lambda_M^e$, given by the logarithms of the eigenvalues of the following matrix [77]

$$\lim_{n\to\infty} (\boldsymbol{T^n}(\boldsymbol{p_0})[\boldsymbol{T^n}(\boldsymbol{p_0})]^T)^{1/(2n)} \tag{4.6}$$

where $\boldsymbol{T^m}(\boldsymbol{p_n}) = \prod_{i=n-m+1}^{n} \boldsymbol{T}(\boldsymbol{p_i})$ with $m > 0$ and $\boldsymbol{T^0}(\boldsymbol{p_n}) = \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix of size M. For chaotic systems, the magnitude of the adjoint solution diverges to infinity when simulated backwards in time for almost all initial conditions $\boldsymbol{\hat{p}_0}$. This is due to the fact that a chaotic system has at least one positive Lyapunov exponent [23].

A special type of ergodic dynamical systems is uniformly hyperbolic dynamical systems. For a uniformly hyperbolic dynamical system, each Lyapunov exponent has a corresponding adjoint Lyapunov covariant vector, $\boldsymbol{\hat{p}^i}(\boldsymbol{p})$ at every $\boldsymbol{p}$ of all trajectories of Equation 4.1 with arbitrary initial conditions, that satisfies the following property [34, 55]

$$\frac{\log\|\boldsymbol{T^m}(\boldsymbol{p})\boldsymbol{\hat{p}^i}(\boldsymbol{p})\|_2}{n-m} \xrightarrow{m\to\infty} \lambda_i \tag{4.7}$$

for all $n \in (-\infty, \infty)$. λ_i is the i^{th} Lyapunov exponent and $\boldsymbol{\hat{p}^i_n}$ is the i^{th} adjoint Lyapunov covariant vector. When there are M distinct Lyapunov exponents, the M adjoint Lyapunov covariant vectors form a basis of $\mathbb{R}^M$. Ginelli [34] described a QR factorization based algorithm for numerically estimating the Lyapunov exponents and adjoint Lyapunov covariant vectors of an ergodic system. This algorithm is used in this thesis for computing the Lyapunov quantities.

Additionally, uniformly hyperbolic dynamical systems satisfy the following property [34], for all $\boldsymbol{p} \in \mathbb{R}^M$, there exists a splitting of $\mathbb{R}^M$, $E^+(\boldsymbol{p})$ and $E^-(\boldsymbol{p})$, such that for all $\boldsymbol{\hat{p}}^- \in E^-(\boldsymbol{p})$ and $\boldsymbol{\hat{p}}^+ \in E^+(\boldsymbol{p})$,

$$\begin{aligned} \|\boldsymbol{T^m}(\boldsymbol{p})\boldsymbol{\hat{p}}^-\|_2 \le C_2\delta^{-m}\|\boldsymbol{\hat{p}}^-\|_2 \\ \|\boldsymbol{T^{-m}}(\boldsymbol{p})\boldsymbol{\hat{p}}^+\|_2 \le C_2\delta^{-m}\|\boldsymbol{\hat{p}}^+\|_2 \end{aligned} \tag{4.8}$$

for a positive constant C_2, $\delta > 1$ and all $m < n$, where $\boldsymbol{T^{-m}}(\boldsymbol{p_n}) = \prod_{i=n+m}^{n+1} \boldsymbol{T^{-1}}(\boldsymbol{p_i}) = [\boldsymbol{T^m}(\boldsymbol{p_{n+m}})]^{-1}$ with $m > 0$. $\|\cdot\|_2$ is the L_2 norm for a vector. $E^+(\boldsymbol{p})$ and $E^-(\boldsymbol{p})$ form a splitting of $\mathbb{R}^M$ and are known as the unstable and stable adjoint subspaces at $\boldsymbol{\hat{p}_n}$ respectively. These subspaces have the following properties, $\boldsymbol{T}(\boldsymbol{p_n})E^-(\boldsymbol{p_n}) = E^-(\boldsymbol{p_{n-1}})$ and $\boldsymbol{T}(\boldsymbol{p_n})E^+(\boldsymbol{p_n}) = E^+(\boldsymbol{p_{n-1}})$.

The divergence of adjoint solution for chaotic systems corrupts the accuracy of the gradient obtained from it. The divergence behavior is shown for almost all initial conditions, $\boldsymbol{\hat{p}_0}$. However, there is a possibility of the adjoint solution staying bounded for a particular choice of an initial condition. Mathematically, this means that L_2 norm of the adjoint solution, $\boldsymbol{\hat{p}_n}$ from Equation 4.5, is bounded for all $n \in (-\infty, \infty)$.

$$\|\boldsymbol{\hat{p}_n}\|_2 \leq C_1 \tag{4.9}$$

where C_1 is a positive constant. Such an adjoint solution is known as the adjoint shadowing solution. The shadowing theorem guarantees that if the adjoint map satisfies the hyperbolicity condition, then the adjoint map has a unique adjoint shadowing solution [44] and it satisfies Equation 4.5. An adjoint shadowing solution can be used to obtain the true design objective gradient. This is the premise on which the Least Squares Shadowing (LSS) method operates [117]. It finds a numerical approximation to the adjoint shadowing solution, which then provides an estimate the design objective gradient. The adjoint shadowing solution is found by solving a finite-time interval approximation to the following least squares optimization problem

$$\begin{aligned} &\min_{\boldsymbol{\hat{p}_n}} \lim_{N\to\infty} \frac{1}{2N} \sum_{i=-\infty}^{\infty} \boldsymbol{\hat{p}_n}^T \boldsymbol{\hat{p}_n} \\ s.t.\ &\boldsymbol{\hat{p}_{n-1}} = \boldsymbol{T}(\boldsymbol{p_n})\boldsymbol{\hat{p}_n} + \frac{\partial J(\boldsymbol{p_n}, \theta)}{\partial \boldsymbol{p_n}}, \forall n \in (-\infty, \infty) \end{aligned} \tag{4.10}$$

4.2 Error analysis

If the artificial viscosity in the viscosity stabilized adjoint method is large enough to stabilize the adjoint solution, the modified (or damped) adjoint equations form a stable damped adjoint system. The design objective gradient estimated by solving this damped system will have some error with respect to the true gradient. In this section, error analysis of the gradient from the damped system is performed by comparing it with the gradient obtained from the adjoint shadowing solution.

If the artificial viscosity is not enough to stabilize the adjoint solution, then Equation 1.26 cannot be used to estimate the design objective gradient. Instead, an LSS-like method can be applied to the unstable damped adjoint system to find a modified adjoint shadowing solution. In this section, error analysis of the gradient computed from the modified adjoint shadowing solution is performed by comparing it with gradient obtained from the shadowing solution of the unmodified adjoint system.

4.2.1 Stable damped adjoint system

The discrete form of the damped adjoint system can be represented in the following form

$$\hat{\boldsymbol{p}}'_{\boldsymbol{n-1}} = \boldsymbol{T'}(\boldsymbol{p_n})\hat{\boldsymbol{p}}'_{\boldsymbol{n}} + \frac{\partial J(\boldsymbol{p_n}, \theta)}{\partial \boldsymbol{p_n}} \tag{4.11}$$

where $\hat{\boldsymbol{p}}'_{\boldsymbol{n}}$ is the solution of the damped adjoint system, which stays bounded for all $n \in (-\infty, \infty)$. The chaotic system is assumed to have a fixed trajectory, $\boldsymbol{p_n}, \forall n \in (-\infty, \infty)$. $\boldsymbol{T'}(\boldsymbol{p_n})$ is the modified or damped linear operator which characterizes the dynamics of this system. As the modification to the adjoint system should preserve the linearity of the operator, $\boldsymbol{T'}(\boldsymbol{p_n})$ is restricted to be of the form $\boldsymbol{T}(\boldsymbol{p_n}) + \epsilon \boldsymbol{W}(\boldsymbol{p_n})$, where $\epsilon \in \mathbb{R}^+$ is the damping parameter (analogous to scaling factor in Section 3.1) and $\boldsymbol{W}(\boldsymbol{p_n}) : \mathbb{R}^M \to \mathbb{R}^M$ is a linear operator, representing the type of damping or damping specification. $\boldsymbol{T}(\boldsymbol{p_n})$ is defined in the same manner as in Section 4.1.

As the system is stable, all the Lyapunov exponents of this operator are non-positive. The system is assumed to be contractive which means that all the exponents are negative and it satisfies the following property

$$\|\boldsymbol{T'^n}(\boldsymbol{p_n})\hat{\boldsymbol{p}}'_{\boldsymbol{n}}\|_2 \leq C_3 \delta_s^{-n} \|\hat{\boldsymbol{p}}'_{\boldsymbol{n}}\|_2 \tag{4.12}$$

for almost any vector $\hat{\boldsymbol{p}}'_{\boldsymbol{n}}$. C_3 is a positive constant and $\delta_s > 1$. The chaotic system formed by the discretized flow equations is assumed to be uniformly hyperbolic, which means that it has an adjoint shadowing solution $\hat{\boldsymbol{p}}_{\boldsymbol{n}}$ which satisfies Equation 4.5. Given a trajectory $\{\boldsymbol{p_n},\ \forall n \in (-\infty, \infty)\}$, the following simplified notation is used: $\boldsymbol{T_n} = \boldsymbol{T}(\boldsymbol{p_n}), \boldsymbol{T'_n} = \boldsymbol{T'}(\boldsymbol{p_n})$, Subtracting Equation 4.5 from 4.11 the following equation is obtained

$$\boldsymbol{\hat{p}'_{n-1}} - \boldsymbol{\hat{p}_{n-1}} = (\boldsymbol{T_n} + \epsilon \boldsymbol{W_n})\boldsymbol{\hat{p}'_n} - \boldsymbol{T_n}\boldsymbol{\hat{p}_n} \tag{4.13}$$

which can be rewritten as,

$$\boldsymbol{\hat{p}'_{n-1}} - \boldsymbol{\hat{p}_{n-1}} = (\boldsymbol{T_n} + \epsilon \boldsymbol{W_n})(\boldsymbol{\hat{p}'_n} - \boldsymbol{\hat{p}_n}) + \epsilon \boldsymbol{W_n}\boldsymbol{\hat{p}_n} \tag{4.14}$$

Let $\boldsymbol{e_n} = \boldsymbol{\hat{p}'_n} - \boldsymbol{\hat{p}_n}$ and $\boldsymbol{b_n} = \epsilon \boldsymbol{W_n}\boldsymbol{\hat{p}_n}$. Hence, the linear system in the above equation can be rewritten as

$$\boldsymbol{e_{n-1}} = \boldsymbol{T'_n}\boldsymbol{e_n} + \boldsymbol{b_n} \tag{4.15}$$

which can be expanded to form

$$\boldsymbol{e_{n-1}} = \boldsymbol{T'^{n-1}_{n+m}}\boldsymbol{e_{n+m}} + \sum_{i=0}^{m} \boldsymbol{T'^{n-1}_{n+i-1}}\boldsymbol{b_{n+i}} \tag{4.16}$$

using $\boldsymbol{T'^{m}_{n}} = \boldsymbol{T'^{n-m}}(\boldsymbol{p_n})$. Taking the vector L_2 norm of both sides of the above equation and using the triangle inequality for norms

$$\|\boldsymbol{e_{n-1}}\|_2 = \|\boldsymbol{T'^{n-1}_{n+m}}\boldsymbol{e_{n+m}}\|_2 + \sum_{i=0}^{m} \|\boldsymbol{T'^{n-1}_{n+i-1}}\boldsymbol{b_{n+i}}\|_2 \tag{4.17}$$

Using the contractive property of the operator $\boldsymbol{T'^{m}_{n}}$, the following equations can be obtained

$$\|\boldsymbol{e_{n-1}}\|_2 \leq C_3 \delta_s^{-m-1} \|\boldsymbol{e_{n+m}}\|_2 + \sum_{i=0}^{m} C_3 \delta_s^{-i} \|\boldsymbol{b_{n+i}}\|_2 \tag{4.18}$$

Using the max norm, $\|\boldsymbol{b}\| = \max_{n \in (-\infty,\infty)} \|\boldsymbol{b_n}\|_2$,

$$\|\boldsymbol{e_{n+1}}\|_2 \leq C_3 \delta_s^{-m-1} \|\boldsymbol{e_{n+m}}\|_2 + C_3 \frac{\delta_s - \delta_s^{-m}}{\delta_s - 1} \|\boldsymbol{b}\| \tag{4.19}$$

Taking the limit $m \to \infty$ of both sides of the above equation, the first term in the right hand side of the above equation becomes 0 using the fact that $\|\boldsymbol{e_n}\|_2$ is bounded (as $\|\hat{\boldsymbol{p}}_{\boldsymbol{n}}\|_2$ and $\|\hat{\boldsymbol{p}}'_{\boldsymbol{n}}\|$ are bounded by a positive constant)

$$\|\boldsymbol{e}_{\boldsymbol{n+1}}\|_2 \leq C_3 \frac{\delta_s}{\delta_s - 1} \|\boldsymbol{b}\| \tag{4.20}$$

Using properties of norms, $\|b\| \leq \epsilon \|W\| \|v\|$. Hence, the above equation can be rewritten as

$$\|\boldsymbol{e}\| \leq \epsilon C_3 \frac{\delta_s}{\delta_s - 1} \|\boldsymbol{W}\| \|\hat{\boldsymbol{p}}\| \tag{4.21}$$

The above equation bounds the error in the adjoint solution of the damped system with respect to the adjoint shadowing solution.

The adjoint solution of the damped system is used to compute an estimate of the design objective gradient ($\frac{\partial \bar{J}}{\partial \theta}$)

$$\frac{\partial \bar{J}'}{\partial \theta} = \lim_{N \to \infty} \frac{1}{2N} \sum_{n=-N}^{N} \left(\frac{\partial J(\boldsymbol{p}_{\boldsymbol{n}}, \theta)}{\partial \theta} + [\hat{\boldsymbol{p}}'_{\boldsymbol{n}}]^T \frac{\partial \mathbf{f}(\boldsymbol{p}_{\boldsymbol{n}}, \theta)}{\partial \theta}\right) \tag{4.22}$$

Subtracting Equation 4.4 from 4.22

$$\frac{\partial \bar{J}'}{\partial \theta} - \frac{\partial \bar{J}}{\partial \theta} = \lim_{N \to \infty} \frac{1}{2N} \sum_{n=-N}^{N} (\hat{\boldsymbol{p}}'_{\boldsymbol{n}} - \hat{\boldsymbol{p}}_{\boldsymbol{n}})^T \frac{\partial \mathbf{f}(\boldsymbol{p}_{\boldsymbol{n}}, \theta)}{\partial \theta} \tag{4.23}$$

Taking absolute values of both sides and applying Cauchy-Schwarz inequality

$$\left|\frac{\partial \bar{J}'}{\partial \theta} - \frac{\partial \bar{J}}{\partial \theta}\right| \leq \lim_{N \to \infty} \frac{1}{2N} \sum_{n=-N}^{N} \|\hat{\boldsymbol{p}}'_{\boldsymbol{n}} - \hat{\boldsymbol{p}}_{\boldsymbol{n}}\|_2 \left\|\frac{\partial \mathbf{f}(\boldsymbol{p}_{\boldsymbol{n}}, \theta)}{\partial \theta}\right\|_2 \tag{4.24}$$

Using the max norm, computing the sum and taking the limit

$$\left|\frac{\partial \bar{J}'}{\partial \theta} - \frac{\partial \bar{J}}{\partial \theta}\right| \leq \|\boldsymbol{e}\| \left\|\frac{\partial \mathbf{f}}{\partial \theta}\right\| \tag{4.25}$$

Substituting Equation 4.21 into the above equation

$$\left|\frac{\partial \bar{J}'}{\partial \theta} - \frac{\partial \bar{J}}{\partial \theta}\right| \leq \epsilon C_3 \frac{\delta_s}{\delta_s - 1} \left\|\frac{\partial \mathbf{f}}{\partial \theta}\right\| \|\boldsymbol{W}\| \|\hat{\boldsymbol{p}}\| \tag{4.26}$$

Hence, the error in the design objective gradient from a stable damped adjoint system with respect to the gradient provided by the adjoint shadowing solution can be bounded by the size of damping parameter. Note that when the gradient is approximated using an adjoint solution over a finite time interval, an additional error is introduced. Methods for quantifying this error are discussed in Section 5.1.2.

4.2.2 Unstable damped adjoint systems

Similar to the stable damped adjoint system, the discrete form of the unstable damped adjoint system can be represented in the following form

$$\hat{\boldsymbol{p}}'_{\boldsymbol{n-1}} = \boldsymbol{T}'_{\boldsymbol{n}}\hat{\boldsymbol{p}}'_{\boldsymbol{n}} + \frac{\partial J(\boldsymbol{p_n}, \theta)}{\partial \boldsymbol{p_n}} \tag{4.27}$$

where $\hat{\boldsymbol{p}}'_{\boldsymbol{n}}$ is the adjoint shadowing solution of the damped system, which stays bounded for all $n \in (-\infty, \infty)$. Such an adjoint solution exists if, in addition to the chaotic system formed by the discretized flow equations (whose adjoint shadowing solution is denoted by $\hat{\boldsymbol{p}}_{\boldsymbol{n}}$), the damped adjoint operator is assumed to be uniformly hyperbolic (with C_4 replacing C_2 and δ_u replacing δ in Equation 4.8). The modified linear operator $\boldsymbol{T}'_{\boldsymbol{n}}$ has the same form as in the previous section, $\boldsymbol{T}'_{\boldsymbol{n}} = \boldsymbol{T_n} + \epsilon \boldsymbol{W_n}$. But, for the unstable damped system, one or more of the Lyapunov exponents are positive.

Subtracting Equation 4.5 from 4.11 the following equation is obtained

$$\boldsymbol{e_{n-1}} = \boldsymbol{T}'_{\boldsymbol{n}}\boldsymbol{e_n} + \boldsymbol{b_n} \tag{4.28}$$

where $\boldsymbol{e_n} = \hat{\boldsymbol{p}}'_{\boldsymbol{n}} - \hat{\boldsymbol{p}}_{\boldsymbol{n}}$ and $\boldsymbol{b_n} = \epsilon \boldsymbol{W_n}\hat{\boldsymbol{p}}_{\boldsymbol{n}}$. Using the hyperbolicity assumption, $\boldsymbol{e_n}$ and $\boldsymbol{b_n}$ can be split into two components, one residing in the unstable adjoint space E_n^+ of $\boldsymbol{T_n}$ and the other in the stable adjoint space. For $\boldsymbol{e_n}$, these are $\boldsymbol{e}_{\boldsymbol{n}}^{+}$ and $\boldsymbol{e}_{\boldsymbol{n}}^{-}$ respectively. The above equation becomes

$$\boldsymbol{e}_{\boldsymbol{n-1}}^{+} + \boldsymbol{e}_{\boldsymbol{n-1}}^{-} = \boldsymbol{T}'_{\boldsymbol{n}}(\boldsymbol{e}_{\boldsymbol{n}}^{+} + \boldsymbol{e}_{\boldsymbol{n}}^{-}) + \boldsymbol{b}_{\boldsymbol{n}}^{+} + \boldsymbol{b}_{\boldsymbol{n}}^{-} \tag{4.29}$$

Rearranging the terms,

$$(\boldsymbol{e^+_{n-1}} - \boldsymbol{T'_n e^+_n} - \boldsymbol{b^+_n}) = -(\boldsymbol{e^-_{n-1}} - \boldsymbol{T'_n e^-_n} - \boldsymbol{b^-_n}) \tag{4.30}$$

The terms on the left and right hand side exist in orthogonal linear subspaces. For them to the equal the only valid explanation is for both the sides to be zero. Hence,

$$\begin{aligned} \boldsymbol{e^+_{n-1}} &= \boldsymbol{T'_n e^+_n} + \boldsymbol{b^+_n} \\ \boldsymbol{e^-_{n-1}} &= \boldsymbol{T'_n e^-_n} + \boldsymbol{b^-_n} \end{aligned} \tag{4.31}$$

Expanding $\boldsymbol{e^-_n}$

$$\boldsymbol{e^-_{n-1}} = \boldsymbol{T'^{n-1}_{n+m} e^-_{n+m}} + \sum_{i=0}^{m} \boldsymbol{T'^{n-1}_{n+i-1} b_{n+i}}- \tag{4.32}$$

Taking the L_2 norm of both sides and using the triangle inequality

$$\|\boldsymbol{e^-_{n-1}}\|_2 = \|\boldsymbol{T'^{n-1}_{n+m} e^-_{n+m}}\|_2 + \sum_{i=0}^{m} \|\boldsymbol{T'^{n-1}_{n+i-1} b^-_{n+i}}\|_2 \tag{4.33}$$

Applying the stable adjoint subspace property of hyperbolicity

$$\|\boldsymbol{e^-_{n-1}}\|_2 \leq C_4 \delta_u^{-m-1} \|\boldsymbol{e^-_{n+m}}\|_2 + \sum_{i=0}^{m} C_4 \delta_u^{-i} \|\boldsymbol{b^-_{n+i}}\|_2 \tag{4.34}$$

Rewriting the unstable subspace part of Equation 4.31 by taking inverse of the damped adjoint operator

$$\boldsymbol{e^+_n} = [\boldsymbol{T'_n}]^{-1} \boldsymbol{e^+_{n-1}} - [\boldsymbol{T'_n}]^{-1} \boldsymbol{b^+_n} \tag{4.35}$$

Expanding $\boldsymbol{e^+_n}$

$$\boldsymbol{e^+_n} = [\boldsymbol{T'^{n-m-1}_n}]^{-1} \boldsymbol{e^+_{n-m-1}} - \sum_{i=0}^{m} [\boldsymbol{T'^{n-i-1}_n}]^{-1} \boldsymbol{b^+_{n-i}} \tag{4.36}$$

Taking L_2 norm of both sides, applying triangle inequality of norms and using unstable subspace property of hyperbolicity, the above equation becomes

$$\|\boldsymbol{e}^+_{\boldsymbol{n}}\|_2 \leq C_4 \delta_u^{-m-1} \|\boldsymbol{e}^+_{\boldsymbol{n-m-1}}\|_2 + \sum_{i=0}^{m} C_4 \delta_u^{-i-1} \|\boldsymbol{b}^+_{\boldsymbol{n-i}}\|_2 \tag{4.37}$$

Using triangle inequality,

$$\|\boldsymbol{e}_{\boldsymbol{n}}\|_2 \leq \|\boldsymbol{e}^+_{\boldsymbol{n}}\|_2 + \|\boldsymbol{e}^-_{\boldsymbol{n}}\|_2 \tag{4.38}$$

Substituting Equations 4.34 (after incrementing the time index by 1) and 4.37 into the above equation

$$\|\boldsymbol{e}_{\boldsymbol{n}}\|_2 \leq C_4 \delta_u^{-m-1} (\|\boldsymbol{e}^+_{\boldsymbol{n-m-1}}\|_2 + \|\boldsymbol{e}^-_{\boldsymbol{n+m+1}}\|_2) + \sum_{i=0}^{m} C_4 \delta_u^{-i} (\frac{1}{\delta_u} \|\boldsymbol{b}^+_{\boldsymbol{n-i}}\|_2 + \|\boldsymbol{b}^-_{\boldsymbol{n+i}}\|_2) \tag{4.39}$$

As $\boldsymbol{e}^+_{\boldsymbol{n}}$ and $\boldsymbol{e}^-_{\boldsymbol{n}}$ are orthogonal to each other and $\|\boldsymbol{e}_{\boldsymbol{n}}\|_2$ is bounded, $\|\boldsymbol{e}^+_{\boldsymbol{n}}\|_2$ and $\|\boldsymbol{e}^-_{\boldsymbol{n}}\|$ are also bounded. Using this fact and taking limit $m \to \infty$ of both sides of the above equation

$$\|\boldsymbol{e}_{\boldsymbol{n}}\|_2 \leq \lim_{m\to\infty} \sum_{i=0}^{m} C_4 \delta_u^{-i} (\frac{1}{\delta_u} \|\boldsymbol{b}^+_{\boldsymbol{n-i}}\|_2 + \|\boldsymbol{b}^-_{\boldsymbol{n+i}}\|_2) \tag{4.40}$$

Using max norm on the two terms in the round brackets of the right hand side of the above equation and taking the limit on computing the sum

$$\|\boldsymbol{e}_{\boldsymbol{n}}\|_2 \leq C_4 \frac{\delta_u}{\delta_u - 1} (\frac{1}{\delta_u} \|\boldsymbol{b}^+\| + \|\boldsymbol{b}^-\|) \tag{4.41}$$

Using $2\|\boldsymbol{b}\| \geq (\frac{1}{\delta_u}\|\boldsymbol{b}^+\| + \|\boldsymbol{b}^-\|)$

$$\|\boldsymbol{e}_{\boldsymbol{n}}\|_2 \leq C_4 \frac{2\delta_u}{\delta_u - 1} \|\boldsymbol{b}\| \tag{4.42}$$

The above equation bounds the error in the adjoint shadowing solution due to the damping. Using this result the following equation can be obtained

$$|\frac{\partial \bar{J}'}{\partial \theta} - \frac{\partial \bar{J}}{\partial \theta}| \leq \epsilon C_4 \frac{2\delta_u}{\delta_u - 1} \|\frac{\partial \mathbf{f}}{\partial \theta}\| \|\boldsymbol{W}\| \|\hat{\boldsymbol{p}}\| \tag{4.43}$$

Hence, the error in the design objective gradient obtained from shadowing solution of an unstable damped adjoint system, with respect to the gradient provided by the shadowing solution of the unmodified adjoint system can be bounded by the size of the damping parameter. Note that when the gradient is approximated using an adjoint shadowing solution over a finite time interval, an additional error is introduced. Methods for quantifying this error using time series analysis techniques are discussed in Section 5.1.2.

4.2.3 A-posteriori error estimate

Equations 4.26 and 4.43 can be used as an a-posteriori error estimate for the gradient obtained using the adjoint solutions for the stable and unstable damped adjoint systems respectively. In practice, as the adjoint solutions are computed over a finite time interval, only approximations of the terms in the error estimate can be obtained.

The terms ϵ and $\|\boldsymbol{W}\|$ can be estimated from the damping specification. $\|\frac{\partial \mathbf{f}}{\partial \theta}\|$ can be estimated from the primal solution. $\|\hat{\boldsymbol{p}}\|$ can be approximated using $\|\hat{\boldsymbol{p}}'\|$, which can be estimated from the adjoint solution of the damped system. As the adjoint solution solution needs to be computed before estimating this term, the error estimate is a-posteriori. Finally, C_3 and δ_s for the stable damped system and C_4 and δ_u for the unstable damped system can be estimated from numerical simulating the adjoint system for various initial conditions in the stable and unstable adjoint subspaces. Particularly, the algorithm for estimating C_4 and δ_u is the following

1. Use $e^{\lambda^e_{min}}$ as an approximation to δ_u, where $\lambda^e_{min} = \min\limits_{i\in[1,M]} |\lambda^e_i|$. λ^e_{min} is the Lyapunov exponent with the smallest magnitude for the unstable damped system.

2. Compute a single trajectory of each of the adjoint covariant Lyapunov vectors, $\hat{\boldsymbol{p}}'^{\boldsymbol{i}}_{\boldsymbol{n}}$ for $i = 1$ to $i = M$, of the damped system by applying the operator $\boldsymbol{T}'_{\boldsymbol{n}}$ from $n = 0$ to $n = -m$ ($m > 0$) for the stable vectors and by applying the operator $\boldsymbol{T}'^{-1}_{\boldsymbol{n}}$ from $n = 1$ to $n = m + 1$ for the unstable vectors. The adjoint covariant Lyapunov vectors can be initialized at $n = 0$ using Ginelli's algorithm [34] for estimating the Lyapunov vectors. The value of m should be chosen such that

the trajectory is computed over multiple time units of the system. This ensures that the trajectory sufficiently explores the phase space of the system.

3. Assuming that the order of the Lyapunov exponents is from the largest to the smallest, compute C_4 using

$$\begin{aligned} C_{4,u} &= \max_{i=1..M_u, n=1...m+1} ([\frac{\|\boldsymbol{T'^{0}_{n}}]^{-1}\boldsymbol{\hat{p}'^{i}_{0}}\|_2 \delta_u^{-n}}{\|\boldsymbol{\hat{p}'^{i}_{n}}\|_2}) \\ C_{4,s} &= \max_{i=M_u+1...M, n=1...m+1} (\frac{\|\boldsymbol{T'^{-n}_{0}}\boldsymbol{\hat{p}'^{i}_{0}}\|_2 \delta_u^{-n}}{\|\boldsymbol{\hat{p}'^{i}_{n}}\|_2}) \\ C_4 &= \max\{C_{4,u}, C_{4,s}\} \end{aligned} \tag{4.44}$$

The error estimate for the unstable damped adjoint system is tested on a design objective for the Lorenz system. The details of this test case are discussed in Section 1.4.1. The damping specification is

$$\boldsymbol{W_n} = -\boldsymbol{I} \tag{4.45}$$

The adjoint shadowing solution is computed over a finite time interval whose length is 20 time units using LSS. Figure 4-1 shows that the error in the gradient of the design objective computed from the adjoint shadowing solution of the unstable damped adjoint system is close to the error estimate for all values of the damping parameter ϵ. In addition, the minimum value of the damping parameter for which the damped adjoint system is stable is $\epsilon = 1$. At this value of ϵ, the relative error in the gradient is low and approximately equal to 10%.

The error in the gradient is computed with respect to the gradient obtained from an adjoint shadowing solution of the unmodified adjoint system using LSS over a finite time interval whose length is 20 units. The terms C_4 and δ_u in the error estimate as computed for the discretized Lorenz system are $C_4 = 2.5$ and $\delta_u = 2.63$. These terms are computed for the unmodified adjoint Lorenz system over a period of 10 time units. The adjoint Lyapunov covariant vector of the discretized system whose Lyapunov exponent is close to 0 is ignored while computing these terms. This vector is the neutral component of the adjoint Lorenz system and it is not a part of either the stable or unstable adjoint subspace. A more rigorous error estimate for the Lorenz system can be found by performing the error analysis due to damping for continuous nonlinear systems or flows.

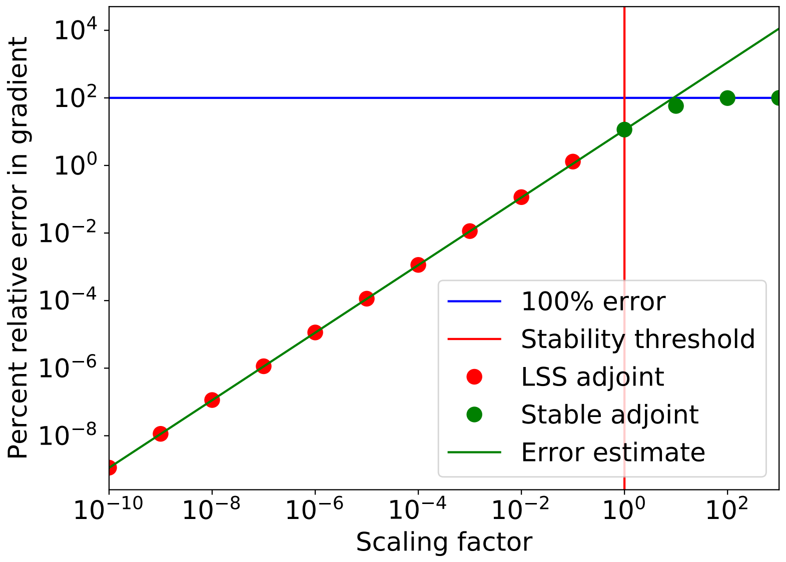


Figure 4-1: Plot of the error in the gradient of the unstable damped adjoint system as a function of the size of the damping parameter (scaling factor) ϵ. The red dots denote the adjoint shadowing solution obtained using NILSAS with a damped adjoint system and the green dots denote the stable adjoint solution obtained using a damped adjoint system. The blue denotes 100% relative error in the design objective gradient, the red line denotes the minimum damping parameter for which the damped adjoint system is stable and the green line denotes the error estimate

4.3 Non-intrusive least squares shadowing

In the viscosity stabilized adjoint method, if the damping is not sufficient to stabilize the adjoint solution, then it is necessary to compute the design objective gradient using an adjoint shadowing solution of the damped system. While the damping parameter can be increased until the adjoint solution is stable, high damping parameters might result in a large error in the design objective gradient, as demonstrated by Equation 4.26. The adjoint version of the non-intrusive least squares shadowing (NILSS) is a recent algorithm for finding the adjoint shadowing solution in a relatively inexpensive manner [74, 73] compared to its predecessor, the adjoint LSS method. The viscosity stabilized adjoint method helps the NILSS method by reducing its computational cost.

The cost of the LSS algorithm scales with the number of time steps in the finite-time interval over which the adjoint shadowing solution is computed and the dimension of the system. For the high-dimensional chaotic systems formed by discretized flow equations on fine meshes, LSS is computationally intractable. The LSS algorithm searches for the M-dimensional shadowing solution in a time interval with N time steps. In contrast, the NILSS method approximates it as an adjoint solution (satisfying Equation 4.5) that is orthogonal to the subspace spanned by the unstable adjoint Lyapunov covariant vectors (corresponding to positive Lyapunov exponents). An adjoint solution in the stable subspace spanned by the stable adjoint Lyapunov covariant vectors (corresponding to negative Lyapunov exponents) is guaranteed to have a bounded norm. The NILSS algorithm finds this solution by solving an M_u-dimensional optimization problem, where M_u is the number of positive Lyapunov exponents. This results in a significant reduction in computational cost in comparison to LSS as $M_u \ll M$ for many chaotic systems.

For a fluid dynamics system, the size of the discretized flow equations (M) can be arbitrarily increased by refining the mesh. This does not mean that the number of positive Lyapunov exponents (M_u) becomes unbounded. In fact, for 2-dimensional incompressible Navier-Stokes equations, M_u is known to be bounded by a positive constant and is proportional to the total energy dissipation in a large volume domain[91, 60]. There is no upper bound result for the number of positive exponents for 3-dimensional compressible turbulence. Even so, in a number of compressible flow problems, it has been observed that M_u initially increases on refining the mesh, but stays below an upper bound on further refinement [13, 26].

Adding artificial viscosity to the adjoint equations reduces the rate of growth of adjoint energy. By stabilizing the adjoint solution, it has the potential to decrease the number of positive of positive exponents of the damped system. Hence, reducing the computational cost of NILSS. From Section 4.2, it is observed that the error introduced in the design objective gradient due to the damping is directly proportional to the amount of artificial viscosity added to the adjoint system.

4.3.1 Adjoint NILSS algorithm

The adjoint version of the NILSS algorithm (also known as NILSAS) represents the adjoint shadowing solution as

$$\hat{\boldsymbol{p}}_{\boldsymbol{n}} = \hat{\boldsymbol{p}}^{\boldsymbol{0}}_{\boldsymbol{n}} + \boldsymbol{Y}_{\boldsymbol{n}}\boldsymbol{a} \tag{4.46}$$

for all $n \in (0, N)$, where $\hat{\boldsymbol{p}}^{\boldsymbol{0}}_{\boldsymbol{n}}$ is a particular diverging adjoint solution (satisfying Equation 4.5) and $\boldsymbol{a}$ is a coefficient vector of length M_u, where M_u is the dimension of the chaotic system. $\boldsymbol{Y}_{\boldsymbol{n}}$ is the matrix formed by the M_u adjoint Lyapunov covariant vectors, $\hat{\boldsymbol{p}}^{\boldsymbol{i}}_{\boldsymbol{n}}$, each satisfying Equation 4.7, as columns. NILSAS tries to find the optimal coefficient vector, $\boldsymbol{a}^*$, that minimizes the vector L_2 norm of $\hat{\boldsymbol{p}}_{\boldsymbol{n}}$ over all time. It does this by solving the following optimization problem

$$\boldsymbol{a}^* = \underset{\boldsymbol{a}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=0}^{N} \hat{\boldsymbol{p}}^T_{\boldsymbol{n}} \hat{\boldsymbol{p}}_{\boldsymbol{n}} \tag{4.47}$$

where the adjoint shadowing solution, $\hat{\boldsymbol{p}}^{\boldsymbol{*}}_{\boldsymbol{n}}$, is given by $\hat{\boldsymbol{p}}^{\boldsymbol{*}}_{\boldsymbol{n}} = \hat{\boldsymbol{p}}^{\boldsymbol{0}}_{\boldsymbol{n}} + \boldsymbol{Y}_{\boldsymbol{n}}\boldsymbol{a}^*$. NILSAS assumes that the chaotic dynamical system is ergodic and uniformly hyperbolic.

The divergence of the adjoint solution, $\hat{\boldsymbol{p}}^{\boldsymbol{0}}_{\boldsymbol{n}}$, makes it difficult to solve the optimization problem due to overflow and round-off error in floating-point representations. In order to circumvent this issue, NILSAS divides the finite time interval into K segments, $N_0, ..., N_{K-1}$, where $N_i = [n_i, n_{i+1}]$, $n_i = i\frac{N}{K}$ and N is divisible by K. The adjoint solution and adjoint covariant vectors are denoted separately for each segment N_i at n, by $\hat{\boldsymbol{p}}^{\boldsymbol{0}}_{\boldsymbol{i,n}}$ and $\boldsymbol{Y}_{\boldsymbol{i,n}}$ respectively, where the first index in the subscript denotes the segment index. The segment length is chosen in such a way that the adjoint solution and vectors stay bounded and do not grow by more than an order of magnitude in the segment. Additionally, there is a separate coefficient vector $\boldsymbol{a}_i$ of length M_u for each segment. A numerical implementation of the NILSAS algorithm uses approximate adjoint Lyapunov covariant vectors and rescales $\hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i,n}}$ and $\boldsymbol{Y_{i,n}}$ at the end of each segment, N_i. Continuity of the adjoint shadowing solution at segment intersections is ensured using

$$\hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i-1,n_i}} + \boldsymbol{Y_{i-1,n_i}} \boldsymbol{a}_{i-1} = \hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i,n_i}} + \boldsymbol{Y_{i,n_i}} \boldsymbol{a}_i \tag{4.48}$$

The above equation introduces $K-1$ equality constraints for the optimization problem. Finally, there is another equality constraint given by the following equation

$$\sum_{i=1}^{K} [\mathbf{f}(\boldsymbol{p_{n_i}}, \theta)^T \boldsymbol{Y_{i-1,n_i}} \boldsymbol{a}_i + \mathbf{f}(\boldsymbol{p_{n_i}}, \theta)^T \hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i-1,n_i}}] = 0 \tag{4.49}$$

which is necessary in order for NILSAS to determine the coefficient of the adjoint Lyapunov covariant vector whose exponent is very close to 0. Such a vector is obtained for discrete ergodic systems which are time discretizations of continuous systems.

The complete NILSAS algorithm is given below

1. Solve the primal system from $n = 0$ to $n = N$.

2. Set initial conditions for $\hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{K-1,N}}$ and $\boldsymbol{Y_{K-1,N}}$,

$$\begin{aligned} \hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{K-1,N}} &= \mathbf{0} \\ \boldsymbol{Y_{K-1,N}} &= \boldsymbol{Q}_K \end{aligned} \tag{4.50}$$

   where $\boldsymbol{Q}_K$ is a random orthonormal matrix.

3. For each segment i, beginning from $i = K-1$ up to $i = 0$

   (a) Solve the adjoint system, given by Equation 4.5, for n_1 steps with $\hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i,n_{i+1}}}$ as the initial condition in order to obtain the vector $\hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i,n_i}}$.

   (b) Using the adjoint operator $\boldsymbol{T^m_n}$ compute the matrix $\boldsymbol{Y_{i,n_i}}$

$$\boldsymbol{Y_{i,n_i}} = \boldsymbol{T^{n_i}_{n_{i+1}}} \boldsymbol{Y_{i,n_{i+1}}} \tag{4.51}$$

(c) Rescale the vector $\hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i},\boldsymbol{n_i}}$ and the matrix $\boldsymbol{Y_{i,n_i}}$ in order to obtain $\hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i-1},\boldsymbol{n_i}}$ and $\boldsymbol{Y_{i-1,n_i}}$ for the next segment

$$\begin{aligned}\boldsymbol{Q_i R_i} &= \boldsymbol{Y_{i,n_i}} \\ \boldsymbol{Y_{i-1,n_i}} &= \boldsymbol{Q}_i \\ \boldsymbol{g}_i &= \boldsymbol{Q_i^T} \hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i},\boldsymbol{n_i}} \\ \hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i-1},\boldsymbol{n_i}} &= \hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i},\boldsymbol{n_i}} - \boldsymbol{Q_i g}_i\end{aligned} \tag{4.52}$$

4. Solve the following optimization problem

$$\begin{aligned}&\min_{\boldsymbol{a}_0,\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{K-1}} \sum_{i=0}^{K-1} \boldsymbol{a}_i^T \boldsymbol{a}_i \\ &\boldsymbol{a}_{i-1} = \boldsymbol{R_i a}_i + \boldsymbol{g}_i, \forall i = 1, 2, ..., K-1 \\ &\sum_{i=1}^{K} [\mathbf{f}(\boldsymbol{p_{n_i}}, \theta)^T \boldsymbol{Y_{i-1,n_i} a}_i + \mathbf{f}(\boldsymbol{p_{n_i}}, \theta)^T \hat{\boldsymbol{p}}^{\mathbf{0}}_{\boldsymbol{i-1},\boldsymbol{n_i}}] = 0\end{aligned} \tag{4.53}$$

using the Lagrange multiplier method. The required solution can be obtained by solving the following linear system of equations

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{B}^T \\ \boldsymbol{B} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{c} \\ \hat{\boldsymbol{c}} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{d} \end{pmatrix} \tag{4.54}$$

where the matrix $\boldsymbol{B}$ and the vectors $\boldsymbol{c}, \boldsymbol{d}$ are given by

$$
\begin{aligned}
\boldsymbol{B} &= \begin{pmatrix} \boldsymbol{I} & -\boldsymbol{R_1} & & & \\ & \boldsymbol{I} & -\boldsymbol{R_2} & & \\ & & \ddots & \ddots & \\ & & & \boldsymbol{I} & \boldsymbol{R_{K-1}} \\ \mathbf{f}(\boldsymbol{p_{n_1}}, \theta)^T \boldsymbol{Y_{0,n_1}} & & \cdots & & \mathbf{f}(\boldsymbol{p_{n_K}}, \theta)^T \boldsymbol{Y_{K-1,n_K}} \end{pmatrix} \\
\boldsymbol{c} &= \begin{pmatrix} \boldsymbol{a}_0 \\ \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_{K-1} \end{pmatrix} \\
\boldsymbol{d} &= \begin{pmatrix} \boldsymbol{1} \\ \vdots \\ \boldsymbol{K-1} \\ -\sum_{i=1}^{K} \mathbf{f}(\boldsymbol{p_{n_i}}, \theta)^T \boldsymbol{\hat{p}^0_{i-1,n_i}} \end{pmatrix}
\end{aligned}
\tag{4.55}
$$

and $\boldsymbol{\hat{c}}$ is the Lagrange multiplier vector for the equality constraints in the optimization problem.

5. Using the optimal coefficient vectors, $\boldsymbol{a}_i^*$, the adjoint shadowing solution, $\boldsymbol{\hat{p}^*_n}$, can be constructed. The shadowing solution can then provide the design objective gradient using Equation 4.4.

4.3.2 Results

The NILSAS algorithm is tested on subsonic flow over a cylinder. The details of this problem are described in Section 1.4.2. The design objective is time-averaged drag over the surface of the cylinder and the gradient is computed with respect to the inlet Mach number of the flow.

The algorithm is applied to the adjoint equations for the discretized flow system and the viscosity stabilized adjoint equations. Plots of the Lyapunov exponents computed for the unmodified and modified adjoint systems are shown in Figure 4-2. The scaling factor for the viscosity stabilized adjoint equations is $\eta = 10$. This value of η reduces the exponential divergence of the adjoint solution, but is not sufficient to completely stabilize it over the course of the simulation. Artificial viscosity is added to the conservative adjoint equations using the Turkel symmetrizing transformation. 20 exponents are computed for the unmodified equations and 10 exponents are computed for the viscosity stabilized equations. The number of segments is $K = 200$ and the number of time steps per segment is $\frac{N}{K} = 500$ for both cases. As can be seen from the figures, the modified adjoint equations with the artificial viscosity has a fewer number of positive Lyapunov exponents ($M_u = 3$) compared with the unmodified adjoint equations ($M_u = 9$).

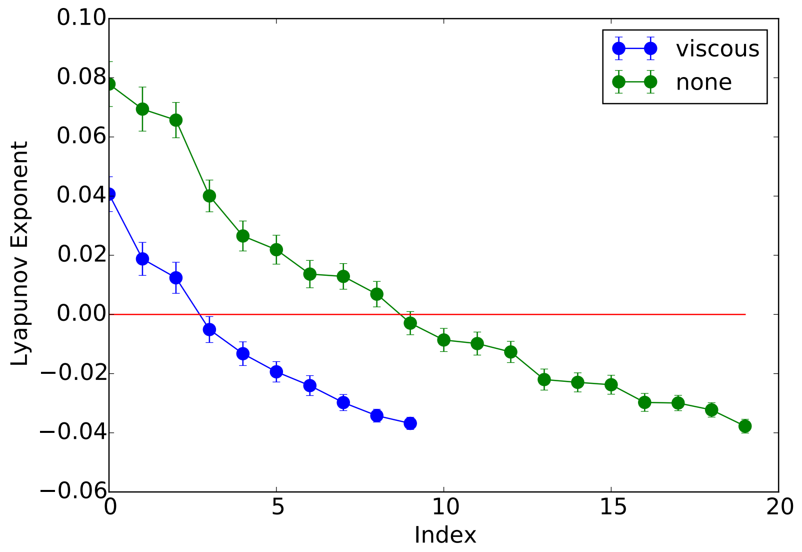


Figure 4-2: Plot showing the positive and a few negative Lyapunov exponents for the unmodified adjoint equations (none in figure legend) and the viscosity stabilized adjoint equations with scaling factor $\eta = 10$ (viscous in figure legend) for the subsonic flow over a cylinder.

The design objective gradient obtained from the shadowing solution provided by NILSAS on the unmodified adjoint system is 20.8 ± 3.5, whereas the gradient obtained from the shadowing solution provided by NILSAS on the viscosity stabilized adjoint system is 17.8 ± 3.2. The relative error of both gradients is less than 30%. The uncertainty in the gradient is computed using time series analysis techniques discussed in Section 5.1.2. In this example, the viscosity stabilized adjoint method reduced the cost of the NILSAS algorithm to 50% while providing a gradient with a similar level of accuracy. Figure 4-3 shows the relative error in the design objective gradient as a function of η for the viscosity stabilized adjoint method and the NILSAS method with artificial viscosity. The relative error is defined in the same manner as in Section 3.3.

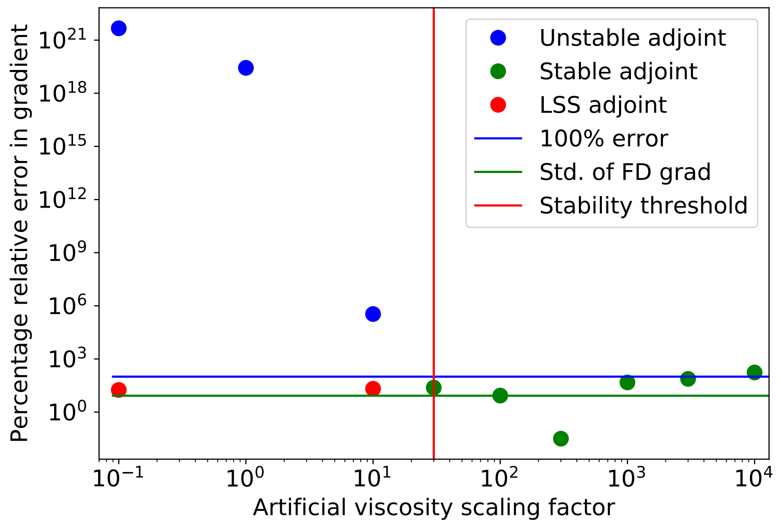


Figure 4-3: Plot showing the relative error in the design objective gradient computed using adjoint solutions with different artificial viscosity scaling factors (η) for the flow over cylinder problem. The blue dots denote the unstable adjoint solutions computed using the viscosity stabilized adjoint method, red dots denote adjoint shadowing solutions computed using NILSAS with the viscosity stabilized adjoint method and the green dots denote the stable adjoint solutions computed using the viscosity stabilized adjoint method. The value of the artificial viscosity scaling factor for the 2 left most dots (1 blue and 1 red dot) is $\eta = 0$ instead of $\eta = 0.1$ as shown in the figure. The blue line denotes 100% relative error and the green line denotes the standard deviation of the finite difference gradient estimate.

Hence, the viscosity stabilized adjoint method can be applied to NILSAS in order to reduce it's computational cost for turbulent flow problems that have a large number of positive Lyapunov exponents. This method is especially useful when the minimum artificial viscosity scaling factor for which the adjoint solution is stable results in a high design objective gradient error (as estimated by Equation 4.26) that is unacceptable for design optimization algorithms.

Chapter 5

Adjoint-based design optimization using large eddy simulations

In this chapter, an adjoint-based design optimization tool, using the artificial viscosity based adjoint method and a Bayesian optimization algorithm, is applied to design the trailing edge shape of a gas turbine nozzle guide vane. The design objective is a linearly weighted combination of the time and mass-flow averaged stagnation pressure loss coefficient downstream of the vane and the time-averaged Nusselt number over the trailing edge surface of the vane. The shape of the trailing edge is parameterized using a linear combination of 5 convex designs. The optimization is performed on the Argonne National Lab Mira supercomputer and a small-scale CPU and GPU based compute cluster. Section 5.1 describes the parameterization procedure of the trailing edge shape for the flow over the turbine vane. Section 5.2 details the artificial viscosity based adjoint method, a modified Bayesian optimization algorithm and its application to a 2-dimensional optimization problem objective function based on the Rastrigrin function. Finally, Section 5.3 discusses results from the adjoint-based design optimization and demonstrates a comparison between gradient-based and derivative-free Bayesian optimization.

5.1 Flow over turbine vane

The shape of a nozzle guide vane in a gas turbine engine plays a significant role in the work extraction efficiency of the turbine. Lower work extraction from the fluid results from a higher pressure loss due to an unfavorable shape of the vane. Additionally, the shape impacts the amount of heat transfer from the hot gas to the vane surface, hence, determining the cooling fluid requirements for the vane [42]. An optimally designed vane can lead to notable fuel and repair cost savings and increased longevity of the vane. The reference (baseline) shape for the nozzle guide vane used for the shape optimization is designed by researchers at the Von Karman Institute (VKI) [5].

5.1.1 Flow physics

Understanding the physics of the flow over the turbine vane is crucial for constructing the optimization problem. The boundary layer development on the two sides of the vane and turbulent mixing in the wake lead to substantial drop in the stagnation pressure of the fluid. These phenomena discourage the use of blunt trailing edge for the vane as they can lead to earlier separation and higher pressure loss.

There is a large increase in the heat transfer coefficient on the suction side due to the transition of the boundary layer from laminar to turbulent. This behavior rules out the use of sharp trailing edge for the vane as the material of the vane cannot simultaneously sustain high temperatures and high stress for prolonged time periods.

The thickness, development and separation location of the boundary layers and the characteristics of the turbulent wake are greatly influenced by the shape of the vane near the trailing edge. Hence, the design optimization restricts shape parameterization of the vane to the trailing edge in order to lower the dimension of the design search space and maintain the enhancement potential of the candidate designs.

The details of the numerics for this flow problem are described in Sections 1.3 and 1.4.3.

5.1.2 Design objective

The design objective for the trailing edge shape optimization is a linear combination of the stagnation pressure loss downstream of the vane and heat transfer near the trailing edge of the vane. The stagnation pressure loss is represented by the infinite time-averaged and mass flow-averaged stagnation pressure loss coefficient ($\bar{p}_l$) defined by the design objective in Equation 1.11, which is restated below

$$\begin{aligned}
\bar{p}_l &= \frac{\bar{p}_{t,l}}{p_{t,in}} \\
\bar{p}_{t,l} &= \lim_{t_e \to \infty} \frac{1}{t_e} \int_0^{t_e} \frac{\int_{S_p} \rho_p u_n (p_{t,in} - p_{t,p})\, dS_p}{\int_{S_p} \rho_p u_n \, dS_p} \, dt \\
p_{t,p} &= p_p (1 + \frac{\gamma - 1}{2} M_p^2)^{\frac{\gamma}{\gamma-1}} \\
p_{t,in} &= p_{ex} (1 + \frac{\gamma - 1}{2} M_{is}^2)^{\frac{\gamma}{\gamma-1}}
\end{aligned} \tag{5.1}$$

A visualization of the time history of the instantaneous pressure loss is shown in Figure 5-1. The heat transfer is represented by the Nusselt number (Nu).

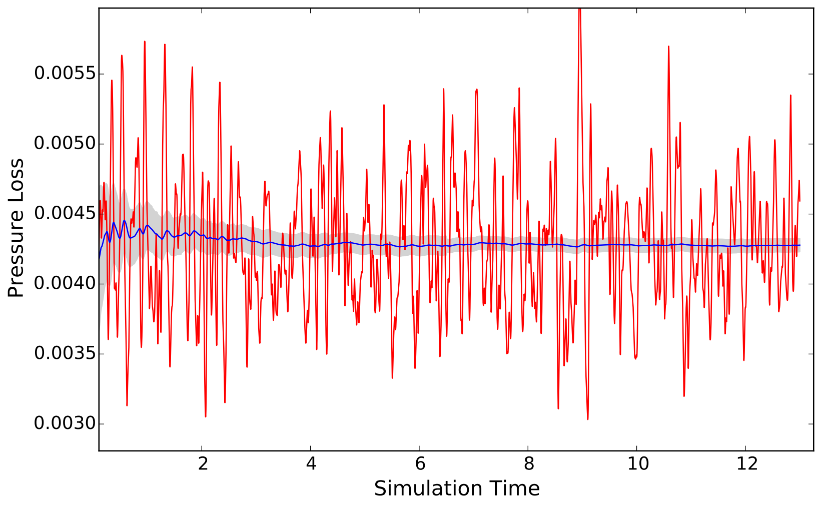


Figure 5-1: Scaled instantaneous pressure loss coefficient ($a(\bar{p}_l)$) plotted as a function of time (represented by time units). The blue time series denotes the cumulative mean and the gray shaded area denotes a single standard deviation of the sample mean. The procedure for time averaging is discussed in Section 5.1.2.

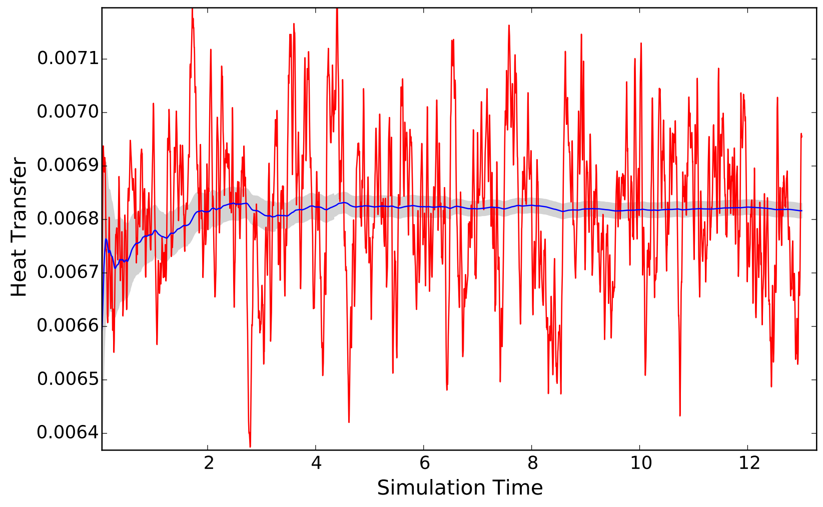


Figure 5-2: Scaled instantaneous Nusselt number ($b(Nu)$) plotted as a function of time (represented by time units). The blue time series denotes the cumulative mean and the gray shaded area denotes a single standard deviation of the sample mean. The procedure for time averaging is discussed in Section 5.1.2.

$$Nu = \frac{\bar{h}L}{k} \tag{5.2}$$

where k is the thermal conductivity at $T = 300\,K$, $k = 0.028\,\frac{W}{mK}$, L is the trailing edge radius for the baseline case, $0.0105\,c_l$, and $\bar{h}$ is the time-averaged heat transfer coefficient over a part of the vane starting from $0.414\,c_l$ downstream of the leading edge in the direction of the inflow and leading up to the tip of the trailing edge. The equation for $\bar{h}$ is

$$\bar{h} = \lim_{t_e \to \infty} \frac{1}{S_v t_e \Delta T} \int_0^{t_e} \int_{S_v} k \frac{\partial T}{\partial n}\, dS_v\, dt, \tag{5.3}$$

where S_v is the surface area. $\Delta T = 120\,K$ is the temperature difference between the surface of the blade and the stagnation temperature of the flow. A visualization of the time history of the instantaneous heat transfer is shown in Figure 5-2 The design objective for the optimization is a linear combination of two quantities, the Nusselt number and pressure loss coefficient, and is given by

$$\bar{J} = a(Nu) + b(\bar{p}_l) \tag{5.4}$$

where $a = 5 \times 10^{-4}$ and $b = 0.4$. The values for a and b are chosen such that the contribution of both Nu and $\bar{p}_l$ to the sum is equal for the baseline case.

As the fluid flow solution is obtained for a finite time, the infinite time-averages in Equations 5.1 and 5.3 are approximated using a finite time-average. The length of the time averaging interval is chosen to provide a statistically converged estimate of the infinite time average. A converged estimate is an estimate that has a standard deviation (or standard error) that is less than 10% of the magnitude of the estimate itself. Through a numerical investigation it is found to be equal to $N = 6$ time units. This interval is sufficient to obtain the design objective with approximately 5% standard error relative to the estimate.

A procedure is required to compute an estimate for the finite time-average from an instantaneous time history of the design objective. The finite time-average can be modeled as a random variable. The mean estimate of this random variable can be computed using a sample average, $\tilde{J}_N$.

$$\tilde{J}_N = \frac{1}{N} \sum_{n=N_0}^{N} J_n \tag{5.5}$$

where J_n represents the instantaneous design objective at time step n. The time averaging is started after an initial transition period to allow for any transient effects in the time history to settle down into a statistical steady state. The transition period is determined by finding the time it takes for a running time-average to lie within 1 standard deviation of the full interval time-average. For the design objective of the trailing edge shape optimization problem, this period is approximately $N_0 = 1$ time units. It is large enough to encompass the transition period for different trailing edge shapes.

The standard deviation of the mean estimate (or sample mean) provides an indication of the amount of error in the finite time-average approximation of the design objective. The standard statistical formula for computing the standard error of a sample mean cannot be used as the instantaneous values of the design objective are not independent and form a correlated time series. Oliver (2014) [76] suggested using data fitted autoregressive models to get the correlation function for the time series. This approach is adopted in this thesis. Autoregressive models can be written in the following form

$$J_n = \sum_{i=1}^{p} a_i J_{n-i} + \epsilon_n \tag{5.6}$$

where a_i are the constant coefficients of the autoregressive model, p is the order of the model and ϵ_n are independent, identically distributed normal random variables, $\epsilon_n \sim N(0, \sigma^2)$. The coefficients of the model are determined using the Burg estimation algorithm with the CIC criterion to decide the model order. Even though the actual model for the time series may not belong to the class of linear stochastic models, the variance computed using the model's correlation function can be utilized to provide a reasonable estimate of the variance of the sample mean.

$$Var(\tilde{J}_N) \approx \frac{Var(J_n)}{N_d} \tag{5.7}$$

$$N_d = \frac{N}{1 + 2\sum_{k=1}^{\infty} \rho(k)} \tag{5.8}$$

where $\rho(k)$ is the autocorrelation between k time steps and N_d is the effective sample size.

5.1.3 Design parameterization

Parameterizing the trailing edge of the turbine vane is a challenging task. For the shapes to be valid, they have to satisfy a convexity condition about the chord line. Such a condition is required in order to ensure that there are no undulations or obstructions in the surface of the vane which can increase flow instability or cause back flow. The convexity condition imposes a nonlinear constraint on the shape parameters, increasing the complexity of the design optimization problem.

One way to get around the nonlinear constraint is to parameterize the trailing edge of a 2-dimensional turbine vane as a linear combination of 5 trailing edge shapes, with the weights serving as the parameters. If the candidate trailing edge shapes are convex, then they form a reduced basis of all convex shapes [96], ensuring that the parameterized trailing edge shape is convex. The linear combination is implemented as a weighted average of the 2-dimensional coordinates of the basis shapes. If the coordinates of the basis shape j are denoted by x_i^j, then the coordinates of a new shape are given by

$$x_i(\boldsymbol{\alpha}) = \sum_{j=1}^{5} \alpha_j x_i^j \tag{5.9}$$

The sum of the weights (α_j) is equal to 1. Hence, the weights α_i of the 5 basis shapes satisfy the following constraint,

$$\sum_{j=1}^{5} \alpha_j = 1 \tag{5.10}$$

The above equality constraint can be transformed into an inequality constraint by eliminating one of the α_i variables. This results in a formation of a 4-dimensional parameterization of the trailing edge shape with the following inequality constraint,

$$\sum_{j=1}^{4} \alpha_j \leq 1 \tag{5.11}$$

The 5 basis shapes are chosen such that the parameterized shapes explore a large subset of convex shapes. They are shown in Figure 5-3. One of the basis shapes is from the baseline turbine vane design from VKI. The 5 basis shapes are linearly independent, meaning that there are no two sets of parameters $\boldsymbol{\alpha}$ that generate the same shape.

Evaluating the design objective and gradient for a trailing edge shape requires the generation of a new mesh for obtaining the flow and adjoint solutions. The mesh corresponding to the parameterized shape is formed by perturbing the mesh of the baseline design. The mesh generation process starts by projecting the nodes of the mesh of the baseline design onto the parameterized design by minimizing distance between the nodes of the respective designs. Once projected, a displacement vector is computed from the nodes of the baseline mesh nodes on the surface of the vane to parameterized mesh nodes. This displacement vector on the vane surface is propagated

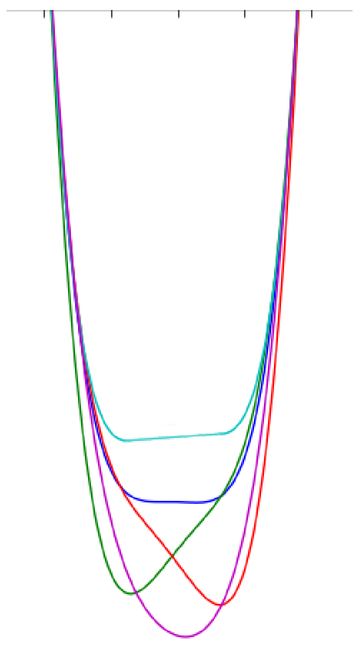


Figure 5-3: Visualization of the 5 basis shapes

through the remaining mesh using a linear elasticity model for the mesh nodes [22]. This model can be used under the assumption that the magnitude of the displacement vector is small. The linear elasticity equations are

$$\nabla \cdot \boldsymbol{\sigma} = 0 \tag{5.12}$$

$$\boldsymbol{\sigma} = \lambda \epsilon_{ii} \boldsymbol{I} + 2\mu \boldsymbol{\epsilon} \tag{5.13}$$

$$\boldsymbol{\epsilon} = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) \tag{5.14}$$

where $\boldsymbol{u}$ is the displacement, λ and μ are constants which govern the elasticity of the mesh nodes for propagating the displacement. The linear elasticity equations are solved using the boundary condition $\boldsymbol{u} = \boldsymbol{c}$ for the mesh nodes on the surface of the vane, where $\boldsymbol{c}$ is the displacement vector of the parameterized design from the baseline design.

The gradient for the new trailing edge shape is evaluated using the viscosity stabilized adjoint method. The function that maps the parameters of the shape to the location of the parameterized mesh nodes is difficult to differentiate. Hence, instead of directly providing the gradient of the design objective with respect to the shape parameters, the adjoint method provides the gradient with respect to discrete mesh fields like cell centers, mesh normals, et cetera, which are typically used to represent the mesh in finite volume methods. These gradients are denoted by $\frac{\partial J}{\partial m_i}$. The finite difference method is used to obtain the gradient with respect to the parameters of the shape ($\frac{\partial J}{\partial \alpha_j}$).

$$\frac{\partial J}{\partial \alpha_j} \approx \frac{\partial J}{\partial m_i} \frac{(m_i(\boldsymbol{\alpha} + \boldsymbol{\epsilon_j}) - m_i(\boldsymbol{\alpha}))}{\epsilon} \tag{5.15}$$

where $\boldsymbol{\epsilon_j}$ is a zero vector with the same dimension as $\boldsymbol{\alpha}$ and with the j^{th} entry set to $\epsilon = 10^{-5}$. Hence, for each design objective gradient evaluation, a set of 4 meshes are generated by perturbing each shape parameter by ϵ separately. The corresponding perturbations in the discrete mesh fields for each of the shape parameters are utilized to obtain $\frac{(m_i(\boldsymbol{\alpha}+\boldsymbol{\epsilon_j})-m_i(\boldsymbol{\alpha}))}{\epsilon}$. Multiplying this quantity with $\frac{\partial J}{\partial m_i}$ obtained from the adjoint method and summing over all the indices i provides the design objective gradient.

5.2 Adjoint-based design optimization

The adjoint method for physics-based numerical simulations is a widely used tool to accelerate the engineering design optimization process. Adjoint-based design optimization tools utilize the adjoint method to efficiently obtain the gradient of the design objective function with respect to multiple design parameters. The adjoint method requires a single additional solution field (known as the adjoint solution) for computing the gradients, in comparison to the finite difference method of computing gradients, which requires n additional solution fields for n design parameters [31].

5.2.1 Viscosity stabilized adjoint method

The viscosity stabilized adjoint method, described in Section 3.2 is used to obtain the design objective gradient. This method damps the divergence of the adjoint solution for turbulent fluid flows, while maintaining reasonable accuracy of the design objective gradients. It requires manual tuning of a parameter known as the artificial viscosity scaling factor (η). After obtaining multiple adjoint solutions with different values of the scaling factor , the optimal value of η for flow over the turbine vane is determined to be $\eta = 4,444$. The adjoint solution is obtained over a finite time interval whose length is 0.5 time units. This value is the minimum value of η for which the adjoint solution is stable for the baseline design of the trailing edge of the turbine vane. The value of η is kept constant for different parameterized designs of the trailing edge of the turbine vane. Empirically, it has been observed that the same value of η stabilizes the adjoint solution for different designs.

5.2.2 Optimization method

Design objectives in LES are often defined as infinite time averages. But, in practice, as the simulations are performed for a finite time, there is a sampling error associated with the objective function and gradient evaluations [75]. This error makes it difficult for the optimization algorithm to know whether a new design point with a lower objective value is actually better than the current optimal or if the difference can be attributed to noise. Additionally, the computational expense of LES mandates optimizers to utilize information from all previous evaluations to decide the next design point to evaluate and not just the last few design points. Bayesian optimization is a robust global optimization algorithm that is suitable for optimizing such noisy and expensive objective functions.

In this class of optimization algorithms, a surrogate model is fit to all the design evaluations using a Gaussian process [86]. The surrogate model explicitly models the amount of noise in the objective and gradient evaluations. It forms a global approximation of the design objective utilizing information from all past evaluations. In Bayesian optimization, the next design point to evaluate is decided by optimizing a metric which is a scalar function on the design space. The metric generally depends on the surrogate model and past evaluations. The Bayesian optimization loop begins by fitting the surrogate model to the past evaluations, proceeds to optimize the metric to find the next design point and finally evaluates the design. The loop terminates when there is no appreciable improvement in the design objective in consecutive evaluation or the computational resources to perform the optimization are exhausted. As the evaluations are noisy, the optimal design at the end of the optimization process is the minimum of the surrogate model and not the design with the minimum objective value in all the evaluations.

Using a stochastic process to model the objective function helps in quantifying the uncertainty in a surrogate model that is trained on noisy evaluations. Additionally, it helps in designing a metric for deciding the next evaluation points that works towards finding the optimal design. A Gaussian process (GP) is a stochastic process whose mean ($\mu(\boldsymbol{x})$) and covariance functions ($k(\boldsymbol{x}, \boldsymbol{x}^*)$) are sufficient to completely define it, where $\boldsymbol{x}$ is the point in the design space ω. A realization of a GP, evaluated on a set of discrete points $\boldsymbol{X}$, is a sample of a multivariate normal distribution with mean $\mu(\boldsymbol{X})$ and covariance $k(\boldsymbol{X}, \boldsymbol{X})$. If the design objective function is denoted by $f(x)$, then the corresponding GP is defined by [86]

$$\mu(\boldsymbol{x}) = \mathrm{E}[f(\boldsymbol{x})], \tag{5.16}$$

$$k(\boldsymbol{x}, \boldsymbol{x}^*) = \mathrm{E}[(f(\boldsymbol{x}) - \mu(\boldsymbol{x}))(f(\boldsymbol{x}^*) - \mu(\boldsymbol{x}^*))], \tag{5.17}$$

where the true functions $\mu(\boldsymbol{x})$ and $k(\boldsymbol{x}, \boldsymbol{x}^*)$ for the design objective are unknown. An initial estimate for the mean and covariance functions is known as the prior GP. On observing a few evaluations, the prior GP is updated using the Bayes rule to form the posterior GP. A prior mean function that is commonly used is $\mu(\boldsymbol{x}) = \mu_f$, where μ_f is a constant. The choice of the prior covariance function restricts the function space associated with the GP. As a majority of the design objectives typically observed in CFD are smooth (continuous and infinitely differentiable), the squared exponential kernel ($k(\boldsymbol{x}, \boldsymbol{x}^*) = \sigma_k^2 e^{-(\frac{|\boldsymbol{x}-\boldsymbol{x}^*|}{c_l})^2}$) is used as the covariance function. This kernel has a few hyperparameters that need to be decided before beginning the optimization process. The hyperparameter, c_l, is the length scale of the kernel that determines how fast the function changes over the design space. The hyperparameter, σ_k^2, is the signal variance and it controls how much nearby points in the design space are correlated.

Assuming additive noise in the design evaluations, the equation for an objective function and gradient evaluation is written as

$$\boldsymbol{y} = \begin{pmatrix} f(\boldsymbol{x}) + \epsilon_1(\boldsymbol{x}) \\ f'(\boldsymbol{x}) + \epsilon_2(\boldsymbol{x}) \end{pmatrix} \tag{5.18}$$

where ϵ_1, ϵ_2 are random variables representing the noise in the objective function and gradient evaluations respectively. $\epsilon_1(\boldsymbol{x})$ and $\epsilon_2(\boldsymbol{x})$ are assumed to be independent normal random variables, $\epsilon_1(\boldsymbol{x}) = N(0, \sigma_1^2(\boldsymbol{x}))$ and $\epsilon_2(\boldsymbol{x}) = N(0, \sigma_2^2(\boldsymbol{x}))$, where σ_1^2 and σ_2^2 denote the variance and are a function of the design space. Independence of ϵ_1 from ϵ_2 can be explained from the observation that the additional artificial viscosity in adjoint equations decorrelates the time series of the gradient evaluations from the time series of the objective evaluations. Due to a lack of knowledge in the correlation structure of the gradient components, the noise terms in each of the components of the gradient evaluation are assumed to be independent. The noise in each design point evaluation is assumed to be independent of other design points as the correlation between the error due to finite time-averaging of the design objectives for different points in design space is observed to be zero. This is due to the fact that the chaotic time series, formed by the instantaneous design objective evaluations, exponentially diverges from the time series of even neighboring design points. The variance of the noise term is modeled heteroskedastically, which means that the amount of noise in the evaluations is a function of the design point. The dependence of σ_1^2 and σ_2^2 on $\boldsymbol{x}$ is not known beforehand. Hence, they are modeled as independent logarithmic (log) GPs. This ensures that the variance term remains positive [53]. $\sigma_1(\boldsymbol{x})$ is represented as

$$log(\frac{\sigma_1^2)}{\mu_{\sigma_1^2}}) \sim GP(0, k_1(\boldsymbol{x}, \boldsymbol{x}^*)) \tag{5.19}$$

where $k_1(\boldsymbol{x}, \boldsymbol{x}^*) = e^{-\frac{|\boldsymbol{x}-\boldsymbol{x}^*|^2}{c_l^2}}$, $\mu_{\sigma_1^2}$ is the prior mean for the log GP. A similar expression can be used to represent $\sigma_2(\boldsymbol{x})$.

Using the aforementioned noise model, evaluations of the objective function and gradient for the design points are used to update the GP using Bayes rule to form a posterior GP. Consider a set of sample points $\boldsymbol{X}_s$ and the corresponding function and gradient evaluations $\boldsymbol{y}_s$. The mean ($\boldsymbol{\mu}_p$) and covariance ($\boldsymbol{\Sigma}_p$) of the posterior process, when evaluated on a set of evaluation points $\boldsymbol{X}$ is given by

$$\begin{aligned}\boldsymbol{\mu}_p &= \boldsymbol{K}^T(\boldsymbol{K}_s + \boldsymbol{\Sigma}_s)^{-1}\boldsymbol{y}_s, \\ \boldsymbol{\Sigma}_p &= k(\boldsymbol{X}, \boldsymbol{X}) - \boldsymbol{K}_*^T(\boldsymbol{K}_s + \boldsymbol{\Sigma}_s)^{-1}\boldsymbol{K},\end{aligned} \tag{5.20}$$

where k' denotes the derivative of k with respect to the first argument, k'' denotes the Hessian of the derivatives with respect to the first and second arguments and

$$\begin{aligned}\boldsymbol{K}_s &= \begin{pmatrix} k(\boldsymbol{X}_s, \boldsymbol{X}_s) & k'(\boldsymbol{X}_s, \boldsymbol{X}_s)^T \\ k'(\boldsymbol{X}_s, \boldsymbol{X}_s) & k''(\boldsymbol{X}_s, \boldsymbol{X}_s)) \end{pmatrix}, \boldsymbol{K} = \begin{pmatrix} k(\boldsymbol{X}, \boldsymbol{X}_s) \\ k'(\boldsymbol{X}, \boldsymbol{X}_s) \end{pmatrix}, \\ \boldsymbol{\Sigma}_s &= \begin{pmatrix} diag[\sigma_1^2(\boldsymbol{X}_s)] & 0 \\ 0 & diag[\sigma_2^2(\boldsymbol{X}_s)] \end{pmatrix},\end{aligned} \tag{5.21}$$

The variance estimate of the sample mean of the design objective and gradient evaluations, discussed in Section 5.1.2, is used to update the noise GPs $\sigma_1(x)$ and $\sigma_2(x)$ using an expression similar to 5.20 without the gradient evaluations.

Before starting the Bayesian optimization process, it is important to have a set of evaluations from which an initial surrogate model can be created [85]. This step, known as design of experiment (DoE), is also used to estimate the hyperparameters of the GP. The hyperparameters are estimated using maximum likelihood estimation [86]. The mean quantities, μ_f, $\mu_{\sigma_1^2}$ and $\mu_{\sigma_2^2}$, are estimated using a sample mean of the design and gradient evaluations and their variance estimates. The design points for the DoE are chosen from a random subset of points at the corners of the 4-dimensional hypercube that contains the design space, midpoints of the edges of the hypercube and the centroid of the hypercube. The points which do not satisfy the constraints of the design optimization problem are omitted.

In the Bayesian optimization loop, after fitting the surrogate model to all the past objective function and gradient evaluations, a metric function is optimized to find the next design point to evaluate. The metric can be designed to achieve a certain set of optimization goals [35, 19, 100, 82, 45]. For example, a popular optimization strategy involves exploring the design space in the initial part of the optimization process in order to improve the quality of the surrogate and then exploiting the surrogate model by evaluating designs close to the minimum of the surrogate in order to find the optimal design. A metric that reflects this strategy is the expected improvement (EI) criterion [50]. It provides a good balance between exploration and exploitation. EI is defined by the following expression

$$\mathrm{EI}(\boldsymbol{x}) = \mathrm{E}[\max(f_{min} - f(\boldsymbol{x}), 0)], \tag{5.22}$$

where f_{min} is the current estimated minimum of the objective, $f_{min} = \min_{\boldsymbol{x}\in\omega}\mu(\boldsymbol{x})$. For GPs, EI has a compact analytical form obtained by integrating over the expectation [97]

$$\mathrm{EI}(\boldsymbol{x}) = (f_{min} - \mu(\boldsymbol{x}))\Phi\left(\frac{f_{min} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) + \sigma(\boldsymbol{x})\phi\left(\frac{f_{min} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right), \tag{5.23}$$

where ϕ is the standard normal density, Φ is the standard normal distribution function and $\mu(\boldsymbol{x}$ and $\sigma(\boldsymbol{x})$ are the mean and standard deviation of the GP. The point in the design space ($\boldsymbol{x}_n$), which maximizes EI, $\boldsymbol{x_n} = \mathrm{argmax}_{\boldsymbol{x}\in\omega}\mathrm{EI}(\boldsymbol{x})$, is chosen as the next design to evaluate.

If the evaluations of the objective function are not noisy, EI can be shown to converge to the global minimum [110] provided the objective function belongs to the class of functions that can be represented by the surrogate model. But, when the evaluations are noisy, there is no proof of convergence. Additionally, it has been observed that using the EI metric can cause the optimization process to get stuck in a local minimum[81]. The reason is that the optimizer spends too many evaluations exploiting the GP without doing enough exploration. One possible solution to this problem is to choose f_{min} in such a way that EI is biased towards exploration.

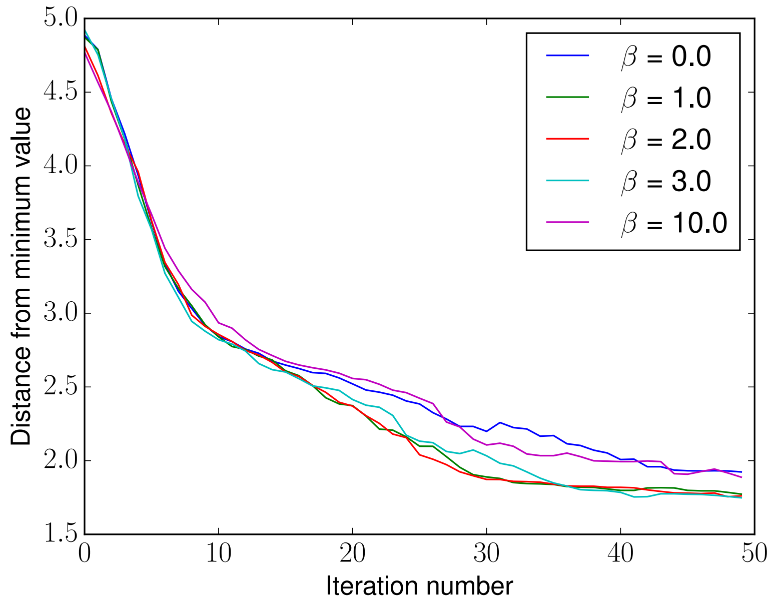
$$f_{min} = \min_{\boldsymbol{x}\in\omega}(\mu(\boldsymbol{x}) + \beta\sigma(\boldsymbol{x})) \tag{5.24}$$

Increasing β leads to a significant increase in f_{min}, and consequently EI, for regions of the design space where $\sigma(\boldsymbol{x})$ is high. Hence, due to the larger EI values in unexplored regions, higher β results in more exploration.

Numerical experiments show that for various values of β the Bayesian optimization algorithm, with the modified EI metric, converges to the global optimum for a range of noisy functions including the noisy Rastrigin and long-time averaged quantities of chaotic systems like the Lorenz system. Figure 5-4 shows the performance of the exploration biased EI metric on 2-dimensional optimization for two parameter choices of ζ for the noisy Rastrigin function. The function is given by

$$J(\boldsymbol{x}) = -30 + x_1^2 + x_2^2 - 10[cos(2\pi\zeta x_1) + cos(2\pi\zeta x_2)] + \psi z \tag{5.25}$$

where z is the standard normal random variable. The global minimum of this function is at $\boldsymbol{x} = (0, 0)$ and the minimum objective value is -50. ψ is set to 4, which means that the objective function has a large amount of noise. When ζ is set to 0.5, the objective function has a lower number of local minima than when $\zeta = 1.5$. Figure 5-4 shows the trajectory of the distance of the minimum evaluation design point from the global minimum design point, averaged over 1000 runs of the optimizer. For the objective function with $\zeta = 0.5$, using any positive value of β results in the Bayesian optimization algorithm finding an optimal design that is closer to the global minimum than using $\beta = 0$. Using higher values of β enables the optimizer to explore more and reach a lower objective value. For the objective function with $\zeta = 1.5$, using $\beta = 1, 2$ or 3 results in better performance for the Bayesian optimization algorithm. The high number of local minima lowers the convergence rate of the various optimization algorithms with different values of β and increases the difficulty of finding the global optimal design. Similar to the $\zeta = 0.5$ case, positive values of β causes the optimizer to suppress exploitation of local minima and do more exploration. For all the optimization runs, the number of DoE points are set to 4.



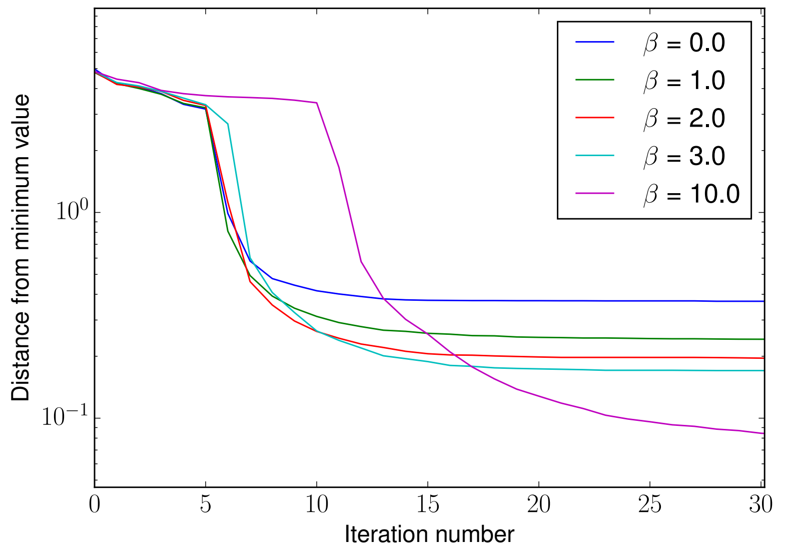


Figure 5-4: Comparison of the distance from the global minimum of the minimum design evaluated by the Bayesian optimization algorithm using different β values for the modified EI metric on the modified noisy Rastrigin function. Top figure: Rastrigin function with $\zeta = 1.5$. Bottom figure: Rastrigin function with $\zeta = 0.5$.

5.2.3 Algorithm

The final optimization procedure for the trailing edge shape optimization problem, where the number of parameters in the optimization problem is $n = 4$, is described below

1. Evaluate $2n$ design points, where the design points are obtained from a design of experiment.

2. Decide hyperparameters for GP using maximum likelihood estimation applied to the $2n$ design evaluations.

3. Obtain posterior GP by fitting the surrogate model to all the past design evaluations.

4. Find next design point to evaluate by maximizing the exploration biased EI metric using $\beta = 1$.

5. Evaluate design point.

6. If the computational resources are exhausted or the optimization process has reached convergence, then return the optimal design, else repeat the process starting from step 3.

The cost of this algorithm is $(10n + 5m)C_p$, where m is the number of design evaluations in the optimization loop and C_p is the cost of obtaining the design objective value (or a single flow solution). The cost of a design evaluation is $5C_p$, as the typical cost of obtaining the design objective gradient (or a single adjoint solution) is $4C_p$.

5.3 Results

The shape of the trailing edge of the turbine vane is optimized using the modified Bayesian optimization algorithm utilizing the viscosity stabilized adjoint method for computing gradients. The optimization process begins with a design of experiment for 8 evaluations. The hyperparameters for the Gaussian process are $c_l = (0.7, 0.4, 0.5, 0.3)$ and $\sigma_k^2 = 10^{-6}$. The prior means for the GPs are $\mu_f = 0.0092, \mu_{\sigma_1^2} = 5 \times 10^{-9}$ and $\mu_{\sigma_2^2} = (10^{-9}, 10^{-7}, 10^{-8}, 10^{-7})$. The Bayesian optimizer is run for a total of 16 design objective and gradient evaluations. The value of β for the exploration biased EI metric is set to 1. During the optimization, multiple trailing edge shapes are found which have a lower design objective value. The optimization ends when consecutive designs evaluated by the optimizer do not lead to a reduction in the design objective value or its standard deviation. Figure 5-5 shows how the optimizer spends a majority of the initial evaluations on exploration (high standard deviation and high mean objective value of the posterior GP at evaluation point) and the evaluations towards the end of the optimization process in exploitation (low standard deviation and low to high mean objective value of the posterior GP at evaluation point).

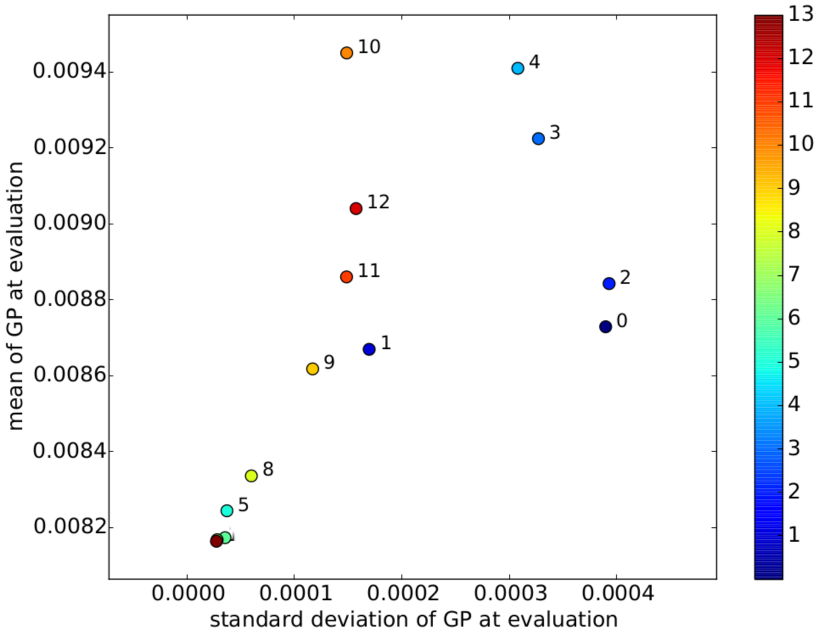


Figure 5-5: Plot of the design objective mean value and standard deviation of the posterior GP evaluated at the design point decided by the modified EI metric at each step of the optimization process. The points in the plot are colored and labeled by the optimization step number.

The optimization used a mixture of computational resources consisting of CPUs and GPUs. The design objective value was obtained by computing the numerical flow solution on an Nvidia GeForce GTX 1080Ti, which has 3584 CUDA cores and 11 GB RAM. The design objective gradient was obtained by computing the numerical adjoint solution on 16 Intel Xeon E5-1650 CPUs, where each of the CPUs has 4 cores and 32 GB RAM. The CPUs are interconnected using Gigabit Ethernet. The time to solution for a single design objective value is 12 hours and for a single design objective gradient is 12 hours. Hence, the total computational cost of the optimization is 18,432 CPU core hours and 288 GPU hours.

5.3.1 Comparison of optimal and baseline designs

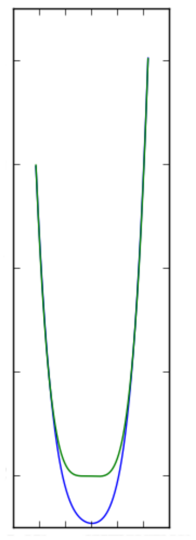


Figure 5-6: Visualization of the current optimal design (blue color) and baseline design (green color)

The optimal design at the end of the optimization process is shown in Figure 5-6. The baseline design has a design objective value equal to 0.00931 ± 0.0004, whereas the optimal design has the objective value 0.008124 ± 0.00003. The optimal design has an approximately 12% reduction in Nusselt number and 16% reduction in pressure loss coefficient. The design of experiment procedure produced a design with an objective value 0.00831. Consequently, the optimization procedure led to a 2.2% improvement in the design objective over the design of experiment.

A visualization of the magnitude of the instantaneous velocity field (U) and the gradient of the temperature field (gradT) are shown in Figure 5-7 and 5-8 respectively. A comparison of the two fields for the baseline and optimal design shows that the optimal design has a slightly thinner turbulent boundary layer on the suction side

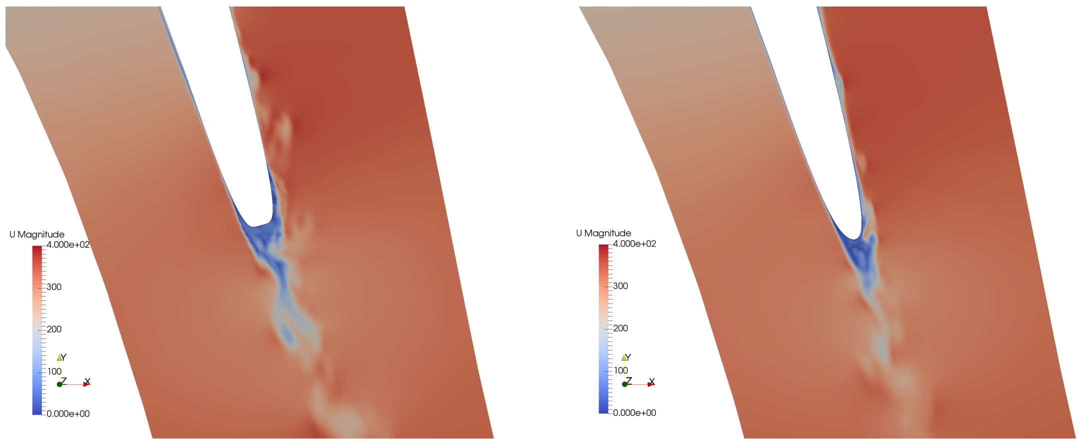


Figure 5-7: Left figure: Visualization of velocity field for baseline design. Right figure: Visualization of velocity field for optimal design

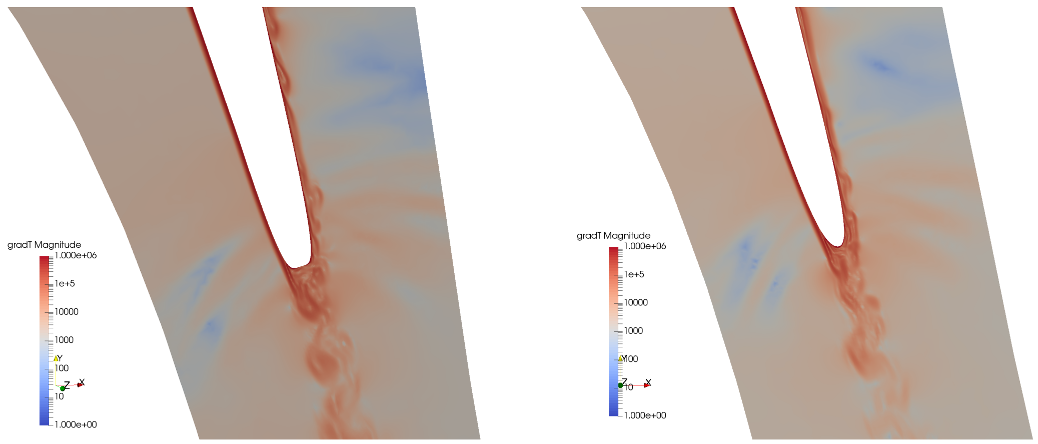
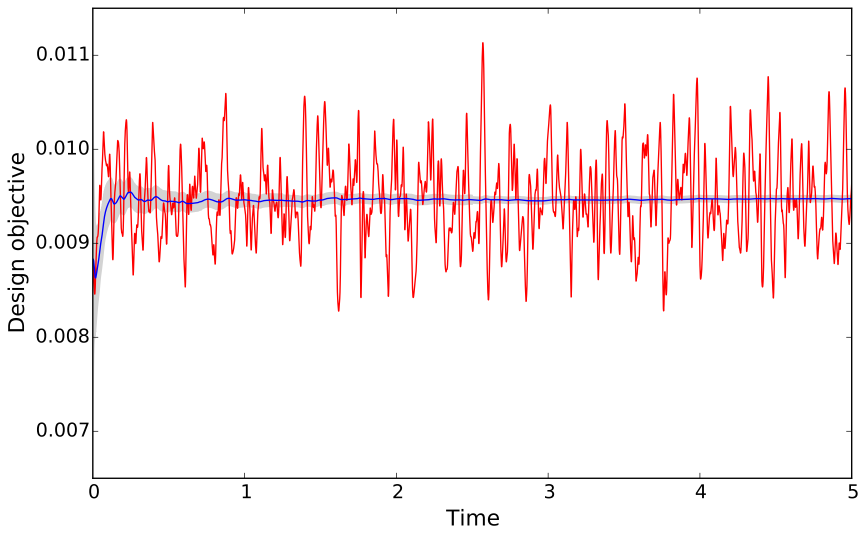


Figure 5-8: Left figure: Visualization of gradient of temperature field for baseline design. Right figure: Visualization of gradient of temperature field for current optimal design

near the trailing edge. The width of the wake and the size of the vortex structures in the wake are smaller in the optimal design. Both these factors contribute to the lower stagnation pressure loss in the optimal design. A thicker turbulent boundary layer leads to a higher convective heat transfer due to more mixing in the flow. Hence, the optimal design has a lower heat transfer than the baseline design. Finally, visualizations of the instantaneous design objective for the baseline and optimal designs are shown in Figure 5-9.



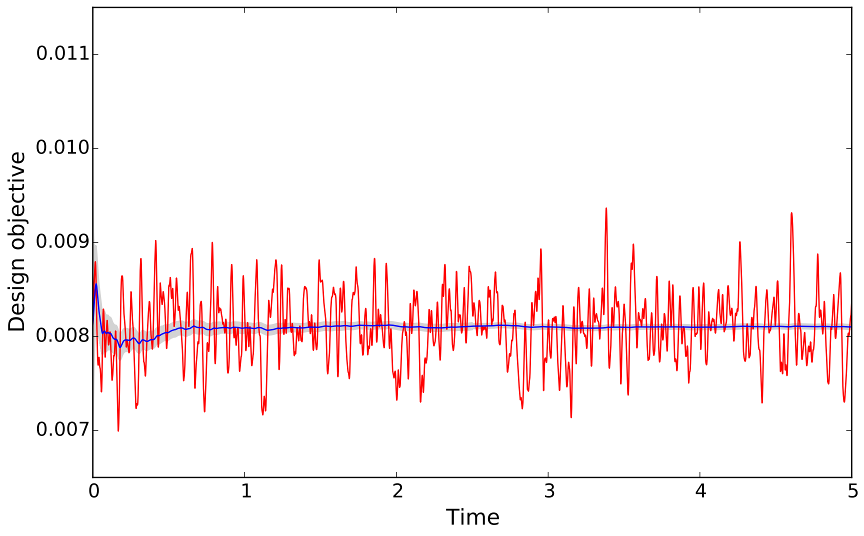


Figure 5-9: Top figure: Visualization of design objective for baseline design. Bottom figure: Visualization of design objective for current optimal design

5.3.2 Comparison to Bayesian optimization without gradients

In order to quantify the utility of the gradient in the optimization process a comparison is performed between Bayesian optimization with and without gradients. Running a separate optimization without gradients is too expensive as this would involve running a large number of large eddy simulations. Hence, the two Bayesian optimization algorithms are compared by averaging multiple optimization runs that

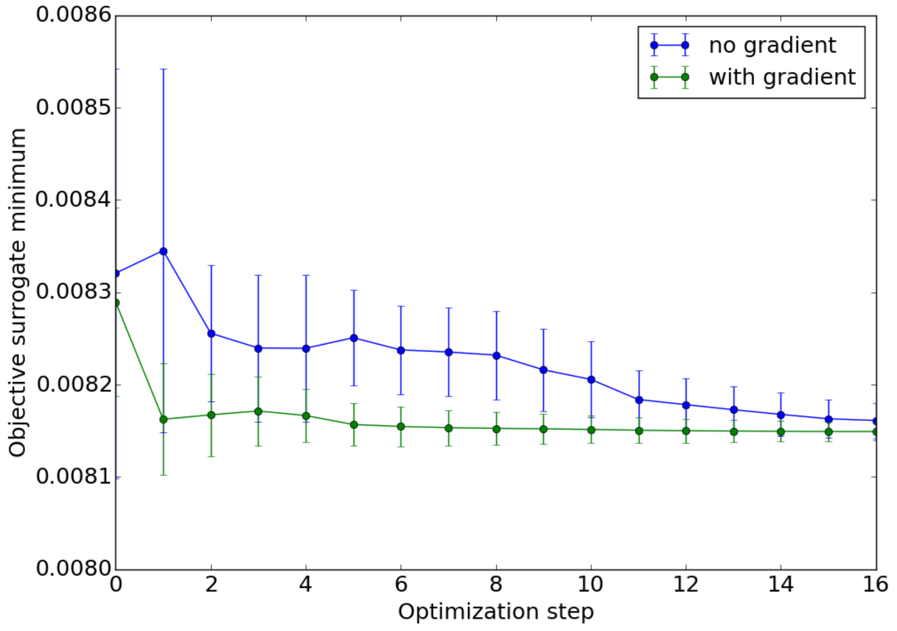


Figure 5-10: Comparison between Bayesian optimization with and without gradients

use an approximate LES design objective function. This objective is a Gaussian process which is trained on all the objective and gradient evaluations obtained during the adjoint-based design optimization process. Both the Bayesian optimization algorithms utilize the same DoE evaluations obtained during the adjoint-based design optimization. While running the optimizations, whenever the algorithm requires an evaluation for a particular design point, the objective is evaluated (including gradients if required) by sampling the Gaussian process at that point using Equation 5.18 by replacing $f(\boldsymbol{x})$ and $f'(\boldsymbol{x})$ with the mean of the corresponding Gaussian process. The noise term is realized by sampling the normal distribution with zero mean and variance computed from the mean of the noise log GP in accordance with Equation 5.19.

Figure 5-10 shows the trajectories of the minimum objective value for the two optimization algorithms as a function of the number of function evaluations. The minimum objective value as reported by the optimizers is minimum of the mean function of the posterior GP of the optimization algorithm. The figure shows a trajectory of the mean and the uncertainty corresponding to a single standard deviation of the posterior GP at the design point with the minimum objective value. The Bayesian optimizers with and without gradients are executed 1000 times to achieve a statistically converged Monte Carlo estimate of the optimization trajectory. The estimates provide less than 1% relative sample standard deviation for each mean value in the optimization trajectory.

From the figure it can be seen that the Bayesian optimization with gradients performs significantly better than optimizer without gradients as it reaches the optimal design faster. Furthermore, the optimizer with gradients obtains the minimum design objective value with a much lower standard deviation.

Chapter 6

Optimized code generation for unsteady adjoint methods

In this chapter a new automatic differentiation method is developed. The method represents the computations involved in the flow solver in a two-level graph structure to facilitate better performance. Section 6.1 describes the graph structure and Section 6.2 demonstrates an implementation the method in the form of a Python library. The library defines an interface for writing a structured or unstructured explicit unsteady flow solver using a finite volume or finite difference discretization scheme. Section 6.3 details the adjoint derivation procedure and implementation. Finally, Section 6.4 shows performance benchmarks of the developed flow solver.

6.1 Representing flow solver as a two-level computational graph

The computations of the flow solver are represented in the form of a two-level directed acyclic graph [30]. The overall structure of the graph is defined by an outer level graph which consists of two types of nodes: arrays and kernels. An array provides a discrete representation of a scalar, vector or tensor field in the flow solver. The first dimension of the array spans the elements of the mesh (either cells, faces, nodes or edges depending on the type of discretization). For scalar fields, like pressure and temperature, the second dimension is 1, for vector fields like velocity the second dimension is 3. For tensor fields like gradient of velocity, the second and third dimensions are 3. A kernel denotes a function that applies a stream of operations on the input arrays of the kernel and produces the corresponding output arrays. Directed edges in the outer graph either denote array feeding into a kernel as an input or a kernel producing an output array.

The operations in each kernel are represented by an inner level graph. This graph signifies a set of operations that can be performed independently for each element of the mesh. Similar to the outer graph, there are two types of nodes in the inner graph: variables and operations. The variables represent intermediate values of operations like arithmetic operations or value modification. They are only defined locally, in the scope of the kernel, while performing the operations in the kernel for each element of the mesh. Unlike in the case of arrays, there is no global allocation of memory for these variables. Similar to the outer graph, directed edges in the inner graph either denote variables feeding into an operation as an input or an operation producing a new variable.

The separation of the graph into two levels, an outer level graph and an inner level graph, allows for more efficient code generation and execution. Furthermore, the computational graph approach simplifies the derivation of the adjoint graph. Utilizing the graph, efficient memory managers can be designed to increase the performance and reduce the memory utilization of the flow solver.

6.1.1 Inner graph

A kernel node, denoting an inner graph, represents the operations applied on the input arrays to produce the output arrays. There are 3 kinds of mathematical operations that are typically encountered in the numerical methods used to discretize the partial differential equations (PDEs) for flow physics. They are:

1. Element-wise arithmetic or transcendental operations.

2. Accessing or modifying value of neighboring element.

3. Global dimension reduction operations.

The first type, element-wise operations, involve arithmetic or transcendental functions operating on field values of every element of the mesh (or in other words the values of an array). The second type of operation involves indirect memory accesses for accessing or modifying field values of the neighboring elements of the mesh. An indirect memory access is a memory access where the memory offset is a variable. The values of an array for neighboring elements may not be present in adjacent memory addresses with a constant memory offset, necessitating the use of an additional variable to locate the correct memory address. This is an expensive operation. The last type of operation, global reduction, involves eliminating the first dimension of a solution field by performing a sum or a min/max operation over the all the elements of the mesh.

In finite volume methods for discretizing flow physics equations, a common kernel node that is encountered is the interpolation of field values of cells onto faces. This kernel uses indirect memory accesses for obtaining the field values of a cell adjacent to a face. A visualization of the computational graph of this kernel is shown in Figure 6-1. The round blocks represent operations and rectangular blocks represent intermediate variables. The kernel is processed for each face of the mesh. The extract operation performs an indirect memory access to get the field value of the cell adjacent to a face. $Left$ represents the index of the cell on one side of the face and $right$ represents the index of the cell on the other side of the face.

Calculations of the flow solver should be divided into kernels in such a way as to maximize the set of operations that can be performed in each of the inner graphs. This results in the formation of *fat* kernels, which are kernels that contain a lot of floating point and load/store operations. They ensure that most intermediate results remain in the CPU or GPU cache. As many current flow solvers are memory bandwidth limited, better cache utilization by reducing memory transfers between main memory and cache can lead to significant performance benefits.

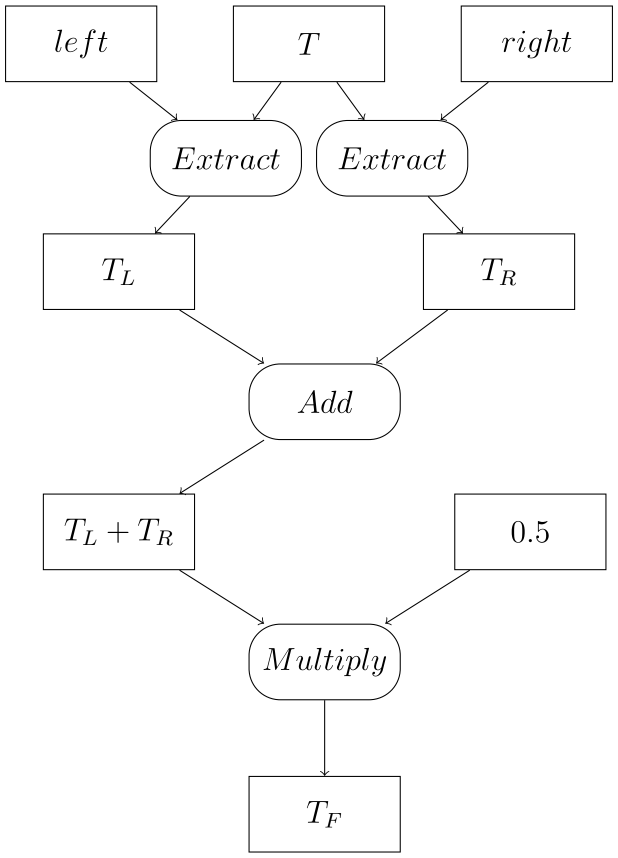


Figure 6-1: Computational graph of the cell-to-face interpolation kernel.

6.1.2 Outer graph

The outer graph describes the relative order and location of the inner graphs in the two-level computational graph. Additionally, it prescribes when arrays should be initialized and passed as inputs to the kernel nodes.

When discretizing the compressible fluid flow (Navier-Stokes) equations using FVM, the following ordinary differential equations (ODEs) are obtained [59]

$$\frac{d\phi^c}{dt} = \Psi^c = \sum_{f_i \in F} \hat{n}^F_{f_i} \cdot (\Psi^F_{f_i}) \Delta S^F_{f_i} \tag{6.1}$$

for each internal cell of the mesh. ϕ^c is the array that denotes the average field values of $\phi = (\rho, \rho\mathbf{u}, \rho E)$ (representing density, momentum and total energy) on the internal cells of the mesh. Ψ_c is the right hand side update term of the ODEs. $\hat{n}^F_{f_i}$ is the normal vector on a face f_i adjacent to a cell and $\Delta S^F_{f_i}$ is the face area. The superscript F means that the array stores field values on both internal faces and boundary faces of the mesh. The computation of the flux term $\Psi_{f_i}^{F}$ utilizes a large number of arithmetic operations performed on each face of the mesh. An outer graph representation for computing the fluxes and update terms for a compressible flow solver is shown in Figure 6-2. In the figure, round nodes represent the kernels and the rectangular nodes represent the arrays. The superscript C means that the array contains field values from both internal and boundary cells of the mesh. The primitive kernel transforms the conservative fields $(\rho, \rho\mathbf{u}, \rho E)$ into primitive fields (U, T, p). The boundary kernels computes the field values on the boundary cells from the field values on the internal cells using predefined boundary conditions of the flow solver. The gradient kernel computes the gradient of the primitive fields using the Green-Gauss theorem. Finally, the flux kernel computes the fluxes on the internal cells for each of the compressible flow equations using the Roe approximate Riemann solver [89]. The ODEs in the flow solver are solved using an explicit strong stability preserving third-order Runge-Kutta (RK3) method [63]. An outer graph representation of the RK3 method is shown in Figure 6-3. In the figure, ϕ_0^c represents the field values of ϕ on the internal cells from the previous time step. Subscripts in this graph denote the stage of the RK3 method. The circles form a compact representation of the outer-level graph for computing the fluxes. The array at the end of this graph, ϕ_3^c, represents the field values at the new time step.

6.2 Implementation

The two-level computational graph method is implemented in the form of a Python library called adFVM. The library defines an application programming interface (API) for constructing the flow solver. Once constructed, it analyzes the graph and generates the corresponding C/C++ code for executing the graph. The library is able to generate code for multiple architectures including Intel/AMD CPUs, Nvidia GPUs (using CUDA) and Intel MICs (using OpenMP).

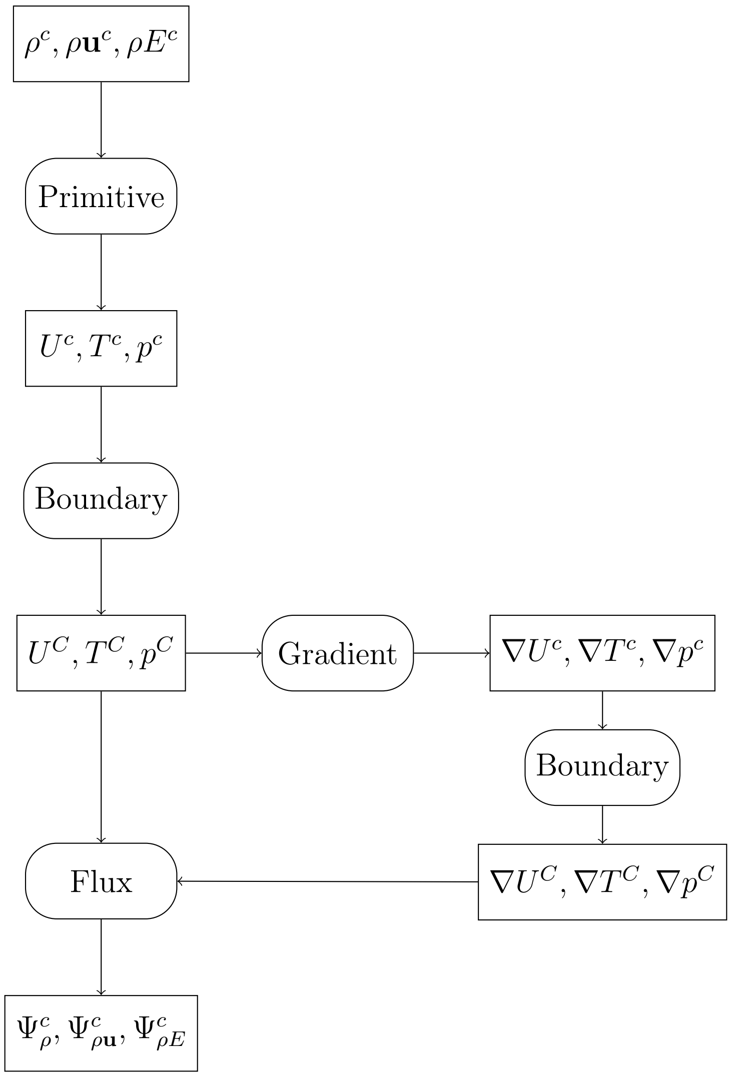

$\rho^c, \rho\mathbf{u}^c, \rho E^c$
Primitive
U^c, T^c, p^c
Boundary
U^C, T^C, p^C
Gradient
$\nabla U^c, \nabla T^c, \nabla p^c$
Boundary
$\nabla U^C, \nabla T^C, \nabla p^C$
Flux
$\Psi^c_\rho, \Psi^c_{\rho\mathbf{u}}, \Psi^c_{\rho E}$

Figure 6-2: Computational graph for computing the fluxes of the compressible flow equations.

6.2.1 Kernel implementation

The first step in the code generation process involves transforming the kernels (inner graphs) into functions in C. These kernel functions accept as arguments the relevant input and output arrays. They execute a single *for* loop (on a CPU), access each element of the input arrays, perform the necessary operations and then store the results into the output arrays. The inner graph is transformed into a series of floating-point arithmetic operations or memory loads/stores by traversing the graph in a topological ordering. A topological ordering of a directed graph is an ordering of its

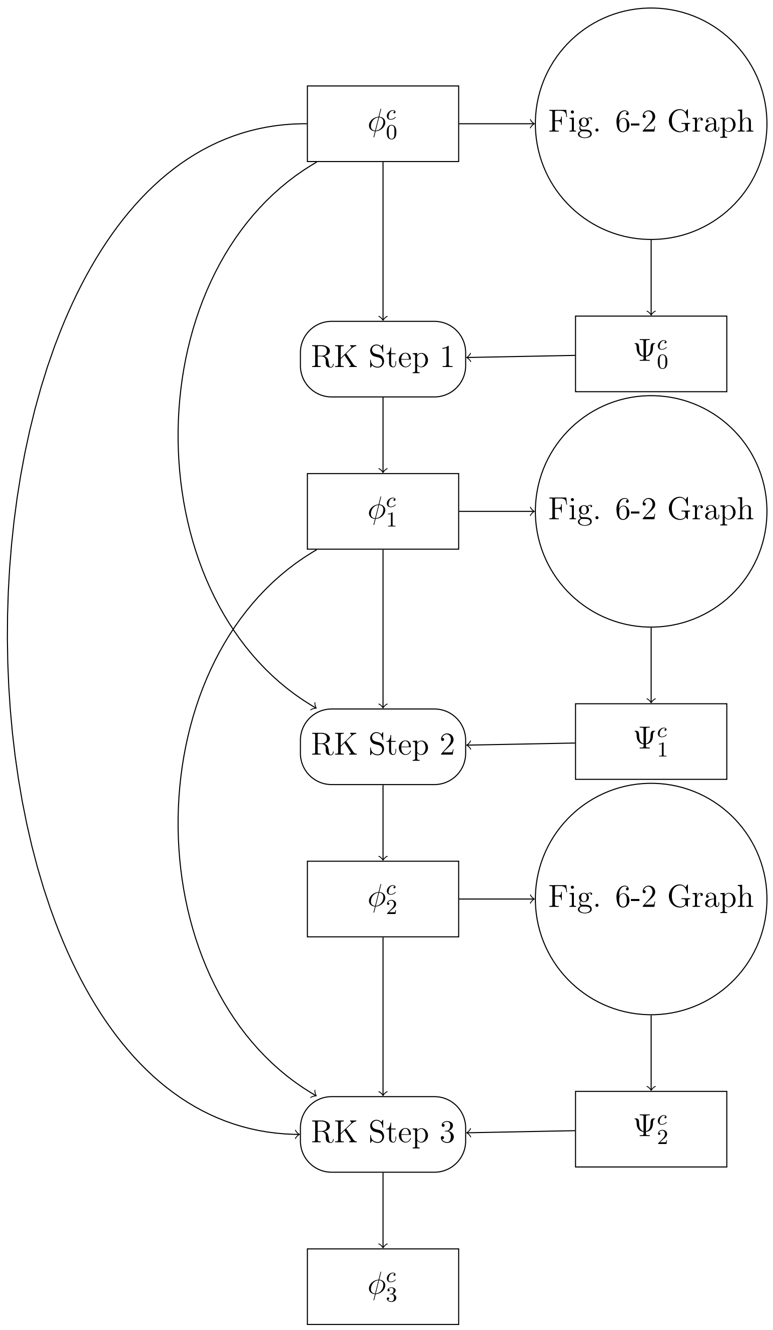


Figure 6-3: Computational graph of the third-order Runge-Kutta (RK3) time integrator method.

nodes such that for every edge in the graph from a parent node to its child node, the parent node comes before the child node in the ordering. In the library, each operation in the inner graph is a Python object with a "c_code" method for generating the C code that performs the operation. A demonstration of the Python API for

```python
def interpolate(T, left, right):
    # These statements extract value of the field on cells
    # adjacent to a particular face.
    TL, TR = T.extract(left), T.extract(right)
    # This statement computes value of the field on a face as the
    # average of its value on adjacent cells.
    TF = 0.5*(TL+TR)
    return TF
```

```c
// The first argument of the function defines the number
// of elements on which the kernel is applied.
// The const pointer arguments define the input arrays.
// The non-const pointer arguments denote the output arrays.
// The __restrict__ keyword specifies that there is no pointer
// aliasing in the input and output array pointers.
void Function_interpolate(int n,
                          const scalar* __restrict__ Tensor_0,
                          const integer* __restrict__ Tensor_1,
                          const integer* __restrict__ Tensor_2,
                          scalar* __restrict__ Tensor_3) {
    integer i;
    // The for loop over all the elements
    // for which the kernel is applied to.
    for (i = 0; i < n; i++) {
        scalar Intermediate_0 = 0.5;

        // These statements extract value of fields
        // for a particular element.
        // code: TL = T.extract(left)
        integer Intermediate_1 = Tensor_2[i*1 + 0];
        scalar Intermediate_3 = Tensor_0[Intermediate_1*1 + 0];
        // code: TL = T.extract(right)
        integer Intermediate_4 = Tensor_1[i*1 + 0];
        scalar Intermediate_5 = Tensor_0[Intermediate_4*1 + 0];

        // These statements perform intermediate operations
        // code: TI = TL + TR
        scalar Intermediate_6 = Intermediate_5 + Intermediate_3;
        // code: TF = 0.5*TI
        scalar Intermediate_7 = Intermediate_6 * Intermediate_0;

        // This statement stores field in output array by adding
        // to an existing value.
        Tensor_3[i*1 + 0] += Intermediate_7;
    }
}
```

Figure 6-4: Python and C code of a simple tensor expression for interpolating a cell-centered field onto faces

writing a kernel that interpolates a cell-centered field on faces and the corresponding generated C code is shown in Figure 6-4. The operations in this kernel form a small subset of the flux kernel used in Figure 6-2. The full flux kernel is not shown due to space constraints. The first argument of a kernel function denotes the number of elements for which the operations of the inner graph are performed and the remaining arguments are the memory addresses for the input and output arrays.

As mentioned in Section 6.1.1, forming *fat* kernels can improve the performance of a flow solver. Other computational graph based code generators, like Theano [7] and Tensorflow [1], try to form such kernels by performing multiple traversals of the graph to combine kernel nodes. As these libraries might not have all the information necessary to know whether multiple kernels can be fused together, they can miss opportunities for optimizations. For example, indirect memory loads or stores are not combined with other arithmetic operations into a single kernel. Additionally, indirect stores on the same array are stored in separate memory locations and then combined, instead of using the same memory buffer for all indirect stores. Failure to optimize these operations can lead to additional memory allocations and transfers between main memory and cache. This degrades performance due a reduction in the ratio of operations to memory transfers and is especially disadvantageous on GPUs. Another advantage of combining all the operations into a single kernel is that the larger number of instructions enable the compiler and the processor to further optimize execution by using sophisticated instruction scheduling algorithms.

Automatic identification and construction of these *fat* kernels is a difficult task. The programmer of a flow solver generally has a good understanding of the pieces of code that can be considered to operate independently on each element of the mesh. Hence, instead of writing a graph analyzer that makes decisions on whether to combine kernels, the responsibility of dividing the code into distinct kernels is left to the programmer. Additionally, in a flow solver, many kernels are applied repeatedly with different input and output arrays. The Python library provides an interface to define such kernels, generating reusable and efficient code.

6.2.2 Outer graph implementation

The outer graph dictates the order of execution of the various kernels. Once the code for the kernels is generated, the outer graph is translated into a C++ function that can be called from Python. The kernel nodes are converted into kernel function calls with the corresponding input and output arrays and number of elements to process the kernel for, as arguments. Figure 6-5 demonstrates the Python code needed to define the outer graph for the RK3 time integrator previously shown in Figure 6-3.

```python
# Arguments of the function are the field at the
# current time step and the required mesh fields.
def RKGraph(phi0, *meshArgs):
    # This function transforms
    # the list of mesh fields into an object with
    # the fields as attributes.
    mesh = createMeshObject(meshArgs)
    # This function computes the flux from the fields
    # at the current time step.
    psi0 = fluxGraph(phi0, *meshArgs)
    # The RKStep1 function is a method that performs
    # the first RK step given the field and the flux.
    # The Kernel function creates a kernel node for RKStep1.
    # The kernel node is called on the internal cells of the
    # mesh with the field and the flux as arguments.
    # This kernel is the first RK step on all the
    # internal cells of the mesh.
    phi1 = Kernel(RKStep1)(mesh.nInternalCells)(phi0, psi0)
    # This function performs the second step of the RK integrator.
    psi1 = fluxGraph(phi1, *meshArgs)
    phi2 = Kernel(RKStep2)(mesh.nInternalCells)(phi0, phi1, psi1)
    # This function performs the third step of the RK integrator.
    psi2 = fluxGraph(phi2, *meshArgs)
    phi3 = Kernel(RKStep3)(mesh.nInternalCells)(phi0, phi1,\
                                                 phi2, psi2)
    return phi3
```

Figure 6-5: Python code for defining the computational graph for a RK time integration scheme

Many kernels in the flow solver, like the gradient and flux kernels, add values to the output arrays and do not overwrite existing values. Hence, to support such operations the library allows passing previously generated arrays as output arrays to the kernels. Additionally, in a kernel, the operation that stores intermediate variables in output arrays always adds to the existing value in the array. Consequently, if no

```python
# Arguments for the function are the fields that need
# to be interpolated and the mesh fields required for it.
def interpolateGraph(T, *meshArgs):
    mesh = createMeshObject(meshArgs)
    # This kernel defines the cell to face interpolation function
    # as shown in Figure 6-1.
    interpolateKernel = Kernel(interpolate)
    # This function creates the output array
    # and initializes it to zero.
    TF = Zeros((mesh.nFaces, 1))
    # This function applies the interpolate kernel to the
    # internal faces of the field.
    TF = interpolateKernel(mesh.nInternalFaces, (TF,))(T,\
                                        mesh.left, mesh.right)
    # This statement extracts information
    # about a particular set of boundary faces.
    startFace = mesh.boundary['inlet'].startFace
    nBoundaryFaces = mesh.boundary['inlet'].nFaces
    # The square bracket is a reference operation
    # used to create a subarray. The kernel is called
    # by reusing the output array from the previous call.
    # This function applies the interpolate kernel
    # to the boundary faces.
    TF = interpolateKernel(nBoundaryFaces, (TF[startFace],))(T, \
                     mesh.left[startFace], mesh.right[startFace])
    return TF
```

Figure 6-6: Python code for a computational graph interpolating a cell-centered field onto faces

output array is explicitly provided to a kernel node, memory for a new output array is allocated and all entries are set to zero before passing it to the kernel function. In the outer graph, a kernel node creates a new array node to represent the incremented output array. Subsequent use of the output array should utilize the array created by the kernel.

In a flow solver, often, there are multiple kernels that operate on separate parts of a single array. Instead of storing the inputs and outputs of these kernels in separate arrays, the library allows passing part of an array as an input or output to the kernel. This reduces memory allocations and copying due to splitting or merging parts of an array. A part of the array (henceforth called a subarray) contains values that belong to a particular set of elements of a mesh. Examples of such sets are the set of internal cells of the mesh, the set of boundary faces or the set of faces belonging to specific regions of the boundary. The values of a subarray are constrained to be stored in contiguous memory locations of the underlying or parent array. A subarray can be created from an array by a reference operation that specifies an offset for the first dimension of the parent array. The subarray and the parent array share a unique identifier that signifies that the two arrays are related.

An example of an outer graph that uses some of the more complex features of the library is shown in Figure 6-6. The *interpolateKernel* kernel function is used twice on different parts of the input arrays *T*, *mesh.left* and *mesh.right* and output array *TF*. The reference operation, *TF[startFace]*, is used to create the subarray that is passed as an output array the second time *interpolateKernel* is called.

6.2.3 Memory management

The computational graph representation allows for global memory management strategies for either reducing memory usage or total number of memory allocations. On many hardware accelerators, like Nvidia GPUs, memory allocations and transfers are synchronous operations that cannot be executed concurrently with each other. They can take up a significant fraction of the execution time of the two-level computational graph. For example, using the simple strategy of allocating new memory whenever it's required and deallocating when it's no longer used, the amount of time spent in memory related tasks is 50% of the total execution time for the RK3 time integrator graph (henceforth called the flow solver graph). This means that by minimizing the number of memory allocations, potentially, the execution time can be reduced by up to 50%.

When executing a graph constructed using the Python library, memory reuse is prioritized by forming a memory pool. The pool is a data structure that stores the available memory addresses and their corresponding sizes. If a kernel requires memory for storing results in an output array, memory is acquired from the memory pool. If the pool does not contain a free memory address with the appropriate size, new memory is allocated. After the execution of a kernel, the memory of input arrays of the kernel that are no longer used in the rest of the computational graph are released into the memory pool. When running a fluid simulation, the flow solver graph is called multiple times with different inputs. At the start of a simulation the memory pool is empty. The memory pool reaches its maximum size at the end of the first execution of the flow solver graph, when all the necessary memory allocations have been performed. On later executions of the graph, there is no new allocation of memory. It is always acquired from the pool. This memory management strategy ensures maximum reuse of allocated memory, albeit at the cost of greater memory usage.

On every execution of the flow solver graph there are only some input arrays that change their values while the rest remain the same. The input arrays that do not change are called *static* arrays. For example, arrays describing the mesh fields like face normals, face areas are static arrays. When running the graph on GPUs, the input arrays of the graph often need to be transferred from the CPU onto the GPU. For static arrays it is beneficial to allocate memory and transfer the array to the GPU only on the first execution of the graph. On subsequent executions of the graph, the static arrays reuse the GPU memory address stored during the first execution. This strategy reduces the execution time of the graph by reducing memory transfers from CPU to GPU.

6.2.4 Parallelization

The flow solver is parallelized on multiple computer nodes by dividing the domain into separate parts such that each computer node gets an approximately equal amount of computation to perform. The mesh is decomposed into multiple subdomains by reducing a function of the total communication among processors [80]. When using flux computation procedure outlined in Figure 6-2, on every execution of the flow solver graph, the “Boundary" kernel uses non-blocking communication primitives from the message passing library (MPI), like *Isend* and *Irecv*, to transfer the field and gradient field values to neighboring processors. The Python library has special kernels written in C++ that can be used as kernel nodes to perform these operations.

The flow solver is constructed in such a way that the message passing communication overlaps with other computation. For example, while the relevant boundary field values of U^c, T^c, p^c are being transferred to neighboring processors, gradient computation of the primitive fields on internal cells occurs at the same time. The library enables this procedure by providing separate kernel nodes that start communication and wait for the messages to finish transferring.

6.3 Adjoint Graph

Similar to the computational graph of the flow solver, the adjoint flow solver is represented in the form of a two-level graph (henceforth known as the adjoint graph). The inner and outer graphs of the adjoint graph are derived with the help of reverse mode AD of the corresponding inner and outer graphs of the flow solver graph (hereby both referred to as the primal graph). In the first stage of the adjoint graph generation process, the kernel nodes or inner primal graphs (inner graphs of the primal graph) are differentiated to produce the corresponding adjoint kernels. Differentiation involves linearization of each operation in the inner graphs. In reverse mode AD, given the gradient of an objective function with respect to outputs of a kernel, the adjoint kernel computes the gradient with respect to the inputs of the kernel by propagating the gradient from outputs to inputs using the chain rule of differentiation while performing a reverse traversal of the inner graph.

A similar differentiation process is followed for the outer primal graph to get the outer adjoint graph. The objective gradient with respect to the output arrays of the outer primal graph are propagated to the inputs arrays of the outer primal graph using the adjoint kernels to get the gradient with respect to the input arrays.

6.3.1 Implementation

To aid the formation of the adjoint kernels, each operation in the inner primal graph has a “grad" method that generates the corresponding operations for performing the gradient propagation. The adjoint kernel is constructed by calling the “grad" method of the operations in a reverse topological ordering of the inner primal graph. Similar to the outer primal graph, the adjoint kernels are called with the appropriate input and output arrays as arguments in order to build the outer adjoint graph. Figure 6-7 shows how a primal and an adjoint kernel process the input and output arrays that are passed to them.

Reverse-mode AD is applied to the outer primal graph to construct the corresponding outer adjoint graph using the generated adjoint kernels. Similar to the outer primal graph, the memory for the input and output arrays of the adjoint kernels is managed explicitly using techniques described in Section 6.2.3. Special care needs to be taken in order to ensure that these arrays are allocated and passed to the adjoint kernels in a topological order of the outer adjoint graph. In contrast, the management of the local memory in the adjoint kernel is automatically handled by the compiler. Each adjoint kernel in the outer adjoint graph requires arrays for storing its outputs, which are the gradients with respect to input arrays of the corresponding primal kernel. In the outer primal graph, an array or its subarrays may be used as an input array in multiple kernels. Such arrays share a unique identifier. Correspondingly in the adjoint graph, the gradient with respect to these arrays (henceforth called a gradient array) share an additional unique identifier. For an adjoint kernel, if the gradient array with a particular unique identifier does not exist, the memory for the gradient array is allocated and it is initialized to zero. In subsequent adjoint kernels, the gradient arrays are matched with previously generated gradient arrays using their unique identifiers. Once matched, the gradient arrays are reused and passed as arguments to the kernel. The library implements this memory management procedure by maintaining a hash table of all gradient arrays with their identifiers serving as a key. As each primal and adjoint kernel always adds values to its output arrays and does not replace any values, using the chain rule of differentiation [38] it can be shown that a topological ordering traversal of the outer adjoint graph provides the correct gradient array after all the adjoint kernels are executed. This can be understood from the following example. Let $\boldsymbol{A_n}$ represent the set of values of all the arrays with unique identifiers in the outer primal graph at index n in a topologically ordering of the graph. The action of a particular kernel $\boldsymbol{K_n}$, with input arguments $\boldsymbol{A_n}$, can be mathematically represented in the following form

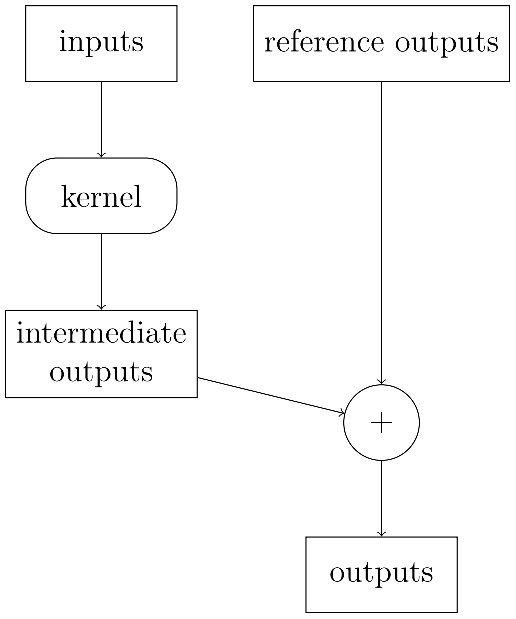
$$\boldsymbol{A_{n+1}} = \boldsymbol{A_n} + \boldsymbol{K_n}(\boldsymbol{A_n}) \tag{6.2}$$

where $\boldsymbol{A_{n+1}}$ represents the values of the distinct set of arrays in the outer primal graph after applying the kernel $\boldsymbol{K_n}$. For arrays of $\boldsymbol{A_n}$ that are not output arrays of $\boldsymbol{K_n}$, denoted by the subset $\boldsymbol{A^j_n}$, $\boldsymbol{K^j_n}(\boldsymbol{A_n}) = 0$. Hence, for these arrays, $\boldsymbol{A^j_{n+1}} = \boldsymbol{A^j_n}$. For arrays of $\boldsymbol{A_n}$ that are output arrays of $\boldsymbol{K_n}$, denoted by the subset $\boldsymbol{A^i_n}$, the resulting arrays from $\boldsymbol{K_n}$, $\boldsymbol{K^i_n}(\boldsymbol{A_n})$, are added to $\boldsymbol{A^i_n}$ in order to form the new set of arrays $\boldsymbol{A^i_{n+1}}$. In the outer adjoint graph, the action of the corresponding adjoint kernel $\boldsymbol{\hat{K}_n}$ on the set of gradient arrays with unique identifiers at index n in a topological ordering of the outer primal graph, $\boldsymbol{\hat{A}_n}$, can be written as

$$\boldsymbol{\hat{A}_{n-1}} = \boldsymbol{\hat{A}_n} + \boldsymbol{\hat{K}_n}(\boldsymbol{\hat{A}_n}, \boldsymbol{A_n}) \tag{6.3}$$

where $\boldsymbol{\hat{K}_n} = \frac{\partial \boldsymbol{K_n}(\boldsymbol{A_n})}{\partial \boldsymbol{A_n}}^T \boldsymbol{\hat{A}_n}$ is a linearization of the kernel $\boldsymbol{K_n}$ with respect to its input arguments multiplied by the output gradient arrays. The decrease in index of $\boldsymbol{\hat{A}}$ in the above equation represents the fact that the outer adjoint graph follows a reverse topological ordering of the outer primal graph. The input and output arrays of $\boldsymbol{K_n}$ are passed as input arrays to $\boldsymbol{\hat{K}_n}$. For arrays of $\boldsymbol{A_n}$ that are not input arrays of $\boldsymbol{K_n}$, denoted by the subset $\boldsymbol{A^k_n}$, $\frac{\partial \boldsymbol{K_n}(\boldsymbol{A_n})}{\partial \boldsymbol{A^k_n}} = 0$. Hence, for the corresponding set of arrays, $\boldsymbol{\hat{A}^k_n}$, $\boldsymbol{\hat{A}^k_{n-1}} = \boldsymbol{\hat{A}^k_n}$. For arrays of $\boldsymbol{A_n}$ that are input arrays of $\boldsymbol{K_n}$, denoted by the subset $\boldsymbol{A^l_n}$, the resulting arrays from the kernel $\boldsymbol{\hat{K}_n}$, $\boldsymbol{\hat{K}^l_n}(\boldsymbol{\hat{A}_n}, \boldsymbol{A_n})$, are added to $\boldsymbol{\hat{A}^l_n}$ in order to form the new set of arrays $\boldsymbol{\hat{A}^l_{n+1}}$. Hence, after applying all the adjoint kernels, the correct gradient arrays are obtained.

Application of reverse mode AD to computational graphs can require significant amounts of memory, due to the requirement of storing the linearization of each operation during a forward traversal of the graph. The memory usage of the adjoint method can be reduced by applying the checkpointing method [39] to two-level computational graphs. During the execution of the outer adjoint graph, each adjoint kernel require as inputs, the input arrays from the corresponding kernel of the outer primal graph.



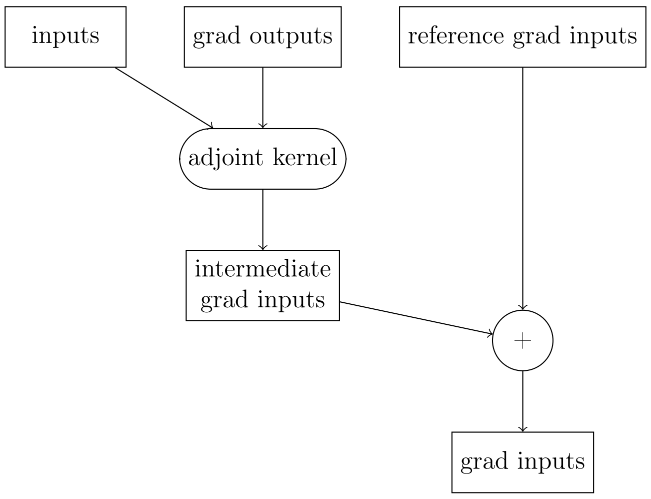


Figure 6-7: Demonstration of how a primal and adjoint kernel are processed by the Python library. The primal kernel processes the input arrays (inputs) and produce intermediate output variables (intermediate outputs) that are added to the output arrays that are passed to them (reference outputs) to generate the final output arrays (outputs). The adjoint kernel follows a similar procedure. “grad" stands for “gradient with respect to".

As the outer adjoint graph operates in the reverse order of the outer primal graph, a part of the outer primal graph needs to be executed before the adjoint kernels in order to precompute all the necessary input arrays from the corresponding primal kernels. For example, in the flow solver graph in Figure 6-3, the arrays Ψ^c_0, Ψ^c_1 and

Ψ_2^c need to be precomputed and stored before executing any of the adjoint kernels. Instead of storing all the intermediate variables of the primal kernels, during the execution of each adjoint kernel they are recomputed in the kernel while propagating the gradient from the output variables to the input variables. Hence, in the execution of the adjoint graph, almost all operations in the primal kernels are executed twice. This makes the two-level adjoint graph procedure more computationally intense than a single-level adjoint graph. But, it leads to a significant reduction in memory usage and access in comparison to a single-level adjoint graph, as the intermediate variables from the primal kernels are not stored in separate arrays.

6.3.2 Verification of adjoint flow solver

The implementation of the adjoint flow solver using the two-level computational graph method is verified by comparing the gradients provided by the adjoint flow solver with those obtained using the finite difference method. The test problem is subsonic flow over a 2-dimensional cylinder at Reynold's number 10,000. This flow problem is similar to the 3-dimensional flow over a cylinder problem described in Section 1.4.2. The mesh contains 92,000 cells. The design objective ($\bar{J}$) is drag over the surface of the cylinder averaged over 0.02 time units, where 1 time unit is the amount of time the flow takes to traverse the cylinder. While the time averaging interval is not sufficient for computing a statistical average of the drag, it is adequate for verifying the adjoint implementation. The governing equations of the flow are perturbed by adding a source term to the density, momentum and energy equations. The source term is non-uniform over the domain of the flow problem. It's shape is a 2-dimension Gaussian function with a center $4d$ upstream of the cylinder and length scale $8d$, where d is the diameter of the cylinder. The scaling of the Gaussian-shaped perturbation serves as the parameter of the system (θ) with respect to which the gradient of the objective is computed. The default scaling parameter is $\theta = 0$.

Figure 6-8 shows the agreement between the one-sided first-order finite difference gradient and the gradient from the adjoint flow solver. The finite difference gradient is computed using the following equation

$$\frac{\partial \bar{J}}{\partial \theta} \sim \frac{\bar{J}(\theta + \Delta\theta) - \bar{J}(\theta)}{\Delta\theta} \tag{6.4}$$

For large perturbations in the scaling parameter ($\Delta\theta$), the relative difference is dominated by the truncation error of the finite difference approximation [67]. As the perturbation in scaling is reduced, this error reduces. But, for even smaller perturbations in scaling parameters ($\Delta\theta < 10$), the round-off error in the finite difference causes the relative difference to increase again.

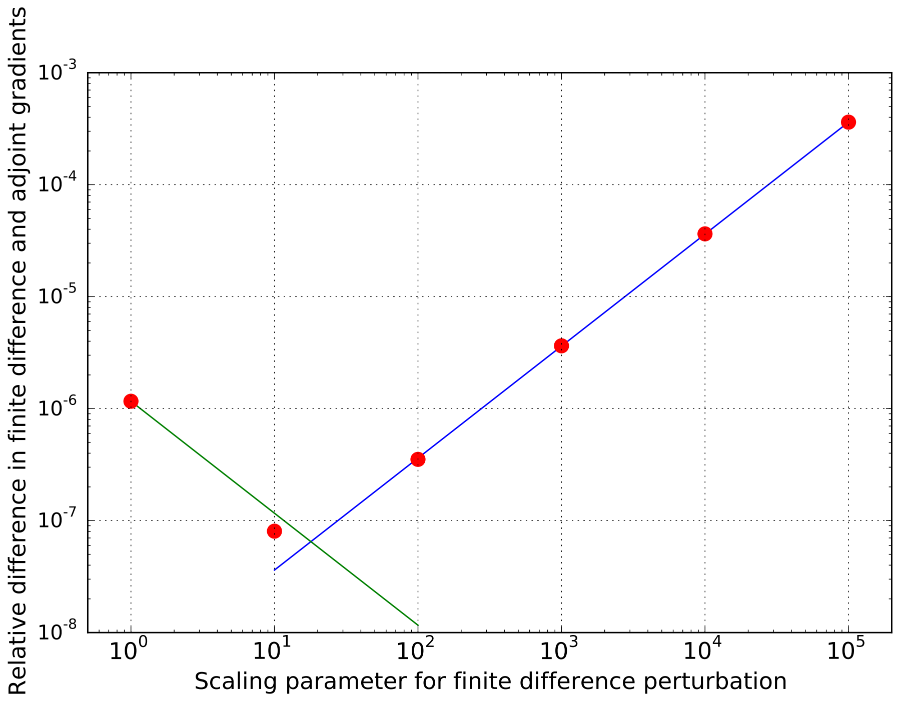


Figure 6-8: Relative difference between gradient computed using the adjoint flow solver and the gradient computed using the finite difference method as a function of the scaling parameter of the perturbation ($\Delta\theta$), denoted by red dots. The green line denotes the slope for the round-off error and the blue line denotes the slope for the truncation error.

In addition to the above test case, the library implementing the two-level computational graph method has many independent unit tests that check various components of the flow solver. These unit tests verify that the gradient provided by the adjoint method matches the gradient provided by the finite difference method up to round-off error. Finally, the implementation of the adjoint flow solver using the two-level computational method is validated by comparing the numerical adjoint solution generated after a couple of time steps on the subsonic flow over 2-dimensional cylinder problem with the numerical adjoint solution generated by an implementation of the adjoint flow solver using an established AD tool (CoDiPack). It is found that the maximum of the relative difference between the two numerical adjoint solutions is close to machine precision.

6.4 Results

The Python library (adFVM), implementing the two-level computational graph method, is used to develop an unsteady flow solver utilizing the FVM for discretization. It has been applied to flow problems with meshes containing 100 millions cells and has been demonstrated to run on 10 thousand cores. The source code of adFVM is available at https://github.com/chaitan3/adFVM.

The efficiency of the two-level computational graph approach is demonstrated by comparing the wall clock time per time step of the primal and adjoint flow solvers with those developed using competing AD libraries. The wall clock measurements are recorded on a 2-dimensional simulation of transonic flow over the turbine vane described in Section 1.4.3. The mesh size for this 2-dimensional flow problem is approximately 77,000 cells. Each of the primal and adjoint flow solvers, developed using the respective AD libraries, utilize the same discretization schemes described in Section 1.3. The flow solvers are executed on a single core of a Intel CPU (Xeon CPU E5-2650) or an Nvidia GPU (GeForce GTX 980), whenever possible. Table 6.1 demonstrates the performance (wall clock time per time step) and memory usage of the flow solvers developed using various AD libraries. It shows that the primal solver built in Python using adFVM is approximately 30% faster than the flow solver written in C++ using CoDiPack [95].

The performance difference can be attributed to adFVM simplifying auto-vectorization for the compiler by combining arithmetic expressions of a kernel in a single function and providing optimized memory management by minimizing number of dynamic memory allocations. The corresponding adjoint solver derived using adFVM is about 3 times faster than the one derived using CoDiPack, which provides operator overloading based AD using expression templates. Typically, a flow solver written in C++ can be transformed to use an AD tool, like CoDiPack, by changing the floating-point variable type used for the numerical flow solution arrays (like $float$ or *double*) to an active variable type (like $codi :: RealReverse$ in CoDiPack). The C++ flow solver code is written using object-oriented programming in a manner similar to the Python flow solver code, without major language-specific optimizations. Even though the C++ code can be optimized further such that the primal solver performance matches that of adFVM, the performance of the adjoint solver will likely remain the same due to the use of operator overloading based AD in CoDiPack.

On CPUs, the primal solver using adFVM is approximately 4 times faster than the primal solver using Theano and the corresponding adjoint solver is 3 times faster. When using GPUs, the two-level computational graph approach of adFVM, with its optimized kernels and memory efficiency, enables an order of magnitude performance gain over the Theano and Tensorflow libraries. Finally, as shown in Table 6.1 the peak memory usage when running the adjoint solver using adFVM on the CPU is considerably lower than CoDiPack, Theano or Tensorflow.

The performance of the flow solver is analyzed on an Intel Xeon CPU E5-2650 running at 2.2 GHz. The CPU uses 128 GB DDR3 RAM running at 1600 MHz in dual channel as main memory. The peak floating point performance of the CPU, when utilizing all the 8 physical cores, is 256 Giga floating point operations per second (Gflops) and the peak DRAM memory (main memory) bandwidth is 51.2 Gigabyte per second (GBps). The measured peak performance of the processor on a practical benchmark, matrix multiplication using Intel MKL, is 20.7 Gigaflops. The measured peak memory bandwidth using a vector addition benchmark tool (*bandwidth*) is 42 GBps. On the turbine vane flow problem, most of the compute time ($\sim 70\%$) of the flow solver is spent in two kernels, the gradient kernel, which computes the gradients of the velocity, temperature and pressure, and the flux kernel which computes the

Library (Version)	Primal CPU (s)	Adjoint CPU (s)	Peak Memory (GB)
CoDiPack (v1.5.0)	0.29	2.93	6.7
Theano (v1.0.1)	0.88	2.64	2.12
Tensorflow (v1.5)	4.62	11.9	2.29
adFVM (v0.2)	0.21	0.96	0.41

Library (Version)	Primal GPU (s)	Adjoint GPU (s)
Theano (v1.0.1)	0.15	14.1
Tensorflow (v1.5)	0.11	0.27
adFVM (v0.2)	0.0065	0.025

Table 6.1: Performance comparison of primal and adjoint flow solvers on a single core of a CPU and a GPU using different AD libraries. The values in the table represent wall clock time of each RK3 time step, where the unit (s) denotes one second. The reason for the slow adjoint solver when using Theano on GPU is an inefficient GPU implementation of the indirect memory access and value modification function in Theano.

momentum and energy fluxes. The average memory bandwidth, Gflops and runtime percentage of the gradient and flux kernels are shown in Table 6.2. These results are obtained by using software performance counters for the kernels. The counters are constructed by tracking the memory loads and stores and floating-point additions and multiplications in the kernel. The measured average bandwidth of the gradient kernel is higher than that of the flux kernel. The flux kernel utilizes a more significant percentage of the runtime, 61.7%, in comparison to the gradient kernel, 9.7%. In addition, an analysis of the kernel functions shows that the flux kernel has a higher flops to memory transfers ratio, 0.8, than the gradient kernel, 0.5. The flow solver is tested on another 8-core Intel CPU (Intel Core i7-7820X) that has a much higher clock rate (4.5 GHz), but approximately the same peak memory bandwidth. On this CPU, the flux kernel achieves 17.1 Gflops, which is more than double the compute performance on the slower CPU. In contrast, the increase in Gflops of the gradient kernel is 50%, which is much less than 100%. This means that the flux kernel is more compute bound than the gradient kernel as it benefits more from a compute performance increase.

Another performance analysis of the flow solver is carried out on a Nvidia GeForce

Kernel	Bandwidth (GBps)	Compute (Gflops)	Percentage of runtime
Gradient	29.8	13.6	9.7%
Flux	16.2	8.1	61.1%

Table 6.2: CPU performance analysis of important kernels

Kernel	Bandwidth (GBps)	Compute (Tflops)	Percentage of runtime
Gradient	180	0.274	27.5%
Flux	211	0.577	38.5%

Table 6.3: GPU performance analysis of important kernels

GTX 980 GPU. It uses the Maxwell architecture, has 16 streaming multiprocessors, 2048 CUDA cores and 4 GB GDDR5 RAM. The GPU has a peak achievable compute performance of 4.6 Teraflops (Tflops) and the peak DRAM memory bandwidth is 224 GBps. The measured peak compute performance is 2.2 Teraflops for single precision on the *matrixMulCUBLAS* benchmark from the Nvidia CUDA SDK. The peak measured memory bandwidth is 150 GBps on the *bandwidthTest* benchmark from the same SDK. The *nvprof* utility offers detailed global memory bandwidth (which includes bandwidth measurements from DRAM and cache memory accesses) and compute performance measurements for every kernel. The gradient kernel for the GPU differs from the one used on the CPU due to the performance characteristics of the CPU and GPU. On the CPU, the gradient kernel loops over all the faces of the mesh and stores the contribution of each face to the gradient in the neighboring cells. The store operation utilizes an indirect memory store. On the GPU, because indirect memory stores are implemented using expensive atomic operations, the gradient kernel processes all the cells of the mesh and directly computes the gradient in each cell by accessing the fields of the neighboring cells. Similar to the CPU case, the gradient and flux computation kernels are the most time consuming kernels. The average memory bandwidth, flops and percentage of runtime of these kernels are shown in Table 6.3 for the two kernels. The average memory bandwidth utilization of both kernels is high, whereas the flops utilization is much below the measured peak. This indicates that both the kernels are primarily memory bandwidth bound. Even though the compute utilization of the flow solver on the GPU is low, it is about 10 times faster than running the flow solver on the 8-core Intel CPU.

The scaling performance of the flow solver is analyzed on subsonic flow in a 3-dimensional periodic box. The flow solver is run on a cluster of nodes with Intel CPUs interconnected using Gigabit Ethernet. There are two types of scaling studies, strong scaling study and weak scaling study. In strong scaling, the number of processors is increased while keeping the number of cells in the mesh constant. Achieving perfect strong scaling means that the runtime of the flow solver linearly decreases with the number of processors on the log scale. In weak scaling, the number of compute nodes is increased while keeping the ratio of the number of cells in the mesh to the number of processors constant. Achieving perfect weak scaling means that the runtime of the flow solver remains the same on increasing grid size.

Figure 6-9 shows the strong scaling performance of the primal and adjoint flow solvers on a mesh with $1,000,000$ cells running on 1, 2, 4, 8 and 16 processors (a single node has 4 processors). The adjoint flow solver achieves closer to optimal scaling as it performs more computations than the primal solver for a given amount of communication. The adjoint flow solver is approximately 3.5 times slower than the primal. Figure 6-10 shows the weak scaling performance of the primal and adjoint flow solvers on meshes with number of cells varying from $100,000$ to $1,600,000$ running on 1, 2, 4, 8 and 16 processors. The number of cells per processor is kept constant at $100,000$. The adjoint flow solver shows better weak scaling than the primal flow solver as it has a higher work to communication ratio.

The scaling performance is further analyzed on the same flow problem running on the Argonne National Lab's Mira supercomputer. The nodes contain IBM PowerPC CPUs which are interconnected using QDR Infiniband. Figure 6-11 shows the strong scaling performance of the primal and adjoint solvers on a mesh with $24,000,000$ cells running on 512, 1024, 2048, 4096 and 8192 processors (a single node is 16 processors with 16 hardware threads). Both flow solvers show near perfect strong scaling up to 3,000 cells per processor. Figure 6-12 shows the weak scaling performance of the primal and adjoint flow solvers on meshes with number of cells varying from $2,500,000$

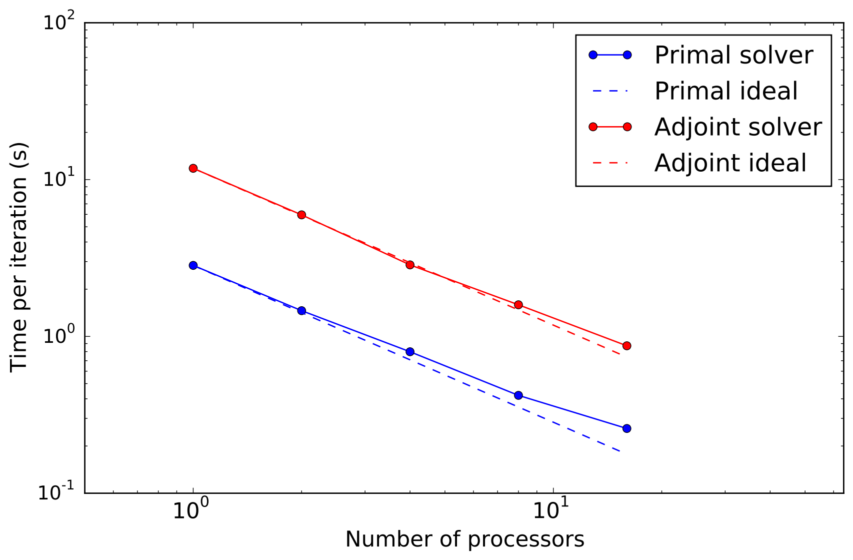


Figure 6-9: Strong scaling study for the box problem on a CPU Ethernet based cluster

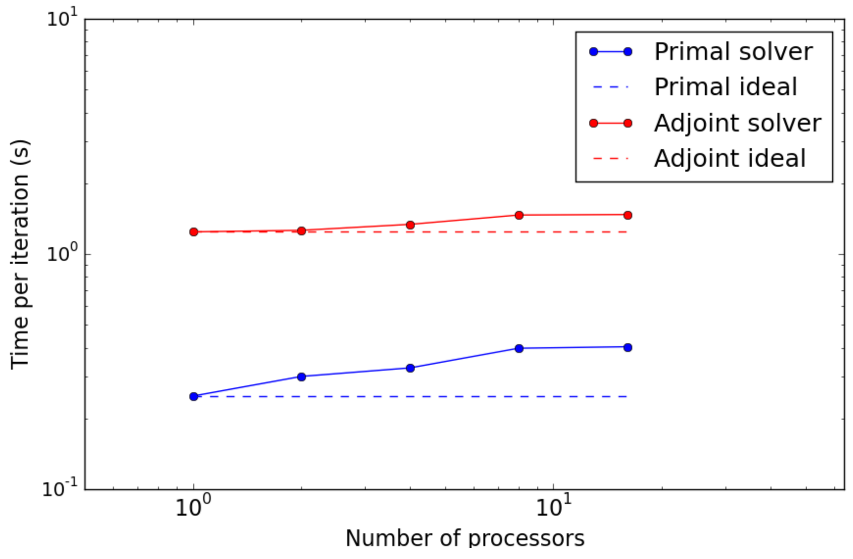


Figure 6-10: Weak scaling study for the box problem on a CPU Ethernet based cluster

to $40,000,000$ running on 512, 1024, 2048, 4096 and 8192 processors. The number of cells per processor is kept constant at approximately 5000. Both flow solvers show perfect weak scaling.

Finally, the scaling performance is analyzed on the same problem running on GPU

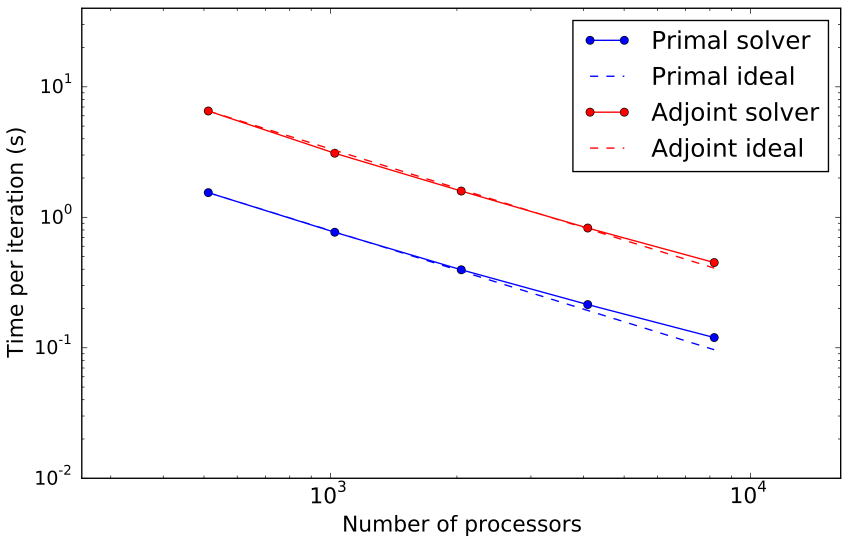


Figure 6-11: Strong scaling study for the box problem on a supercomputer

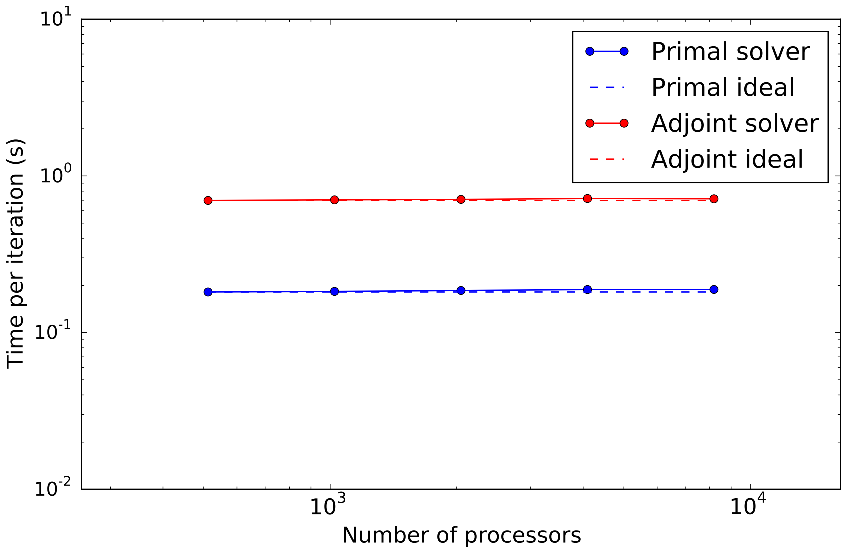


Figure 6-12: Weak scaling study for the box problem on a supercomputer

compute *p*2.*xlarge* instances in the Amazon EC2 Cloud. The nodes contain Tesla K80 GPUs interconnected using high-speed Ethernet. Figure 6-13 shows the strong scaling performance of the primal and adjoint solvers on a mesh with $10,000,000$ cells running on 1, 2, 4, 8 and 16 GPUs (a single node has 1 GPU). The primal solver shows worse scaling than the adjoint solver. Figure 6-14 shows the weak scaling performance of the primal and adjoint flow solvers on meshes with number of cells varying from $2,500,000$ to $40,000,000$ running on 1, 2, 4, 8 and 16 GPUs. The number of cells per GPU is kept constant at $2,500,000$. The relatively poor scaling on GPUs on the Amazon Cloud can be attributed to the higher latency and lower bandwidth of Ethernet in comparison to Infiniband.

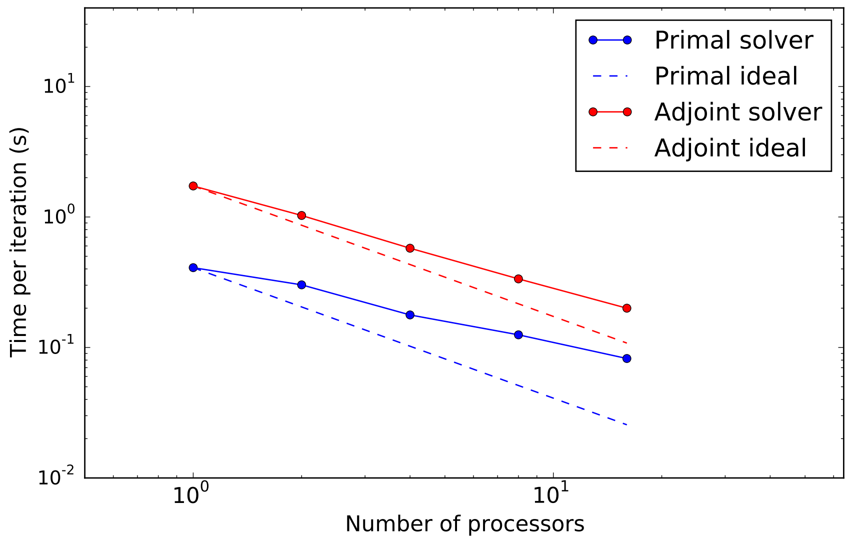


Figure 6-13: Strong scaling study for the box problem on a GPU cloud

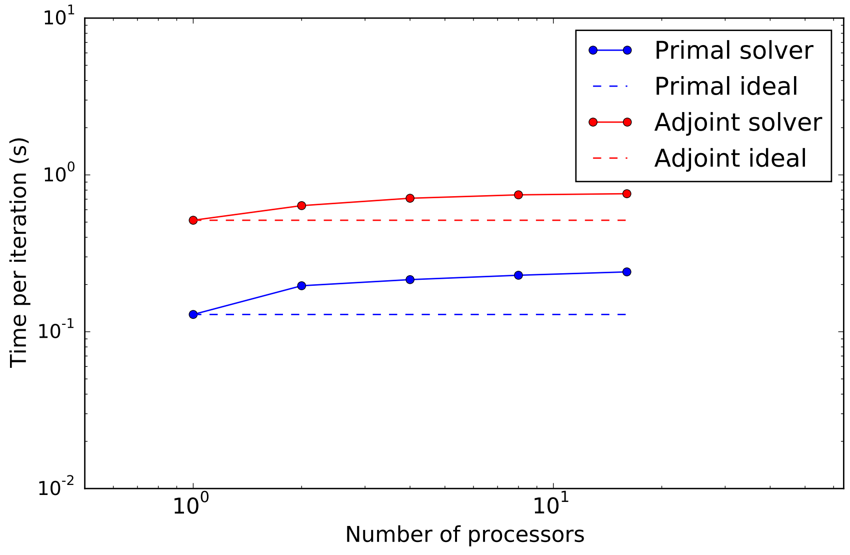


Figure 6-14: Weak scaling study for the box problem on a GPU cloud

Chapter 7

Conclusion

Adjoint-based design optimization using high-fidelity fluid flow simulations is a powerful tool for the design of fluid machinery components. Using a viscosity stabilized adjoint method and a gradient utilizing Bayesian optimization method, the trailing edge of a gas turbine vane was optimized in Chapter 5. The optimal design has a 12% reduction in Nusselt number and 16% reduction in pressure loss coefficient.

The Bayesian optimizer has a few issues scaling to a large number of design parameters (> 20). The number of design points needed to obtain a surrogate model with low error scales exponentially with the number of design parameters. Additionally, optimizing the EI-based metric becomes more difficult with a high number of parameters. Further research needs to be performed in order to increase its efficiency for higher dimensional problems. Alternative optimization algorithms like multi-start stochastic gradient descent should be investigated for design optimization problems with a large number of parameters. The successful application of the adjoint-based design optimization method to the trailing edge shape optimization problem using LES increases its applicability to more challenging design problems like shape design of an entire low-pressure turbine blade.

The chaotic behavior of turbulent fluid flows hampers the utility of the adjoint method. The energy analysis of the adjoint equations, derived in Chapter 2, provides valuable insight into the mechanisms of divergence of the adjoint solution for the compressible Navier-Stokes equations. The viscosity stabilized adjoint method, derived in

Chapter 3, is an inexpensive adjoint method for obtaining a stabilized adjoint solution and reasonable accurate design objective gradients. The effectiveness of this method was demonstrated on a canonical flow problem and an industrial flow problem. The viscosity stabilized adjoint method can be combined with another adjoint method for chaotic systems, the adjoint version of non-intrusive least squares shadowing, in order to reduce its computational cost. Additionally, in Chapter 4 it was shown that the error in the gradient due to the addition of artificial viscosity to the adjoint equations can be bounded by the size of the damping parameter (or scaling factor η).

There are many avenues of additional research for the viscosity stabilized adjoint method. In the energy analysis of the entropy symmetrized form of the adjoint equations, the tensor $\hat{A}^v_{kji}$ is not symmetric in the indices k and i for the Hughes symmetrizing transformation. If there exists another entropy-based symmetrizing transformation that ensures that the tensor is symmetric, then it would allow the construction of an indicator field that takes into consideration the viscous terms from the energy analysis. The existence of such a symmetrizing transformation is an open question. In order to reduce the burden of the numerical implementation of the viscosity stabilized adjoint method, the form of the viscous term modification was kept simple. More complex forms of the modification mentioned in the thesis might result in better stabilization performance and should be explored in the future. As the optimal value of η depends on the Reynolds number of the flow problem, an alternative scaling of the artificial viscosity that depends on the convective scales should be constructed. The error analysis of the viscosity stabilized adjoint method was applied to the discretized flow equations. More insight can be obtained by extending the analysis to the continuous form of the flow equations. Such an analysis would require significant effort for handling the neutral subspace of the adjoint flow operator. The error estimates for the design objective gradient (Equations 4.26 and 4.43) can only be applied to low-dimensional chaotic systems. Efficient algorithms for estimating C_3, C_4 and δ_u, δ_s for high-dimensional chaotic systems need to be devised.

The two-level graph method, described in Chapter 6, for defining the computations in a flow solver increases the performance of the flow solver and reduces memory usage.

The formation of *fat* kernels that require a single loop to process a set of computations for all the elements of the mesh leads to better cache utilization. The computational graph approach simplifies the derivation of the adjoint flow solver with the help of reverse-mode AD. Additionally, it facilitates the creation of asynchronous schedulers that overlap communication with computation and optimal memory management strategies. The developed Python library implements the two-level graph method by providing an abstract and expressive interface for writing high performance structured or unstructured flow solvers. The library hides the low-level implementation details of how to get optimal performance from the available hardware resources. The library can generate code for a large variety of hardware platforms including CPUs and GPUs. Furthermore, it provides kernels for asynchronous communication between different instances of the flow solver for distributed execution. The flow solver scales to thousands of CPUs and tens of GPUs. It is able to utilize a significant fraction of the peak memory bandwidth and compute performance of both CPUs and GPUs. The Python library, adFVM, significantly outperforms state of the art expression template based AD and computational graph based AD libraries.

There are a number of avenues for future research to increase the performance of the flow solver by improving the Python library. A program autotuner, like OpenTuner [4], can be used to optimize the different hardware specific parameters in the library, like the kernel block size for GPUs. Using vectorized memory loads and stores on the GPU can increase the DRAM memory bandwidth utilization of the flow solver. Using AVX instructions on CPU can increase the flops utilization. Implementing an asynchronous scheduler on GPUs for the two-level computational graph can increase performance by allowing multiple kernels and memory transfers to run concurrently on the same device.

Appendix A

Tensors for the compressible Navier-Stokes equations

A.1 Convective flux tensor in conservative formulation

The tensor A_{ijk} in the conservative formulation of the compressible Navier-Stokes equations is given by

$$A_{:1:} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ -u_1^2 + \frac{(\gamma-1)}{2}q^2 & -(\gamma-3)u_1 & -(\gamma-1)u_2 & -(\gamma-1)u_3 & \gamma-1 \\ -u_1u_2 & u_2 & u_1 & 0 & 0 \\ -u_1u_3 & u_3 & 0 & u_1 & 0 \\ -u_1(\gamma E - (\gamma-1)q^2) & \gamma E - (\gamma-1)(u_1^2 - \frac{q^2}{2}) & -(\gamma-1)u_1u_2 & -(\gamma-1)u_1u_3 & \gamma u_1 \end{pmatrix} \tag{A.1}$$

$$A_{:2:} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -u_2u_1 & u_2 & u_1 & 0 & 0 \\ -u_2u_2 + \frac{(\gamma-1)}{2}q^2 & -(\gamma-1)u_1 & -(\gamma-3)u_2 & -(\gamma-1)u_3 & \gamma-1 \\ -u_2u_3 & 0 & u_3 & u_2 & 0 \\ -u_2(\gamma E - (\gamma-1)q^2) & -(\gamma-1)u_2u_1 & \gamma E - (\gamma-1)(u_2^2 - \frac{q^2}{2}) & -(\gamma-1)u_2u_3 & \gamma u_2 \end{pmatrix} \tag{A.2}$$

$$A_{:3:} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ -u_3u_1 & u_3 & 0 & u_1 & 0 \\ -u_3u_2 & 0 & u_3 & u_2 & 0 \\ -u_3u_3 + \frac{(\gamma-1)}{2}q^2 & -(\gamma-1)u_1 & -(\gamma-1)u_2 & -(\gamma-3)u_3 & \gamma-1 \\ -u_3(\gamma E - (\gamma-1)q^2) & -(\gamma-1)u_3u_1 & -(\gamma-1)u_3u_2 & \gamma E - (\gamma-1)(u_3^2 - \frac{q^2}{2}) & \gamma u_3 \end{pmatrix} \tag{A.3}$$

A.2 Viscous flux tensor in conservative formulation

The tensor D_{ijkl} in the conservative formulation of the compressible Navier-Stokes equations is given by

$$D_{:1:1} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -\frac{4}{3}\frac{\mu}{\rho}u_1 & \frac{4}{3}\frac{\mu}{\rho} & 0 & 0 & 0 \\ -\frac{\mu}{\rho}u_2 & 0 & \frac{\mu}{\rho} & 0 & 0 \\ -\frac{\mu}{\rho}u_3 & 0 & 0 & \frac{\mu}{\rho} & 0 \\ \alpha\gamma(-E+q^2) - \frac{\mu}{\rho}(\frac{1}{3}u_1^2 + q^2) & (\frac{4}{3}\frac{\mu}{\rho} - \alpha\gamma)u_2 & (\frac{\mu}{\rho} - \alpha\gamma)u_2 & (\frac{\mu}{\rho} - \alpha\gamma)u_3 & \alpha\gamma \end{pmatrix} \tag{A.4}$$

$$D_{:1:2} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -\frac{\mu}{\rho}u_1 & \frac{\mu}{\rho} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -\frac{\mu}{\rho}u_1u_2 & \frac{\mu}{\rho}u_2 & 0 & 0 & 0 \end{pmatrix} \tag{A.5}$$

$$D_{:2:1} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -\frac{\mu}{\rho}u_2 & 0 & \frac{\mu}{\rho} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -\frac{\mu}{\rho}u_1u_2 & 0 & \frac{\mu}{\rho}u_2 & 0 & 0 \end{pmatrix} \tag{A.6}$$

with similar matrices for $D_{:2:3}, D_{:3:2}, D_{:1:3}$ and $D_{:3:1}$.

A.3 Abarbanel symmetrizing transformation

The linear transformation, S_{ij}, is given by

$$\mathbf{S} = \begin{pmatrix} \frac{c}{\sqrt{\gamma}\rho} & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\frac{c}{\rho\sqrt{\gamma(\gamma-1)}} & 0 & 0 & 0 & \sqrt{\frac{\gamma}{\gamma-1}}\frac{1}{\rho c} \end{pmatrix} \tag{A.7}$$

The linear transformation, V_{jk}, is given by

$$\mathbf{V} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{u_1}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{u_2}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{u_3}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ \frac{(\gamma-1)u_i u_i}{2} & -(\gamma-1)u_1 & -(\gamma-1)u_2 & -(\gamma-1)u_3 & (\gamma-1) \end{pmatrix} \tag{A.8}$$

The tensor $\hat{A}_{ijk}$ is given by the following 3 matrices

$$\hat{A}_{:1:} = \begin{pmatrix} u_1 & \frac{c}{\sqrt{\gamma}} & 0 & 0 & 0 \\ \frac{c}{\sqrt{\gamma}} & u_1 & 0 & 0 & \sqrt{\frac{\gamma-1}{\gamma}}c \\ 0 & 0 & u_1 & 0 & 0 \\ 0 & 0 & 0 & u_1 & 0 \\ 0 & \sqrt{\frac{\gamma-1}{\gamma}}c & 0 & 0 & u_1 \end{pmatrix} \tag{A.9}$$

$$\hat{A}_{:2:} = \begin{pmatrix} u_2 & 0 & \frac{c}{\sqrt{\gamma}} & 0 & 0 \\ 0 & u_2 & 0 & 0 & 0 \\ \frac{c}{\sqrt{\gamma}} & 0 & u_2 & 0 & \sqrt{\frac{\gamma-1}{\gamma}}c \\ 0 & 0 & 0 & u_2 & 0 \\ 0 & 0 & \sqrt{\frac{\gamma-1}{\gamma}}c & 0 & u_2 \end{pmatrix} \tag{A.10}$$

$$\hat{A}_{:3:} = \begin{pmatrix} u_3 & 0 & 0 & \frac{c}{\sqrt{\gamma}} & 0 \\ 0 & u_3 & 0 & 0 & 0 \\ 0 & 0 & u_3 & 0 & 0 \\ \frac{c}{\sqrt{\gamma}} & 0 & 0 & u_3 & \sqrt{\frac{\gamma-1}{\gamma}}c \\ 0 & 0 & 0 & \sqrt{\frac{\gamma-1}{\gamma}}c & u_3 \end{pmatrix} \tag{A.11}$$

A.4 Turkel symmetrizing transformation

The linear transformation, S_{ij}, is given by

$$\mathbf{S} = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{\rho c} \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\frac{\gamma p}{\rho} & 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.12}$$

The linear transformation, T_{ik}, is given by

$$\mathbf{T} = \begin{pmatrix} (\gamma-1)\frac{u_i u_i}{2\rho c} & -(\gamma-1)\frac{u_1}{\rho c} & -(\gamma-1)\frac{u_2}{\rho c} & 0-(\gamma-1)\frac{u_3}{\rho c} & (\gamma-1)\frac{1}{\rho c} \\ -\frac{u_1}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{u_2}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{u_3}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ \frac{(\gamma-1)u_i u_i}{2} - c^2 & -(\gamma-1)u_1 & -(\gamma-1)u_2 & -(\gamma-1)u_3 & (\gamma-1) \end{pmatrix} \tag{A.13}$$

The tensor $\hat{A}_{ijk}$ is given by the following 3 matrices

$$\hat{A}_{:1:} = \begin{pmatrix} u_1 & c & 0 & 0 & 0 \\ c & u_1 & 0 & 0 & 0 \\ 0 & 0 & u_1 & 0 & 0 \\ 0 & 0 & 0 & u_1 & 0 \\ 0 & 0 & 0 & 0 & u_1 \end{pmatrix} \tag{A.14}$$

$$\hat{A}_{:1:} = \begin{pmatrix} u_2 & 0 & c & 0 & 0 \\ 0 & u_2 & 0 & 0 & 0 \\ c & 0 & u_2 & 0 & 0 \\ 0 & 0 & 0 & u_2 & 0 \\ 0 & 0 & 0 & 0 & u_2 \end{pmatrix} \tag{A.15}$$

$$\hat{A}_{:1:} = \begin{pmatrix} u_3 & 0 & 0 & c & 0 \\ 0 & u_3 & 0 & 0 & 0 \\ 0 & 0 & u_3 & 0 & 0 \\ c & 0 & 0 & u_3 & 0 \\ 0 & 0 & 0 & 0 & u_3 \end{pmatrix} \tag{A.16}$$

A.5 Hughes symmetrizing transformation

Using the following notation

$$\begin{aligned} &k_1 = \frac{1}{2}\frac{(v_2^2+v_3^2+v_4^2)}{v_5},\ \ k_2 = k_1 - \gamma,\ \ k_3 = k_1^2 - 2\gamma k_1 + \gamma \\ &k_4 = k_2 - \gamma + 1,\ \ k_5 = k_2^2 - (\gamma-1)(k_1+k_2) \\ &c_1 = (\gamma-1)v_5 - v_2^2,\ \ d_1 = -v_2 v_3,\ \ e_1 = v_2 v_5 \\ &c_2 = (\gamma-1)v_5 - v_2^2,\ \ d_2 = -v_2 v_4,\ \ e_2 = v_3 v_5 \\ &c_3 = (\gamma-1)v_5 - v_2^2,\ \ d_3 = -v_3 v_4,\ \ e_3 = v_4 v_5 \end{aligned} \tag{A.17}$$

the symmetrized tensors $\hat{A}_{ijm}$ and $\hat{D}_{ijml}$ are

$$\hat{A}_{:1:} = \frac{\rho e}{(\gamma-1)v_5^2}\begin{pmatrix} e_1 v_5 & c_1 v_5 & d_1 v_5 & d_2 v_5 & k_2 e_1 \\ * & -(c_1 + 2(\gamma-1)v_5)v_2 & -c_1 v_3 & -c_1 v_4 & c_1 k_2 + (\gamma-1)v_2^2 \\ * & * & -c_2 v_2 & -d_1 v_4 & k_4 d_1 \\ * & * & * & -c_3 v_2 & k_4 d_2 \\ * & * & * & * & k_5 v_2 \end{pmatrix} \tag{A.18}$$

$$\hat{A}_{:2:} = \frac{\rho e}{(\gamma-1)v_5^2}\begin{pmatrix} e_2 v_5 & d_1 v_5 & c_2 v_5 & d_3 v_5 & k_2 e_2 \\ * & -c_1 v_3 & -c_2 v_2 & -d_1 v_4 & k_4 d_1 \\ * & * & -(c_2 + 2(\gamma-1)v_5)v_3 & -c_2 v_4 & c_2 k_2 + (\gamma-1)v_3^2 \\ * & * & * & -c_3 v_3 & k_4 d_3 \\ * & * & * & * & k_5 v_3 \end{pmatrix} \tag{A.19}$$

$$\hat{A}_{:3:} = \frac{\rho e}{(\gamma-1)v_5^2}\begin{pmatrix} e_3 v_5 & d_2 v_5 & d_3 v_5 & c_3 v_5 & k_2 e_3 \\ * & -c_1 v_4 & -d_2 v_3 & -c_3 v_2 & k_4 d_2 \\ * & * & -c_2 v_4 & -c_3 v_3 & k_4 d_3 \\ * & * & * & -(c_3 + 2(\gamma-1)v_5)v_4 & c_3 k_2 + (\gamma-1)v_4^2 \\ * & * & * & * & k_5 v_4 \end{pmatrix} \tag{A.20}$$

$$D_{:1:1} = \frac{1}{v_5^3}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ * & -\frac{4}{3}\mu v_5^2 & 0 & 0 & \frac{4}{3}\mu e_1 \\ * & * & -\mu v_5^2 & 0 & \mu e_2 \\ * & * & * & -\mu v_5^2 & \mu e_3 \\ * & * & * & * & -[\frac{4}{3}\mu v_2^2 + \mu(v_3^2 + v_4^2 - \gamma\alpha\rho v_5)] \end{pmatrix} \tag{A.21}$$

$$D_{:2:2} = \frac{1}{v_5^3}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ * & -\mu v_5^2 & 0 & 0 & \mu e_1 \\ * & * & -\frac{4}{3}\mu v_5^2 & 0 & \frac{4}{3}\mu e_2 \\ * & * & * & -\mu v_5^2 & \mu e_3 \\ * & * & * & * & -[\frac{4}{3}\mu v_3^2 + \mu(v_2^2 + v_4^2 - \gamma\alpha\rho v_5)] \end{pmatrix} \tag{A.22}$$

$$D_{:3:3} = \frac{1}{v_5^3}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ * & -\mu v_5^2 & 0 & 0 & \mu e_1 \\ * & * & -\mu v_5^2 & 0 & \mu e_2 \\ * & * & * & -\frac{4}{3}\mu v_5^2 & \frac{4}{3}\mu e_3 \\ * & * & * & * & -[\frac{4}{3}\mu v_4^2 + \mu(v_3^2 + v_2^2 - \gamma\alpha\rho v_5)] \end{pmatrix} \tag{A.23}$$

$$D_{:1:2} = \frac{1}{v_5^3}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{2}{3}\mu v_5^2 & 0 & -\frac{2}{3}\mu e_2 \\ 0 & -\mu v_5^2 & 0 & 0 & \mu e_1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \mu e_2 & -\frac{2}{3}\mu e_1 & 0 & \frac{1}{3}\mu d_1 \end{pmatrix} \tag{A.24}$$

$$D_{:1:3} = \frac{1}{v_5^3}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{2}{3}\mu v_5^2 & -\frac{2}{3}\mu e_3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -\mu v_5^2 & 0 & 0 & \mu e_1 \\ 0 & \mu e_3 & 0 & -\frac{2}{3}\mu e_1 & \frac{1}{3}\mu d_2 \end{pmatrix} \tag{A.25}$$

$$D_{:2:3} = \frac{1}{v_5^3}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3}\mu v_5^2 & -\frac{1}{3}\mu e_3 \\ 0 & 0 & -\mu v_5^2 & 0 & \mu e_2 \\ 0 & 0 & \mu e_3 & -\frac{2}{3}\mu e_2 & \frac{1}{3}\mu d_3 \end{pmatrix} \tag{A.26}$$

(A.27)

The sign $(*)$ denotes the fact that the entry is the same as the entry in the matrix with its indices swapped.

Appendix B

Analysis of error due to sparse addition of artificial viscosity

Adding b times artificial viscosity every b time steps, where b is a positive integer, introduces an additional error in the numerical adjoint solution of the modified adjoint equations. The addition of artificial viscosity is modelled using the following equation

$$\frac{dx}{dt} = ax \tag{B.1}$$

where x is a scalar and a is a constant.

Using the implicit Euler integration method, when adding viscosity at every time step, the solution at a time step $n+1$ can be written as

$$x(t_{n+1}) = \frac{1}{1 - a\Delta t} x(t_n) + a^2 O(\Delta t^2) \tag{B.2}$$

where $\Delta t = t_{n+1} - t_n$ is the time step. Using a constant time step and representing $x(t_{n+1})$ as x_{n+1}

$$x_{n+1} = \frac{1}{1 - a\Delta t} x_n + a^2 O(\Delta t^2) \tag{B.3}$$

Using the above equation for b time steps

$$x_{n+b} = \frac{1}{(1 - a\Delta t)^b} x_n + a^2 O(\Delta t^2) \tag{B.4}$$

For small time steps ($a\Delta t \ll 1$), the above equation can be simplified to

$$x_{n+b} = (1 + ba\Delta t)x_n + (ba)^2 O(\Delta t^2) \tag{B.5}$$

Using the implicit Euler time integration method, when adding b times the viscosity at every b time steps, the solution at time step $n + b$ can be written as

$$x_{n+b} = \frac{1}{1 - ba\Delta t} x_n + (ba)^2 O(\Delta t^2) \tag{B.6}$$

For small time steps ($ba\Delta t \ll 1$), the above equation can be simplified to

$$x_{n+b} = (1 + ba\Delta t)x_n + (ba)^2 O(\Delta t^2) \tag{B.7}$$

Equations B.5 and B.7 show that the order of magnitude of the error in the solution for the two different methods of adding viscosity is the same for small time steps. Hence, adding b times the artificial viscosity every b time steps does not lead to a large increase in the error of the adjoint solution for the modified adjoint equations.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Saul Abarbanel and David Gottlieb. Optimal time splitting for two-and three-dimensional Navier-Stokes equations with mixed derivatives. *Journal of Computational Physics*, 41(1):1–33, 1981.

[3] Rafail V Abramov and Andrew J Majda. Blended response algorithms for linear fluctuation-dissipation for complex nonlinear dynamical systems. *Nonlinearity*, 20(12):2793, 2007.

[4] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 303–316. ACM, 2014.

[5] Tony Arts and M Lambert de Rouvroit. Aero-thermal performance of a two-dimensional highly loaded transonic turbine nozzle guide vane: A test case for inviscid and viscous flow computations. *J. Turbomach.*, 114(1):147–154, 1992.

[6] Pierre Aubert, Nicolas Di Césaré, and Olivier Pironneau. Automatic differentiation in C++ using expression templates and. application to a flow control problem. *Computing and Visualization in Science*, 3(4):197–208, Jan 2001.

[7] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU math compiler in Python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[8] Rathakrishnan Bhaskaran and Sanjiva K Lele. Heat Transfer Prediction in High Pressure Turbine Cascade with Free-Stream Turbulence using LES. *AIAA Paper*, (2011-3266), 2011.

[9] Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland. ADIFOR–generating derivative codes from Fortran programs. *Scientific Programming*, 1(1):11–29, 1992.

[10] Christian H Bischof, Mohammad R Haghighat, et al. Hierarchical approaches to automatic differentiation. *Computational Differentiation: Techniques, Applications, and Tools*, pages 83–94, 1996.

[11] Christian H Bischof, Paul D Hovland, and Boyana Norris. On the implementation of automatic differentiation tools. *Higher-Order and Symbolic Computation*, 21(3):311–331, 2008.

[12] Patrick Blonigan. *Least Squares Shadowing for sensitivity analysis of large chaotic systems and fluid flows.* PhD thesis, Massachusetts Institute of Technology, 2016.

[13] Patrick J Blonigan. Adjoint sensitivity analysis of chaotic dynamical systems with non-intrusive least squares shadowing. *Journal of Computational Physics*, 348:803–826, 2017.

[14] PJ Blonigan, R Chen, Q Wang, and J Larsson. Towards adjoint sensitivity analysis of statistics in turbulent flow simulation. In *Proceedings of the Stanford Center of Turbulence Research Summer Program 2014*, page 229, 2012.

[15] Emilio F. Campana, Daniele Peri, Yusuke Tahara, and Frederick Stern. Shape optimization in ship hydrodynamics using computational fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, 196(1):634 – 651, 2006.

[16] Carlos Castro, Carlos Lozano, Francisco Palacios, and Enrique Zuazua. Systematic continuous adjoint approach to viscous aerodynamic design on unstructured grids. *AIAA journal*, 45(9):2125, 2007.

[17] Daniele Cavaglieri, Pooriya Beyhaghi, and Thomas Bewley. Low-storage IMEX Runge-Kutta schemes for the simulations of Navier-Stokes systems. In *21st AIAA Computational Fluid Dynamics Conference*, page 3094, 2013.

[18] Haecheon Choi and Parviz Moin. Grid-point requirements for large eddy simulation: ChapmanâĂŹs estimates revisited. *Physics of Fluids (1994-present)*, 24(1):011702, 2012.

[19] Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 225–240. Springer, 2013.

[20] Richard Courant, Kurt Friedrichs, and Hans Lewy. On the partial difference equations of mathematical physics. *IBM journal of Research and Development*, 11(2):215–234, 1967.

[21] Laslo T Diosady and Scott M Murman. Higher-order methods for compressible turbulent flows using entropy variables. In *53rd AIAA Aerospace Sciences Meeting*, page 0294, 2015.

[22] Richard P Dwight. Robust mesh deformation using the linear elasticity equations. *Computational Fluid Dynamics 2006*, pages 401–406, 2009.

[23] J-P Eckmann and David Ruelle. Ergodic theory of chaos and strange attractors. In *The Theory of Chaotic Attractors*, pages 273–312. Springer, 1985.

[24] Thomas D Economon, Francisco Palacios, and Juan J Alonso. A viscous continuous adjoint approach for the design of rotating engineering applications. *AIAA Paper*, 2580:24–27, 2013.

[25] GL Eyink, TWN Haine, and DJ Lea. Ruelle's linear response formula, ensemble adjoint schemes and Lévy flights. *Nonlinearity*, 17(5):1867, 2004.

[26] Pablo Fernandez and Qiqi Wang. Lyapunov spectrum of scale-resolving turbulent simulations. Application to chaotic adjoints. In *23rd AIAA Computational Fluid Dynamics Conference*, page 3797, 2017.

[27] Alexander I.J Forrester, Neil W Bressloff, and Andy J Keane. Optimization using surrogate models and partially converged computational fluid dynamics simulations. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 462(2071):2177–2204, 2006.

[28] P Fosso, Hugues Deniau, Nicolas Lamarque, Thierry Poinsot, et al. Comparison of outflow boundary conditions for subsonic aeroacoustic simulations. *International journal for numerical methods in fluids*, 68(10):1207–1233, 2012.

[29] Eric Garnier, Nikolaus Adams, and Pierre Sagaut. *Large eddy simulation for compressible flows.* Springer Science & Business Media, 2009.

[30] Apostolos Gerasoulis and Tao Yang. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Computing*, 16(4):276–291, 1992.

[31] Michael B Giles and Niles A Pierce. Adjoint equations in CFD: duality, boundary conditions and solution behaviour. *AIAA paper*, 1850:1997, 1997.

[32] Michael B Giles and Niles A Pierce. An introduction to the adjoint approach to design. *Flow, turbulence and combustion*, 65(3-4):393–415, 2000.

[33] Michael B Giles and Endre Süli. Adjoint methods for PDEs: a posteriori error analysis and postprocessing by duality. *Acta numerica*, 11:145–236, 2002.

[34] Francesco Ginelli, P Poggi, A Turchi, H Chaté, R Livi, and A Politi. Characterizing dynamics with covariant Lyapunov vectors. *Physical review letters*, 99(13):130601, 2007.

[35] David Ginsbourger, Rodolphe Le Riche, Laurent Carraro, et al. A multi-points criterion for deterministic parallel global optimization based on Gaussian processes. *J. Global Optim., in revision*, 2009.

[36] Nicolas Gourdain, Laurent YM Gicquel, and Elena Collado. Comparison of RANS and LES for prediction of wall heat transfer in a highly loaded turbine guide vane. *Journal of Propulsion and Power*, 28(2):423–433, 2012.

[37] Edward M Greitzer, Choon Sooi Tan, and Martin B Graf. *Internal flow: concepts and applications*, volume 3. Cambridge University Press, 2007.

[38] Andreas Griewank. A mathematical view of automatic differentiation. *Acta Numerica*, 12:321–398, 2003.

[39] Andreas Griewank and Andrea Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):19–45, 2000.

[40] John Guckenheimer and Robert F Williams. Structural stability of Lorenz attractors. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 50(1):59–72, 1979.

[41] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[42] Je-Chin Han, Sandip Dutta, and Srinath Ekkad. *Gas turbine heat transfer and cooling technology*. CRC Press, 2012.

[43] Laurent Hascoet and Valérie Pascual. The Tapenade Automatic Differentiation tool: principles, model, and specification. *ACM Transactions on Mathematical Software (TOMS)*, 39(3):20, 2013.

[44] Boris Hasselblatt. Hyperbolic dynamical systems. *Handbook of dynamical systems*, 1:239–319, 2002.

[45] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012.

[46] Robin J Hogan. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM Transactions on Mathematical Software (TOMS)*, 40(4):26, 2014.

[47] Thomas JR Hughes, LP Franca, and M Mallet. A new finite element formulation for computational fluid dynamics: I. Symmetric forms of the compressible Euler and Navier-Stokes equations and the second law of thermodynamics. *Computer Methods in Applied Mechanics and Engineering*, 54(2):223–234, 1986.

[48] Waltraud Huyer and Arnold Neumaier. SNOBFIT–stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software (TOMS)*, 35(2):9, 2008.

[49] Antony Jameson. Optimum aerodynamic design using CFD and control theory. *AIAA paper*, 1729:124–131, 1995.

[50] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[51] Soshi Kawai and Sanjiva K Lele. Large-eddy simulation of jet mixing in supersonic crossflows. *AIAA journal*, 48(9):2063–2083, 2010.

[52] Steven Kelly and Juha-Pekka Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.

[53] Kristian Kersting, Christian Plagemann, Patrick Pfaff, and Wolfram Burgard. Most likely heteroscedastic Gaussian process regression. In *Proceedings of the 24th international conference on Machine learning*, pages 393–400. ACM, 2007.

[54] Arthur G Kravchenko and Parviz Moin. Numerical studies of flow over a circular cylinder at Re D= 3900. *Physics of fluids*, 12(2):403–417, 2000.

[55] Pavel V Kuptsov and Ulrich Parlitz. Theory and computation of covariant Lyapunov vectors. *Journal of nonlinear science*, 22(5):727–762, 2012.

[56] Guillaume Lapeyre. Characterization of finite-time Lyapunov exponents and vectors in two-dimensional turbulence. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 12(3):688–698, 2002.

[57] Gerald John Le Beau, SE Ray, SK Aliabadi, and TE Tezduyar. SUPG finite element computation of compressible flows with the entropy and conservation variables formulations. *Computer Methods in Applied Mechanics and Engineering*, 104(3):397–422, 1993.

[58] Daniel J Lea, Myles R Allen, and Thomas WN Haine. Sensitivity analysis of the climate of a chaotic system. *Tellus A*, 52(5):523–532, 2000.

[59] Randall J LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[60] Elliott H Lieb. On characteristic exponents in turbulence. *Communications in Mathematical Physics*, 92(4):473–480, 1984.

[61] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141, 1963.

[62] Zhoujie Lyu and Joaquim RRA Martins. Aerodynamic design optimization studies of a blended-wing-body aircraft. *Journal of Aircraft*, 51(5):1604–1617, 2014.

[63] colin barr macdonald. *constructing high-order runge-kutta methods with embedded strong-stability-preserving pairs*. PhD thesis, simon fraser university, 2003.

[64] Alison L Marsden, Meng Wang, JE Dennis, and Parviz Moin. Trailing-edge noise reduction using derivative-free optimization and large-eddy simulation. *Journal of Fluid Mechanics*, 572:13–36, 2007.

[65] Andre C Marta, Sriram Shankaran, Qiqi Wang, and Prem Venugopal. Interpretation of adjoint solutions for turbomachinery flows. *AIAA journal*, 2013.

[66] G Medic, J Joo, SK Lele, and OP Sharma. Prediction of heat transfer in a turbine cascade with high levels of free-stream turbulence. In *Proceedings of the Summer Program*, page 147, 2012.

[67] William Edmund Milne and WE Milne. *Numerical solution of differential equations*, volume 19. Wiley New York, 1953.

[68] Chin-Hoh Moeng and John C Wyngaard. Evaluation of turbulent transport and dissipation closures in second-order modeling. *Journal of the Atmospheric Sciences*, 46(14):2311–2330, 1989.

[69] Parviz Moin and John Kim. Tackling turbulence with supercomputers. *Scientific American*, 276(1):46–52, 1997.

[70] E Collado Morata, N Gourdain, F Duchaine, and LYM Gicquel. Effects of free-stream turbulence on high pressure turbine blade heat transfer predicted by structured and unstructured LES. *International Journal of Heat and Mass Transfer*, 55(21):5754–5768, 2012.

[71] Siva Nadarajah and Antony Jameson. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. *AIAA paper*, 667(2000):3–17, 2000.

[72] Siva Nadarajah and Antony Jameson. Studies of the continuous and discrete adjoint approaches to viscous automatic aerodynamic shape optimization. In *15th AIAA Computational Fluid Dynamics Conference*, page 2530, 2001.

[73] Angxiu Ni. Sensitivity analysis on chaotic dynamical systems by Non-Intrusive Least Squares Adjoint Shadowing (NILSAS). *arXiv preprint arXiv:1801.08674*, 2018.

[74] Angxiu Ni and Qiqi Wang. Sensitivity analysis on chaotic dynamical systems by Non-Intrusive Least Squares Shadowing (NILSS). *Journal of Computational Physics*, 2017.

[75] T. Oliver, N. Malaya, R. Ulerich, and R. Moser. Estimating uncertainties in statistics computed from direct numerical simulation. *Phys. Fluids*, 26, 2014.

[76] Todd A Oliver, Nicholas Malaya, Rhys Ulerich, and Robert D Moser. Estimating uncertainties in statistics computed from direct numerical simulation. *Physics of Fluids*, 26(3):035101, 2014.

[77] Valery Iustinovich Oseledets. A multiplicative ergodic theorem. Characteristic Ljapunov, exponents of dynamical systems. *Trudy Moskovskogo Matematicheskogo Obshchestva*, 19:179–210, 1968.

[78] DI Papadimitriou and KC Giannakoglou. A continuous adjoint method with objective function derivatives based on boundary integrals, for inviscid and viscous flows. *Computers & Fluids*, 36(2):325–341, 2007.

[79] DI Papadimitriou and KC Giannakoglou. Total pressure loss minimization in turbomachinery cascades using a new continuous adjoint formulation. *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 221(6):865–872, 2007.

[80] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[81] Victor Picheny, David Ginsbourger, and Yann Richet. Noisy expected improvement and on-line computation time allocation for the optimization of simulators with tunable fidelity. 2010.

[82] Victor Picheny, David Ginsbourger, Yann Richet, and Gregory Caplin. Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics*, 55(1):2–13, 2013.

[83] T J&amp Poinsot and SK Lelef. Boundary conditions for direct simulations of compressible viscous flows. *Journal of computational physics*, 101(1):104–129, 1992.

[84] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[85] Luc Pronzato and Werner G Müller. Design of computer experiments: space filling and beyond. *Statistics and Computing*, 22(3):681–701, 2012.

[86] Carl Rasmussen and Chris Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

[87] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. *J. Global Optim.*, 56(3):1247–1293, 2013.

[88] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[89] Philip L Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of computational physics*, 43(2):357–372, 1981.

[90] Anatol Roshko. Experiments on the flow past a circular cylinder at very high Reynolds number. *Journal of Fluid Mechanics*, 10(3):345–356, 1961.

[91] David Ruelle. Large volume limit of the distribution of characteristic exponents in turbulence. *Communications in Mathematical Physics*, 87(2):287–302, 1982.

[92] David Ruelle. Differentiation of SRB states. *Communications in Mathematical Physics*, 187(1):227–241, 1997.

[93] David Ruelle. Differentiation of SRB states for hyperbolic flows. *Ergodic Theory and Dynamical Systems*, 28(02):613–631, 2008.

[94] David Ruelle. A review of linear response theory for general differentiable dynamical systems. *Nonlinearity*, 22(4):855, 2009.

[95] Max Sagebaum, Tim Albring, and Nicolas R Gauger. High-Performance Derivative Computations using CoDiPack. *arXiv preprint arXiv:1709.07229*, 2017.

[96] Jamshid A Samareh. Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization. *AIAA journal*, 39(5):877–884, 2001.

[97] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Adv. Neur. In.*, pages 2951–2959, 2012.

[98] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[99] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering.* Westview press, 2014.

[100] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw Bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

[101] Barna Aladar Szabo and Ivo BabuÂška. *Finite element analysis.* John Wiley & Sons, 1991.

[102] C Talnikar, P Blonigan, J Bodart, and Q Wang. Parallel Optimization for LES. In *Proceedings of the Summer Program*, page 315, 2014.

[103] Chaitanya Talnikar, Qiqi Wang, and Gregory Laskowski. Unsteady adjoint of pressure loss for a fundamental transonic turbine vane. *Journal of Turbomachinery*, 2016.

[104] Chaitanya A Talnikar, Patrick J Blonigan, Julien Bodart, and Qiqi Wang. Optimization with LES–algorithms for dealing with sampling error of turbulence statistics. In *53rd AIAA Aerospace Sciences Meeting*, page 1954, 2015.

[105] Kevin W Thompson. Time dependent boundary conditions for hyperbolic systems. *Journal of computational physics*, 68(1):1–24, 1987.

[106] J Thuburn. Climate sensitivities via a Fokker–Planck adjoint approach. *Quarterly Journal of the Royal Meteorological Society*, 131(605):73–92, 2005.

[107] D_J Tritton. Experiments on the flow past a circular cylinder at low Reynolds numbers. *Journal of Fluid Mechanics*, 6(4):547–567, 1959.

[108] Eli Turkel. Symmetrization of the fluid dynamic matrices with applications. *Mathematics of computation*, 27(124):729–736, 1973.

[109] Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch. OpenAD/F: A modular open-source tool for automatic differentiation of Fortran codes. *ACM Transactions on Mathematical Software (TOMS)*, 34(4):18, 2008.

[110] Emmanuel Vazquez and Julien Bect. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and inference*, 140(11):3088–3095, 2010.

[111] V Venkatakrishnan and Dimitri J Mavriplis. Implicit solvers for unstructured meshes. *Journal of computational Physics*, 105(1):83–91, 1993.

[112] HK Versteeg and W Malalasekera. Computational fluid dynamics. *The finite volume method*, 1995.

[113] DX Wang and L He. Adjoint aerodynamic design optimization for blades in multistage turbomachinesâĂŤpart i: Methodology and verification. *Journal of Turbomachinery*, 132(2):021011, 2010.

[114] Qiqi Wang. *Uncertainty quantification for unsteady fluid flow using adjoint-based approaches*. Stanford University, 2009.

[115] Qiqi Wang. Convergence of the least squares shadowing method for computing derivative of ergodic averages. *SIAM Journal on Numerical Analysis*, 52(1):156–170, 2014.

[116] Qiqi Wang and Jun-Hui Gao. The drag-adjoint field of a circular cylinder wake at Reynolds numbers 20, 100 and 500. *Journal of Fluid Mechanics*, 730:145–161, 2013.

[117] Qiqi Wang, Rui Hu, and Patrick Blonigan. Least squares shadowing sensitivity analysis of chaotic limit cycle oscillations. *Journal of Computational Physics*, 267:210–224, 2014.

[118] Carl Wieselsberger. New data on the laws of fluid resistance. 1922.

[119] Jian Wu, Matthias Poloczek, Andrew Gordon Wilson, and Peter I Frazier. Bayesian Optimization with Gradients. *arXiv preprint arXiv:1703.04389*, 2017.